Why can't I GetProcAddress for CreateWindow?

devblogs.microsoft.com/oldnewthing/20070102-04

January 2, 2007

Raymond Chen

Occasionally, I'll see people having trouble trying to `GetProcAddress` for functions like `CreateWindow` or `ExitWindows`. Usually, it's coming from people who are trying to write p/invoke signatures, for p/invoke does a `GetProcAddress` under the covers. Why can't you `GetProcAddress` for these functions?

Because they're not really functions. They're function-like macros:

```
#define CreateWindowA(lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)\
CreateWindowExA(0L, lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)
#define CreateWindowW(lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)\
CreateWindowExW(0L, lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)
#ifdef UNICODE
#define CreateWindow  CreateWindowW
#else
#define CreateWindow  CreateWindowA
#endif // !UNICODE
#define ExitWindows(dwReserved, Code) ExitWindowsEx(EWX_LOGOFF, 0xFFFFFFFF)
```

In fact, as you can see above `CreateWindow` is doubly a macro. First, it's a redirecting macro that expands to either `CreateWindowA` or `CreateWindowW`, depending on whether or not you are compiling `UNICODE`. Those are in turn function-like macros that call the real function `CreateWindowExA` or `CreateWindowExW`. All this is handled by the compiler if you include the `winuser.h` header file, but if for some reason you want to `GetProcAddress` for a function-like macro like `CreateWindow`, you'll have to manually expand the macro to see what the real function is and pass that function name to `GetProcAddress`.

Similar remarks apply to inline functions. These functions can't be obtained via `GetProcAddress` because they aren't exported at all; they are provided to you as source code in the header file.

Note that whether something is a true function or a function-like macro (or an inline function) can depend on your target platform. For example, `GetWindowLongPtrA` is a true exported function on 64-bit Windows, but on 32-bit Windows, it’s just a macro that resolves to `GetWindowLongA`. As another example, the `Interlocked` family of functions are exported functions on the x86 version of Windows but are inlined functions on all other Windows architectures.

How can you figure all this out? Read the header files. That’ll show you whether the function you want is a redirecting macro, a function-like macro, an inline function, an intrinsic function, or a proper exported function. If you can’t figure it out from the header files, you can always just write a program that calls the function you’re interested in and then look at the disassembly to see what actually got generated.

Raymond Chen

Follow