Why not just require each application to declare what version of Windows it is compatible with?

devblogs.microsoft.com/oldnewthing/20100927-00

September 27, 2010

Raymond Chen

Via the Suggestion Box, Arno Shoedl asked, "could not a lot of compatibility problems be solved by simply declaring (via manifest?) the earliest and latest version of Windows a program has been tested to run on?" Actually, programs already declare that, sort of. Each module has a subsystem field in the header that specifies the earliest version of Windows the program will run on. There isn't a corresponding way to declare the maximum version of Windows you want to run on, however, so the manifesty thing could be done there. (There is a new manifesty way of saying what version of Windows you would like to see.) But let's look at the bigger picture: In order to benefit from this proposed feature, an application author would have to do some extra work to declare in a manifest what versions of Windows they prefer to run on. That's already a bunch of work that gets in the way (because to an application author, any amount of work greater than zero that doesn't correspond to business logic is "a bunch of work that gets in the way"). Which means most people won't bother doing it. So you're back to where you started. Even if we could somehow convince application authors to get off their butts and do things that do not benefit them personally (like all the other programming taxes), that won't help all the programs out there that don't have any such manifest. "Okay, well, let's just say that any program that doesn't have a compatibility manifest is treated as compatible with a maximum of Windows 7." Great. Where do you put the manifest for a batch file? There's a bigger problem, which is applications which are built out of DLLs. What if an application is manifested as "I have been tested only on Windows XP" and it loads a DLL that is manifested as "I have been tested only on Windows Vista"? Now there is no version of Windows that both the application and the DLL have been tested with together. What does `GetVersion` return? Even if your program doesn't have a plug-in model, you're not out of the woods. If you call `ShellExecute` or `GetOpenFileName`, the shell namespace will get loaded, and with it all sorts of shell extensions who may be manifested all sorts of different ways. And then you have the cross-process communication problem. Suppose you drag an object out of a program marked as *I was tested with Windows 7* and drag it over a window that belongs to a program marked as *I was tested only with Windows XP*. How do you communicate this information back to the data source so it knows "Be very careful. You're from the future. Don't tell that process about any data that didn't exist in Windows XP. You might disrupt the time stream." Another

problem, of course, is that just because a program says that it has been tested on Windows XP doesn't mean that it's actually going to run on Windows XP. There are many categories of bugs that appear only on configurations that were unusual on Windows XP but more common on later versions of Windows. (For example, "My user profile isn't in a directory called `Documents and Settings`.") Even if a program says that it was tested on Windows XP, that doesn't mean that running the old Windows XP operating system code is enough to keep it happy. And of course there's the question of whether you want this in the first place. If we had introduced this model back in Windows 2000, by the time you have reached Windows 7, you could have a screen with applications that have three different visual styles: A program manifested as being tested only on Windows 2000 would have the Windows 2000 look, another program would have the Windows XP look, and a third program would have the another program would have the Windows 7 look. Even better: When Windows 7 came out, *no applications would have the Windows 7 look* since the manifest couldn't declare that it was compatible with something that didn't exist. Looking at it another way, the manifest is basically a way for an application to say "Please run me in compatibility mode." But as we saw before, application compatibility layers are there for the customer, not for the program.

Bonus reading: Manifesting for Compatibility on Windows 7.

Raymond Chen

Follow