What happens when you get dllimport wrong?

devblogs.microsoft.com/oldnewthing/20060726-00

July 26, 2006

Raymond Chen

Now that we've learned what the `dllimport` declaration specifier does, what if you get it wrong? If you forget to declare a function as `dllimport`, then you're basically making the compiler act like a naive compiler that doesn't understand `dllimport`. When the linker goes to resolve the external reference for the function, it will use the stub from the import library, and everything will work as before. You do miss out on the optimization that `dllimport` enables, but the code will still run. You're just running in naive mode. (There are still some header files in the Platform SDK that neglect to use the `dllimport` declaration specifier. As a result, anybody who uses those header files to import functions from the corresponding DLL will be operating in "naive mode". Hopefully the people responsible for those header files will recognize themselves in this parenthetical and fix the problem for a future release of the Platform SDK.) Now, what about the reverse problem? What if you declare a function as `dllimport` when it really isn't? The linker detects this since it sees an attempt to import a `__imp__FunctionName` symbol and can't find one, though it can find the normal `FunctionName` symbol. When this happens, the linker raises warning LNK4217. It recovers from this error by simply manufacturing a fake `__imp__FunctionName` variable and initializing it with the address of the `FunctionName` function. In effect, you've imported the function from yourself. Your code now goes through all the gyrations associated with calling an imported function unnecessarily; it could have just called `FunctionName` directly. (There are cases where the linker can be a little smarter. For example, if it sees a `call [__imp__FunctionName]`, it can change it to `call FunctionName + nop`. The `nop` is necessary because the `call [__imp__FunctionName]` instruction is six bytes long, whereas `call FunctionName` is only five. The extra `nop` gets everything back in sync.) Thus, in both cases where you mess up the `dllimport` declaration specifier, the linker manages to recover from your mistake, and your program does run fine, though the patching up did cost you in code size and efficiency. (All this discussion is for x86, by the way. Other architectures have different quirks.)

Next time, more on import libraries, and exposing some "little white lies" I've been telling.

Raymond Chen

Follow