The Resource Compiler defaults to CP_ACP, even in the face of subtle hints that the file is UTF-8

devblogs.microsoft.com/oldnewthing/20190607-00

June 7, 2019

Raymond Chen

The Resource Compiler assumes that its input is in the ANSI code page of the system that is doing the compiling. This may not be the same as the ANSI code page of the system that the `.rc` was authored on, nor may it be the same as the ANSI code page of the system that will consume the resulting resources.

It also completely ignores any clues in the file itself.

The saga begins in 1981.

At this time, code pages roamed the earth. There was no way to know what encoding to use for a file; you just assumed it was the ambient code page for the system that opened the file and hoped for the best.

This is the world the Resource Compiler was born into.

```
STRINGTABLE BEGIN
IDS_MYSTRING "Hello, world."
END
```

Some years later, Unicode was invented, and the Resource Compiler let you indicate that you wanted a Unicode string by using the `L` prefix.

```
STRINGTABLE BEGIN
IDS_MYSTRING L"Hello, world."
END
```

In the above case, the `L` didn't have any effect since the string itself limits itself to 7-bit ASCII. But let's say that you used a fancy apostrophe in the Windows-1252 code page.

```
STRINGTABLE BEGIN
IDS_MYSTRING L"What’s up?"
END
```

There are two things to note. First is that you need to put the `L` prefix on the string to get it to be interpreted as Unicode. And second, the apostrophe is encoded as the single byte `92h` because the file is in the Windows-1252 code page.

Now, it's possible that the system doing the compiling isn't using Windows-1252 as its default code page. For example, you might author the files in Windows-1252 because your main office is in Redmond, Washington, but you then send the file to your Japanese office, and their code page is 932. The byte sequence `92h 73h` means "apostrophe, small Latin letter s" in the Windows-1252 code page, but in code page 932, that byte sequence represents the character 痴. When the Japanese office compiles your resource script, they get *What痴up?*. This is already embarrassing enough, but it's compounded by the fact that the character 痴 means gonorrhea.

To avoid this problem, the Resource Compiler lets you declare the code page in which the subsequent lines should be interpreted. This removes any dependency on the execution environment of the compiler.

```
#pragma code_page(1252)

STRINGTABLE BEGIN
IDS_MYSTRING L"What's up?"
END
```

Some years later, UTF-8 was introduced. This created an interesting problem, because you might load a file as Windows-1252, but then when you save it, your text editor "helpfully" converts it to UTF-8. This change often goes undetected because file comparison tools will frequently "helpfully" normalize the two files into a common encoding before comparing them, thereby hiding the encoding change.

And then you get a bug that says "Garbage characters in message. Message is supposed to say `What's up?`, but instead it says `Whatâ€™s up?`."

What happened is that the byte `92h` in Windows-1252 was re-encoded into UTF-8 as the bytes `E2h 80h 99h`. Those bytes then were interpreted by the compiler as Windows-1252, resulting in â€™. The presence of a UTF-8 BOM at the start of the file was a subtle hint that the file was really UTF-8 encoded, but computers aren't very good at subtlety. They just follow the rules they were given, and that rule is "Interpret the bytes in the system ANSI code page unless given *explicit* instructions to the contrary."

The fix is to give explicit instructions to the contrary. Put this at the top of the file:

```
#pragma code_page(65001) // UTF-8
```

Now save the file in UTF-8.

Now you're all set. Text editors nowadays will happily "help" you out by silently converting to UTF-8, but I don't know of any that silently convert to Windows-1252.

Raymond Chen

Follow