A question about the FileTimeToLocalFileTime function turned out to be something else

devblogs.microsoft.com/oldnewthing/20150306-00

March 6, 2015

Raymond Chen

A customer reported that their program was running into problems with the `FileTimeToLocalFileTime` function. Specifically, they found that the values reported by the function varied wildly for different time zones. Even though the two time zones were only a few hours apart, the results were hundreds of centuries apart.

The customer did a very good job of reducing the problem, providing a very simple program that illustrated the problem. I cleaned it up a bit.

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char **argv)
{
 FILETIME ftUTC = { 0, 0 };
 FILETIME ftLocal;
 SYSTEMTIME stLocal;
 double vLocal = 0;
 BOOL result = 0;

 printf("ftUTC = {%d,%d}\n",
        ftUTC.dwHighDateTime, ftUTC.dwLowDateTime);

 result = FileTimeToLocalFileTime(&ftUTC, &ftLocal);
 printf("FT2LFT returns %d\n", result);

 printf("ftLocal = {%d,%d}\n",
        ftLocal.dwHighDateTime, ftLocal.dwLowDateTime);

 FileTimeToSystemTime(&ftLocal, &stLocal);

 printf("stLocal = %d.%d.%d %02d:%02d:%02d\n",
        stLocal.wYear, stLocal.wMonth, stLocal.wDay,
        stLocal.wHour, stLocal.wMinute, stLocal.wSecond);

 SystemTimeToVariantTime(&stLocal, &vLocal);
 printf("vLocal = %f\n", vLocal);

 return 0;
}
```

According to the customer, “When we run the program with the current time zone set to UTC-8, we get the correct values, but if we run it with the time zone set to UTC+8, we get the wrong values. We expect that a zero starting file time should result in a zero variant time.” They also provided two screen shots, which I converted to a table.

UTC+8	**UTC-8**

`ftUTC = {0,0}`
`FT2LFT returns 1`
`ftLocal = {67,237191168}`
`stLocal = 1601.1.1 08:00:00`
`vLocal = -109205.000000`
	`ftUTC = {0,0}`
	`FT2LFT returns 1`
	`ftLocal = {-68,-237191168}`
	`stLocal = 34453.15281.0 00:30:19624`
	`vLocal = 0.000000`

Incorrect	**Correct**

Okay, first of all, let’s see which is actually correct and which is incorrect.

The `FileTimeToLocalFileTime` function subtracts or adds eight hours. Since the starting time was zero, the result in the case of UTC-8 is an integer underflow, which prints as negative numbers if you use the `%d` format. (Note to language lawyers: Don't get all worked up about stuff like "passing an unsigned integer to the `%d` format results in undefined behavior." I'm talking about Win32 here, and I'm trying to explain observed behavior, not justify theoretical behavior.)

The value `{67,237191168}` corresponds to `0x00000043`0e234000`, which has the signed decimal value `288000000000` which is exactly equal to `8 * 10000 * 1000 * 3600`, or eight hours after zero. On the other hand, the value `{-68,-237191168}` corresponds to `0xffffffbc`f1dcc000` which has the signed decimal value `-288000000000` which is exactly equal to `-8 * 10000 * 1000 * 3600`, or eight hours before zero.

So far, the numbers match what we expect. Although we do have an issue that in the UTC-8 case, the value underflowed to a very large positive number.

Next, we convert `ftLocal` to `stLocal`. The easy case is UTC+8, where the timestamp of eight hours after zero is converted to January 1, 1601 at 8am, because the zero time for `FILETIME` is January 1, 1601 at midnight. This is spelled out in the very first sentence of the documentation for the `FILETIME` structure.

Okay, now the hard case of UTC-8. The timestamp `0xffffffbc`f1dcc000`, if interpreted as an unsigned number, corresponds to May 27, 58456 (at around 9:30pm), but if interpreted as a signed number, corresponds to 4pm December 31, 1600. The `FileTimeToSystemTime` function rejects negative timestamps, return `FALSE` and `ERROR_INVALID_PARAMETER`. Since the call failed, the value in `stLocal` is undefined, and here, it just contains uninitialized garbage. (Because "uninitialized garbage" is a valid value for "undefined".)

The next thing we do is convert the `stLocal` to a variant time. As noted in the documentation, the zero time for variant time is December 30, 1899. (Required reading: Eric's Complete Guide to VT_DATE, wherein the insanities of variant time are investigated.) Again, the case of UTC+8 is easy: January 1, 1601 is many many days before December 30, 1899, apparently −109205 days. I'm going to take this for granted and not check the math, because the goal is not to double-check the results but rather to explain why the results are what they are. On the other hand, the (garbage) date of the zeroth day of the 15281th month of the year 34453 is not valid, and the `SystemTimeToVariantTime` fails because the parameter is invalid. In this case, the output variable `vLocal` is left unchanged, and it continues to have the value zero, the value it was initialized with.

Therefore, the fact that in the so-called "correct" case the value of `vLocal` is zero has nothing to do with the functioning of the API, but rather has everything to do with the line

```
double vLocal = 0;
```

at the start of the program. Change the line to

```
double vLocal = 3.14159;
```

and the result in the "correct" case will be 3.14159.

The conclusion here is that the so-called "incorrect" result is actually correct, and the so-called "correct" result is just an accident. The customer is under the mistaken impression that a zero `FILETIME` matches a zero variant time, but they do not. The zero points for the two time formats are quite different. The problem was exacerbated by the fact that the test program didn't check the return values of `FileTimeToSystemTime` or `SystemTimeToVariantTime`, so what it thought were the values set by those two functions were actually just the uninitialized values passed into the respective functions.

Raymond Chen

Follow