How can I convert between IANA time zones and Windows registry-based time zones?

devblogs.microsoft.com/oldnewthing/20210527-00

May 27, 2021

Raymond Chen

A customer wanted to be able to convert from IANA time zones and Windows registry-based time zones. Is there a function for doing that?

Yes, but it's not somewhere you might think to look.

The Windows globalization team emphasizes that what you should *not* do is take the information from CLDR, use it to produce a translation table, and hard-code that table into your application. Time zones are notoriously fickle, and there's a team of people inside Microsoft whose full-time job is to keep track of time zone changes worldwide.

What you need to do is delegate the work of updating time zone information to somebody else.

One such *somebody else* is ICU, the International Components for Unicode.

A copy of ICU has been included with Windows since Windows 10 Version 1703 (build 15063). All you have to do is include `icu.h`, and you're off to the races. An advantage of using the version that comes with Windows is that it is actively maintained and updated by the Windows team. If you need to run on older systems, you can build your own copy from their fork of the ICU repo, but the job of servicing the project is now on you.

Here's a proof-of-concept program that shows how to convert between IANA time zones and Windows time zones.

```
#include <windows.h>
#define UCHAR_TYPE wchar_t
#include <icu.h>

int wmain(int argc, wchar_t** argv)
{
    wchar_t buffer[128];
    UErrorCode status = U_ZERO_ERROR;
    if (argc == 2) {
        auto result = ucal_getWindowsTimeZoneID(
            argv[1], -1, buffer, ARRAYSIZE(buffer), &status);
        if (U_SUCCESS(status)) {
            printf("result = %d, IANA %ls -> Windows %ls\n",
                   result, argv[1], buffer);
        }
    } else if (argc == 3) {
        char region[64];
        if (WideCharToMultiByte(CP_UTF8, 0, argv[2], -1,
                                region, 64, 0, nullptr)) {
            auto result = ucal_getTimeZoneIDForWindowsID(
                argv[1], -1, region, buffer, ARRAYSIZE(buffer), &status);
            if (U_SUCCESS(status)) {
                printf("result = %d, Windows %ls:%s -> IANA %ls\n",
                       result, argv[1], region, buffer);
            }
        }
    }
    return 0;
}
```

By default, the `icu.h` header uses `char16_t` to represent UTF-16 code units. Windows, however, uses `wchar_t`, so we define `UCHAR_TYPE` to get the type we want.

If you run the program with a single command line parameter, then we treat it as an IANA time zone and ask `ucal_getWindowsTimeZoneID` to convert it to a Windows time zone.

```
C:\> scratch America/Vancouver
result = 21, IANA America/Vancouver -> Windows Pacific Standard Time
```

If you run it with two command line parameters, then we treat the first as the Windows time zone and the second as the region, and convert the pair to an IANA time zone.

```
C:\> scratch "Pacific Standard Time" US
result = 19, Windows Pacific Standard Time:US -> IANA America/Los_Angeles

C:\> scratch "Pacific Standard Time" CA
result = 17, Windows Pacific Standard Time:CA -> IANA America/Vancouver
```

To resolve these calls to the system-provided `ICU.DLL`, link either with `icu.lib` or with the umbrella library `windowsapp.lib`.

Bonus chatter: The Windows globalization team also strongly recommends that programs use IANA time zones and use the Windows registry-based time zones only for legacy interop purposes.

Important bonus chatter: Even though support arrived in Windows 10 Version 1703 (build 15063), using ICU prior to Windows 10 Version 1903 (build 18362) is a little trickier. When it was introduced, the ICU library was added as two system DLLs:

Feature	Header file	Library	DLL
Common	`icucommon.h`	`icuuc.lib`	`icuuc.dll`
Internationalization	`icui18n.h`	`icuin.lib`	`icuin.dll`

In Windows 10 Version 1709, the two header files were combined into a single `icu.h` header file.

Feature	Header file	Library	DLL
Common	`icu.h`	`icuuc.lib`	`icuuc.dll`
Internationalization		`icuin.lib`	`icuin.dll`

The separate header files still remain for source code backward compatibility, but no new features will be added to the separate header files.

And in Windows 10 Version 1903, the separate LIBs and DLLs were also combined:

Feature	Header file	Library	DLL
Common	`icu.h`	`icu.lib`	`icu.dll`
Internationalization			

Again, the old LIBs and DLLs remain for backward compatibility. (The old DLLs are now forwarders to the combined `icu.dll`.)

This change was triggered by a breaking change in the underlying ICU library that moved one of the functions from Common to Internationalization. Since the ICU team had to do some architectural work to accommodate the breaking change, they took the time to do some other tidying.

Combining two DLLs into one DLL allowed for a reduction in the overall footprint of ICU. They also removed the requirement that you call `CoInitializeEx` before calling any ICU functions. (In earlier versions, if you failed to call `CoInitializeEx`, then the default ICU locale is always `en_US` and the default time zone is always `Etc/UTC`.)

TL;DR: If you want to support versions prior to Windows 10 Version 1903, you should load the appropriate separately-named DLL and make sure to call `CoInitializeEx`.

To be sure you're getting the one that doesn't require COM to be initialized, go straight to `icu.dll`. (That's what .NET Core does.)

Raymond Chen

Follow