Wait, but why can I GetProcAddress for IsDialogMessage?

devblogs.microsoft.com/oldnewthing/20070103-15

January 3, 2007

Raymond Chen

Okay, so I explained that a lot of so-called functions are really redirecting macros, function-like macros, intrinsic functions, and inline functions, and consequently, `GetProcAddress` won't actually get anything since the function doesn't exist in the form of an exported function. But why, then, can you `GetProcAddress` for `IsDialogMessage`?

Let's take a closer look at the exports from `user32.dll`. Here's the relevant excerpt.

```
        417   1A0 0002C661 IsDialogMessage
        418   1A1 0002C661 IsDialogMessageA
        419   1A2 0001DFBC IsDialogMessageW
```

Notice that this function is exported **three** ways. The last two are the ones you expect, `IsDialogMessageA` for ANSI callers and `IsDialogMessageW` for UNICODE callers. That first one is the one you didn't expect: `IsDialogMessage` with no A or W suffix. But notice that its entry point address is identical to that of `IsDialogMessageA`. The `IsDialogMessage` entry point is just an alias for `IsDialogMessageA`.

This phantom third function is hidden from C and C++ programs because any attempt to call `IsDialogMessage` gets converted to `IsDialogMessageA` or `IsDialogMessageW` due to the redirection macro:

```
#ifdef UNICODE
#define IsDialogMessage  IsDialogMessageW
#else
#define IsDialogMessage  IsDialogMessageA
#endif // !UNICODE
```

(Of course, you can play fancy games to remove the redirection macros; I'm just talking about the non-fancy case.) If nobody can call the function, then why does it exist?

Because of mistakes made long ago.

If you hunt through `user32.dll` you'll find a few other functions that follow a similar pattern of having three versions, an A version, a W version, and a phantom undecorated version (which is an alias for the A version). At one point long ago, the function existed only in an undecorated version. This turned out to have been a mistake, since there was a character set dependency in the parameters (perhaps obvious, perhaps subtle). The mistake was corrected by splitting the function into the A and W versions you see today, but in order to maintain compatibility with older programs that were written before the mistake was recognized, the original undecorated function was left in the export table.

When you don't have a time machine, you have to live with your mistakes.

In a sense, these functions are vestigial organs of Win32.

Postscript: Unfortunately, like your appendix, which can get infected, these vestigial organs can create a different sort of infection: If you are using p/invoke to call these functions and mistakenly override the default name declaration with `ExactSpelling=true`, like so:

```
[DllImport("user32.dll", ExactSpelling=true)]
public static extern
bool IsDialogMessage(IntPtr hWndDlg,
                     [In] ref MSG msg);
```

then you will in fact get the normally-inaccessible undecorated name, since you specified that you wanted the exact spelling. This highlights once again that you need to be alert when doing interop programming: You get what you ask for, which might not be what you actually wanted.

Raymond Chen

Follow