



C++

Information

Tutorials

Reference

Articles

Forum

Forum

Beginners

Windows Programming

UNIX/Linux Programming

General C++ Programming

Lounge

Jobs

? Help with UNICODE?

dp1 (13)

Mar 29, 2013 at 4:58am

Hello everybody,

I'm developing a simple notepad-like application, using only WinAPI.

I've added some macro checking to compile for unicode or ASCII without changing the source code. When I load a file in ASCII mode, I don't have any error, but when I do the same in unicode mode, I get chinese-like symbols (the file I load is plain ASCII)

If you have any suggestion, please write them to me.

Best regards,

dp1

Disch (13769)

Mar 29, 2013 at 6:49pm

When I load a file in ASCII mode, I don't have any error, but when I do the same in unicode mode, I get chinese-like symbols

I think you're misunderstanding what "Unicode mode" really means. In Visual Studio, all it really means is that the wide version of WinAPI functions are used by default. That's it. For you to actually use them properly, you must give them proper UTF-16 strings.

If you're just taking an ASCII string and expecting it to be UTF-16... it isn't. And it will get interpreted strangely.

For example:

```
1 // say you have this in an ASCII text file
2
3 hello!
4
5 // this is 6 bytes long, one byte for each character. Each byte would be the
6 // ASCII code for each character. It would look like this in a hex editor:
7
8 68 65 6C 6C 6F 21
9
10 // where 0x68 is the ASCII code for 'h', 0x65 is the code for 'e', etc
11
12 // the problem you're having is that you are interpreting the file as UTF-16. UTF-16
13 // has 2 bytes per character (it actually can be up to 4 bytes, but don't worry about that
14 // for now -- it's usually only 2)
```

```
15 //  
16 // That same binary data:  
17  
18 68 65 6C 6C 6F 21  
19  
20 // interpreted as UTF-16 (little endian):  
21  
22 6568 6C6C 216F  
23  
24 // is only 3 characters long. And those 3 characters are:  
25 // U+6568 ( 散 )  
26 // U+6C6C ( 菜 )  
27 // U+216F ( M )  
28 //  
29 // As you can see, this text is nothing at all like "hello"
```

If the text is in ASCII and you want to give it to a function that expects UTF-16... you need to convert the text data. Simply changing the compiler setting does not do this automatically... you have to write code to do it.

WinAPI has conversion functions but I can't for the life of me remember what they are because I hardly ever use them. I'll see if I can look them up...

EDIT: also that Unicode compiler setting is stupid. If you want to use the UTF-16 version of functions, just call them directly. Each function which accepts strings in WinAPI has 3 forms:

normal form: MessageBox (takes TCHAR strings -- a TCHAR is either a char or a wchar_t depending on that stupid Unicode setting). It's worth noting that TCHARs are incredibly stupid and you probably shouldn't be using them, which means you probably should never be calling the normal form of any WinAPI function that takes strings.

ansi form: MessageBoxA (takes char strings)

wide form: MessageBoxW (takes wchar_t strings, accepts UTF-16 text)

So if you want Unicode support, you'll probably need to always use the 'W' functions and just deal with wide characters all the time. Either that or set the code page to UTF-8 and use the A forms, giving them UTF-8 strings... but I don't know if that actually works or not.

EDIT 2:

Found the conversion functions:

<http://msdn.microsoft.com/en-us/library/windows/desktop/dd319072%28v=vs.85%29.aspx>


You would want to use CP_UTF8 as the code page.

Untested example:

```
1 char ascii[100] = "This is a test string.";
2
3 wchar_t utf16[100];
4
5 int widelength = MultiByteToWideChar( CP_UTF8, 0, ascii, -1, utf16, 100 );
6
7 // output the ASCII text
8 MessageBoxA( NULL, ascii, NULL, MB_OK );
9
10 // output the UTF-16 text (should be identical)
11 MessageBoxW( NULL, utf16, NULL, MB_OK );
```

Last edited on Mar 29, 2013 at 7:04pm

closed account ([ozUkoGIT](#))

 Mar 30, 2013 at 12:44pm

Well, the EDIT 2 is most effective also when developing such application keep in mind that the client using it would probably have different keyboard layout to yours so always consider Unicode as well.

blackcoder41 (1426)

 Mar 31, 2013 at 1:14pm

Disch wrote:

normal form: MessageBox (takes TCHAR strings -- a TCHAR is either a char or a wchar_t depending on that **stupid** Unicode setting). It's worth noting that TCHARs are incredibly **stupid** and you probably shouldn't be using them, which means you probably should never be calling the normal form of any WinAPI function that takes strings.

What's so stupid about it??

Disch (13769)

 Mar 31, 2013 at 1:29pm

Using it as it's intended to be used results in incredibly obfuscated code.

The whole point of TCHAR is that they can be either char or wchar_t depending on that setting, and you should write your code to allow for either case.

The benefit of that is you can compile your code without Unicode support if it isn't needed, but just flip a switch and recompile if you do need Unicode support. However I don't see why it'd be beneficial to ever build without Unicode support (does it make for a smaller binary? I know it doesn't improve performance).

If you're going to take the effort to support Unicode, you might as well just support it and not be wishy-washy

about it.

To write proper TCHAR code you have to be constantly aware of the TCHAR's variable size and meaning. Even something that's relatively straightforward as expanding a UTF-8 string to UTF-16 becomes extra work as a result:

```
1 const char* userstring = "some 8-bit string data that game from a text file or something";
2
3 TCHAR buffer[1000];
4
5 // strcpy() will only work if TCHAR==char
6 // MultiByteToWideChar() will only work if TCHAR==wchar_t
7 //
8 // so to probably assign 'buffer' you have to #ifdef it
9 #if defined(UNICODE) || defined(_UNICODE)
10 MultiByteToWideChar( CP_UTF8, 0, userstring, -1, buffer, 1000 );
11 #else
12 strcpy( buffer, userstring );
13 #endif
14
15 SetWindowText( somewnd, buffer );
```

Of course the smarter thing to do here would be to have some kind of "toTchar" function which does this so you don't have to embed the #ifdefs in your actual code. But the #ifdefs still have to be there.

Compare that to just supporting Unicode unconditionally:

```
1 const char* userstring = "some 8-bit string data that game from a text file or something";
2
3 wchar_t buffer[1000];
4 MultiByteToWideChar( CP_UTF8, 0, userstring, -1, buffer, 1000 );
5
6 SetWindowTextW( somewnd, buffer );
```

So much simpler.

Topic archived. No new replies allowed.