Converting to Unicode usually involves, you know, some sort of conversion

devblogs.microsoft.com/oldnewthing/20120328-00

March 28, 2012

Raymond Chen

A colleague was investigating a problem with a third party application and found an unusual window class name: L"整瑳整瑳". He remarked, "This looks quite odd and could be some problem with the application."

The string is nonsense in Chinese, but I immediately recognized what was up.

Here's a hint: Rewrite the string as

> L"\x6574" L"\x7473" L"\x6574" L"\x7473"

Still don't see it? How about looking at the byte sequence, remembering that Windows uses UTF-16LE.

> 0x74 0x65 0x73 0x74 0x74 0x65 0x73 0x74

Okay, maybe you don't have your ASCII table memorized.

> | 0x74 | 0x65 | 0x73 | 0x74 | 0x74 | 0x65 | 0x73 | 0x74 |
> |---|---|---|---|---|---|---|---|
> | t | e | s | t | t | e | s | t |

That's right, the application took the ASCII string "testtest" and just treated it as a Unicode string without actually converting it to Unicode. When the compiler complained "Cannot convert char * to wchar_t *" they just stuck a cast to make the compiler shut up.

```
// Code in italics is wrong
WNDCLASSW wc;
wc.lpszClassName = (LPWSTR)"testtest";
```

They were lucky that the compiler happened to put *two* null bytes at the end of the "testtest" string.

Bonus psychic powers: Actually, I have a theory as to how this happened that doesn't involve maliciousness. (This is generally a good mindset to maintain, since most of the time, when people cause a problem, it's not willful; it's accidental.) Consider a library with the following interface header file:

```
// mylib.h
#ifdef __cplusplus
extern "C" {
#endif
BOOL RegisterWindowClass(LPCTSTR pszClassName);
#ifdef __cplusplus
}; // extern "C"
#endif
```

Somebody uses this header file like this:

```
#include <mylib.h>
BOOL Initialize()
{
    return RegisterWindowClass(TEXT("testtest"));
}
```

So far so good.

Meanwhile, the library implementation goes like this:

```
#define UNICODE
#define _UNICODE
#include <mylib.h>
LRESULT CALLBACK StandardWndProc(HWND, UINT, WPARAM, LPARAM);
BOOL RegisterWindowClass(LPCTSTR pszClassName)
{
    WNDCLASS wc = { 0, StandardWndProc, 0, 0, g_hInstance,
                    LoadIcon(IDI_APPLICATION),
                    LoadCursor(IDC_ARROW),
                    (HBRUSH)(COLOR_WINDOW + 1),
                    NULL, pszClassName);
    return RegisterClass(&wc);
}
```

The two files both compile successfully, and they even link together. Unfortunately, one of them was compiled with Unicode disabled, and the other was compiled with Unicode enabled. Since the header file uses `LPCTSTR`, the actual declaration of `RegisterWindowClass` *changes* depending on whether the code that includes the header file is compiled as Unicode or ANSI.

Result: If one file is compiled as ANSI and the other is compiled as Unicode, then one will pass an ANSI string, which the other will receive and treat as Unicode.

This is why functions in Windows which are dependent on whether the caller is compiled as ANSI or Unicode are really two functions, one with the A suffix (for ANSI) and another with the W suffix (for Wnicode?), and the generic name is really a macro that forwards to one or the other. It prevents `TCHAR` s from sneaking past the compiler and ending up being interpreted differently by the two sides.

Raymond Chen

Follow