Names in the import library are decorated for a reason

devblogs.microsoft.com/oldnewthing/20060727-00

July 27, 2006

Raymond Chen

When I wrote that the symbolic name for the imported function table entry for a function is called `__imp__FunctionName`, the statement was "true enough" for the discussion at hand, but the reality is messier, and the reason for the messy reality is function name decoration. When a naive compiler generates a reference to a function, the reference is decorated in a manner consistent with its architecture, language, and calling convention. (Some time ago, I discussed some of the decorations you'll see on x86 systems.) For example, a naive call to the `GetVersion` function results in the compiler generating code equivalent to `call _GetVersion@0` (on an x86 system; other architectures decorate differently). The import library therefore must have an entry for the symbol `_GetVersion@0` in order for the external reference to be resolved. To correspond to the stub function whose real name is `_GetVersion@0` is the import table entry whose name is `__imp__GetVersion@0`. In general, the import table entry name is `__imp_` prefixed to the decorated function name. The fact that names in import libraries are decorated means that it is doubly crucial that you use the official import library for the DLL you wish to use rather than trying to manufacture one with an import library generation tool. As we noted earlier, the tool won't know whether the ordinal assigned to a named function was by design or merely coincidental. But what's more, the tool won't know what decorations to apply to the function (if the name was exported under an undecorated name). Consequently, your attempts to call the function will fail to link since the decorations will most likely not match up. In that parenthetical, I mentioned exporting under undecorated names. Doesn't that mean that you can also export with a decorated name? Yes you can, but as I described earlier, you probably shouldn't. For as I noted there, if you export a decorated name, then that name cannot be located via `GetProcAddress` unless you also pass the decorated name to `GetProcAddress`. But the decoration schema changes from language to language, from architecture to architecture, and even from compiler vendor to compiler vendor, so even if you manage to pass a decorated name to the `GetProcAddress` function, you'll have to wrap it inside a huge number of `#ifdef`s so you pass the correct name for the x86 or ia64 or x64, accordingly, as well as changing the name depending on whether you're using the Microsoft C compiler, the Borland C compiler, the Watcom C compiler, or maybe you're using one of the C++ compilers. And

woe unto you if you hope to call the function from Visual Basic or C# or some other language that provides interop facilities. Just export those names undecorated. Your future customers will thank you.

(Exercise: Why is it okay for the C runtime DLLs to use decorated exports?)

Raymond Chen

Follow