What can I do if IMlangConvertCharset is unable to convert from code page 28591 directly to UTF-8?

devblogs.microsoft.com/oldnewthing/20240726-00

July 26, 2024

Raymond Chen

A customer wanted to do a character set conversion from code page 28591 directly to UTF-8. They found that when they ask `IMultiLanguage::CreateConvertCharset` to create such a converter, it returns `S_FALSE`, meaning that no such conversion is available.

```
auto mlang = wil::CoCreateInstance<IMultiLanguage>(
        CLSID_CMultiLanguage);

// This next call returns S_FALSE, indicating no conversion.
wil::com_ptr<IMLangConvertCharset> convert;
mlang->CreateConvertCharset(28591, CP_UTF8, 0, &convert);
```

Oh no, what shall we ever do?

Okay, so `CMultiLanguage` can't convert from 28591 directly to UTF-8. But you can just convert through UTF-16.

```
HRESULT ConvertStringFrom28591ToUtf8(
    char const* input,
    int inputLength,
    char * output,
    int outputCapacity,
    int* actualOutput)
{
    *actualOutput = 0;

    // Ensure we are not working with negative numbers.
    RETURN_HR_IF(E_INVALIDARG, inputLength < 0 ||
                               outputCapacity < 0);

    // Empty string converts to empty string.
    if (inputLength == 0)
    {
        return S_OK;
    }

    // Avoid edge cases if outputCapacity = 0.
    // This also short-circuits cases where we know that the
    // output buffer isn't big enough to hold the converted input.
    RETURN_HR_IF(HRESULT_FROM_WIN32(ERROR_INSUFFICIENT_BUFFER),
                inputLength > outputCapacity);

    // Code page 28591 resides completely in the BMP.
    auto bufferCapacity = std::min(inputLength, outputLength);
    auto buffer = wil::make_unique_hlocal_nothrow<wchar_t[]>(
        bufferCapacity);
    RETURN_IF_NULL_ALLOC(buffer);

    // Convert from 28591 to UTF-16LE.
    auto result = MultibyteToWideChar(28591, MB_ERR_INVALID_CHARS,
        input, inputLength, buffer.get(), maximumOutput);
    RETURN_IF_WIN32_BOOL_FALSE(result != 0);

    // Convert from UTF-16LE to UTF-8.
    *actualOutput = WideCharToMultiByte(CP_UTF8, 0,
        buffer.get(), bufferCapacity,
        output, outputCapacity, nullptr, nullptr);
    RETURN_IF_WIN32_BOOL_FALSE(*actualOutput != 0);

    return S_OK;
},
```

After dealing with some edge cases, we allocate a temporary UTF-16LE buffer. That buffer needs to be big enough to hold the converted input, but doesn’t need to be so big that the caller-provided output couldn’t possibly hold the result.

Since all the characters of code page 28591 have code points less than U+10000, they will convert to a single UTF-16LE code unit. Therefore, we will need at most `inputLength` UTF-16LE code units to hold the intermediate UTF-16LE output.

And since all the code points of the intermediate buffer will be less than U+10000, there will never be a need for more UTF-16LE code units than corresponding UTF-8 code units. Therefore, any intermediate buffer bigger than `outputCapacity` wouldn't fit in the caller-provided buffer anyway, so we can just return the "insufficient buffer" error right away without having to do any work.

The rest is anticlimactic: We convert the input buffer to our temporary buffer, and then we convert the temporary buffer to the output buffer.

In the general case, a single input byte could result in two UTF-16LE code units, if it represents a character outside the BMP. (We assume that no code page has a single input byte that converts to multiple Unicode characters.) And the worst-case expansion from UTF-8 bytes to UTF-16LE code units is just 1:1. So in the general case, the required temporary buffer capacity is `std::min(2 * inputLength, outputCapacity)`.

The whole `IMultiLanguage` interface was a red herring. You never needed it. The conversion was in front of you the whole time.

Bonus chatter: The entire MultiLanguage API family has been deprecated since at least 2008, possibly longer, so it's a good thing we migrated away from it.

Bonus bonus chatter: The International Components for Unicode (ICU) have been included with Windows since Windows 10 version 1703, so if you don't need to support anything older than that, you can just use the copy of ICU built into Windows. The `ucnv_convertEx` function lets you convert from one encoding to another. Mind you, it "pivots" through UTF-16, so it's internally doing the same thing we are, but at least it done for you. You can consult the ICU documentation for more information about converters.