Before claiming that a function doesn't work, you should check what you're passing to it and what it returns

devblogs.microsoft.com/oldnewthing/20140801-00

August 1, 2014

Raymond Chen

Before claiming that a function doesn't work, you should check what you're passing to it and what it returns, because it may be that the function is behaving just fine and the problem is elsewhere.

> The `GetCurrentDirectoryW` function does not appear to support directories with Unicode characters in their names.
>
> ```
> wchar_t currentDirectory[MAX_PATH];
> GetCurrentDirectoryW(MAX_PATH, currentDirectory);
> wcout << currentDirectory << endl;
> ```
>
> The correct directory name is obtained if it contains only ASCII characters in its name, but it truncates the string at the first non-ASCII character.

If you step through the code in the debugger, you'll see that the `GetCurrentDirectoryW` function is working just fine. The buffer is filled with the current directory, including the non-ASCII characters. The problem is that the `wcout` stream stops printing the directory name at the first non-ASCII characters. And that's because the default locale for `wcout` is the `"C"` locale, and the `"C"` locale is "the minimal environment for C translation." The `"C"` locale is useless for actual work involving, you know, locales. You will have to do some language-specific munging to get the characters to reach the screen in the format you want, the details of which are not the point of today's topic.

In other words, the bug was not in the `GetCurrentDirectoryW` function. It was in what you did with the result of the `GetCurrentDirectoryW` function.

Here's another example of thinking the problem is in a function when it isn't:

> The `SetWindowTextW` function does not appear to support Unicode, despite its name.
>
> ```
> wstring line;
> wifstream file("test"); // this file is in Unicode
> getline(file, line);
> SetWindowTextW(hwnd, line.c_str());
> ```

If you look at the `line` variable before you even get around to calling `SetWindowTextW`, you'll see that it does not contain the text from your Unicode file. The problem is that the default `wifstream` reads the text as an 8-bit file, and then internally converts it (according to the lame `"C"` locale) to Unicode. If the original file is already Unicode, you're doing a double conversion and things don't go well. You then pass this incorrectly-converted string to `SetWindowTextW`, which naturally displays something different from what you intended.

Again, the point is not to delve into the intricacies of `wifstream`. The point is that the problem occurred even before you called `SetWindowTextW`. The observed behavior, then, is simple a case of Garbage In, Garbage Out.

Here's another example from a few years ago.

Raymond Chen

Follow