# How do I convert an ANSI string directly to UTF-8?

devblogs.microsoft.com/oldnewthing/20100603-00

June 3, 2010

Raymond Chen

A customer asked the following question:

> Is there a way to convert an ANSI string directly to UTF-8 string? I have an ANSI string which was converted from Unicode based of the current code page. I need to convert this string to UTF-8.
>
> Currently I am converting the string from ANSI to Unicode ( `MultiByteToWideChar(CP_ACP)` ) and then converting the Unicode to UTF-8 ( `WideCharToMultibyte(CP_UTF8)` ). Is there a way to do the conversion without the redundant conversion back to Unicode?

There is no multibyte-to-multibyte conversion function built into Windows (as of this writing). To convert from one 8-bit encoding to another, you have to use Unicode as an intermediate step. Fortunately, one of my colleagues chose not to answer the question but instead responded to the question with another question:

> Is the data loss created by the initial conversion to ANSI really acceptable? Convert from the original Unicode string to UTF-8, and you avoid the potential mess introduced by the Unicode-to-ANSI conversion step.

The customer was puzzled by this data loss remark:

> I'm using the same code page when converting from Unicode to ANSI as I am from converting from ANSI to Unicode. Will there still be a data loss?

None of the code pages which Windows supports as an ANSI code page can express the full repertoire of Unicode characters. It's simple mathematics: Since one of the requirements for being an ANSI code page is that no single character can be more than 2 bytes, there simply isn't enough expressive power to encode all of Unicode. Now, if you're lucky, all of the characters you're encoding will exist in the ANSI code page, and they will survive the round trip, but that's just if you're lucky. It's like converting an image from 32-bit color to 8-bit color via the halftone palette. The palette is the "code page" for the conversion. Remembering to use the same palette when converting back is an essential step, but the result of the round trip will be a degraded image because you can’t encode all 32-bit colors in a single 256-color palette. If you’re lucky, all the colors in the original image will exist in your palette and the conversion will not result in loss of information, but you shouldn’t count on being lucky. The customer went on to explain:

> Unfortunately, my code does not have access to the original Unicode string. It is a bridge between two interfaces, one that accepts an ANSI string, and another that accepts a UTF-8 string. I would have to create a new Unicode interface, and modify all existing callers to switch to the new one.

If all the callers are generating Unicode strings and converting them to ANSI just to call the original ANSI-based interface, then creating a new Unicode-based interface might actually be a breath of fresh air. Keep the poorly-designed ANSI interface around for backward compatibility, so that callers could switch to the Unicode-based interface at their leisure.

**Bonus chatter**: Even the round trip from ANSI to Unicode and back to ANSI can be lossy, depending on the flags you pass regarding use of precomposed characters, for example.