It’s okay to be contrary, but you need to be consistently contrary: Going against the ambient character set

devblogs.microsoft.com/oldnewthing/20211210-00

December 10, 2021

Raymond Chen

In Windows, you declare your character set preference implicitly by defining or not defining the symbol `UNICODE` before including the `windows.h` header file. (Related: TEXT vs. _TEXT vs. _T, and UNICODE vs. _UNICODE.) This determines whether undecorated function names redirect to the ANSI version or the Unicode version, but it doesn’t make the opposite-version inaccessible. You just have to call them by their explicit names. And it’s important that you be consistent about it. If you miss a spot, the characters get all messed up.

```
// UNICODE not defined
#include <windows.h>

void UpdateTitle(HWND hwnd, PCWSTR title)
{
    SetWindowTextW(hwnd, title);
}
```

In the above example, we did not define the symbol `UNICODE`, so the ambient character set is ANSI. Since we want to call the Unicode version of `SetWindowText`, we must use its explicit Unicode name `SetWindowTextW`.

Most of the time, these errors are detected at compile time due to type mismatches. For example, if we forgot to put the trailing `W` on the function name, we would get the error

```
error C2664: 'BOOL SetWindowTextA(HWND,const char *)': cannot convert argument 2 from
'const wchar_t *' to 'const char *'
note: Types pointed to are unrelated; conversion requires reinterpret_cast, C-style
cast or function-style cast
```

And that’s your clue that you forgot to W-ize the `SetWindowText` call. You should have called the W version explicitly: `SetWindowTextW`.

However, there’s a category of functions that elude this compile-time detection: The functions that have separate ANSI and Unicode versions but take only character-set-independent parameters. Common examples are `DispatchMessage`, `TranslateMessage`, `TranslateAccelerator`, `CreateAcceleratorTable`, and most notably, `DefWindowProc`.

For some reason, when I get called in to investigate this sort of problem, it’s usually the `Def-WindowProc` that is the source of the problem.

But I don’t think it’s because people get the others right and miss the `DefWindowProc`. I think it’s because the mistakes in the other functions are much less noticeable. The mistakes are still there, and maybe you’ll get a bug report from a user in Japan when they run into it, but that’s not something that is going to be noticed in English-based testing as much as a string that is truncated down to its first letter.

Raymond Chen

Follow