How does the MultiByteToWideChar function treat invalid characters?

devblogs.microsoft.com/oldnewthing/20120504-00

May 4, 2012

Raymond Chen

The `MB_ERR_INVALID_CHARS` flag controls how the `MultiByteToWideChar` function treats invalid characters. Some people claim that the following sentences in the documentation are contradictory:

- "Starting with Windows Vista, the function does not drop illegal code points if the application does not set the flag."
- "Windows XP: If this flag is not set, the function silently drops illegal code points."
- "The function fails if `MB_ERR_INVALID_CHARS` is set and an invalid character is encountered in the source string."

Actually, the three sentences are talking about different cases. The first two talk about what happens if you omit the flag; the third talks about what happens if you include the flag.

Since people seem to like tables, here's a description of the `MB_ERR_INVALID_CHARS` flag in tabular form:

`MB_ERR_INVALID_CHARS` set?	Operating system	Treatment of invalid character
Yes	Any	Function fails
No	XP and earlier	Character is dropped
	Vista and later	Character is not dropped

Here's a sample program that illustrates the possibilities:

```
#include <windows.h>
#include <ole2.h>
#include <windowsx.h>
#include <commctrl.h>
#include <strsafe.h>
#include <uxtheme.h>
void MB2WCTest(DWORD flags)
{
 WCHAR szOut[256];
 int cch = MultiByteToWideChar(CP_UTF8, flags,
                               "\xC0\x41\x42", 3, szOut, 256);
 printf("Called with flags %d\n", flags);
 printf("Return value is %d\n", cch);
 for (int i = 0; i < cch; i++) {
  printf("value[%d] = %d\n", i, szOut[i]);
 }
 printf("-----\n");
}
int __cdecl main(int argc, char **argv)
{
 MB2WCTest(0);
 MB2WCTest(MB_ERR_INVALID_CHARS);
 return 0;
}
```

If you run this on Windows XP, you get

```
Called with flags 0
Return value is 2
Value[0] = 65
Value[1] = 66
-----
Called with flags 8
Return value is 0
-----
```

This demonstrates that passing the `MB_ERR_INVALID_CHARS` flag causes the function to fail, and omitting it causes the invalid character \xC0 to be dropped.

If you run this on Windows Vista, you get

```
Called with flags 0
Return value is 3
Value[0] = 65533
Value[1] = 65
Value[2] = 66
-----
Called with flags 8
Return value is 0
-----
```

This demonstrates again that passing the `MB_ERR_INVALID_CHARS` flag causes the function to fail, but this time, if you omit the flag, the invalid character \xC0 is converted to U+FFFD, which is REPLACEMENT CHARACTER. (Note that it does not appear to be documented precisely *what* happens to invalid characters, aside from the fact that they are not dropped. Perhaps code pages other than `CP_UTF8` convert them to some other default character.)

Raymond Chen

Follow