Why can't VarDateFromStr parse back a Hungarian date that was generated by VarBstrFromDate?

devblogs.microsoft.com/oldnewthing/20161219-00

December 19, 2016

Raymond Chen

A customer liaison reported a problem with date parsing.

Ugh, date parsing.

> The customer is receiving date information from a scanner that they want to parse. They are using the `COleDateTime::ParseDateTime` method. The customer reports that clients in Hungary (locale 1038) are unable to parse dates. The call to `COleDateTime::ParseDateTime` fails with `false`. That method internally uses `VarDateFromStr`, and calling `VarDateFromStr` directly fails with the error `DISP_E_TYPEMISMATCH`.
>
> This problem is observed only for Hungarian.

The customer included a demonstration program that calls methods on `COleDateTime`, but I've stripped away the wrapper below, so we can focus on the problem better.

```
LCID hungarian = MAKELANGID(LANG_HUNGARIAN, SUBLANG_HUNGARIAN_HUNGARY);
DATE date = ...; // something
BSTR str;
hr = VarBstrFromDate(date, hungarian, VAR_DATEVALUEONLY, &str);
// The call to VarBstrFromDate succeeds and returns something like
// "2010. 12. 05". Now let's try to parse it back.

hr = VarDateFromStr(str, hungarian, VAR_DATEVALUEONLY, &date);
// The attempt to parse back to a date fails with DISP_E_TYPEMISMATCH.
```

The customer noted that this change in behavior was relatively recent.

The reason is that the localization team in Windows 10 made a change to the date formats for Hungarian. In earlier versions of Windows, the call to `VarBstrFromDate` produced `"2010.12.05"`. Notice the difference?

The date separator changed from a period to a period *followed by a space*.

This highlights that culture data is not stable. Any code that generates Hungarian-formatted dates will produce different results on Windows 10 compared to earlier versions of Windows.

Of course, one should also note that the date formatting preferences can also be customized by the user at any time, so the statement is even stronger: Any code that generates locale-sensitive formatted dates may produce different results at any time, even within a single run of the program.

So if your goal is to format the date as a string, with the intention of parsing it back, then you don't want to use anything that is locale-sensitive. Instead, use a locale-insensitive format, such as ISO 8601.

The customer said that they were getting the information from a scanner, but it wasn't clear where the scanner was getting it from.

If this is a timestamp generated by the scanner itself, then they should try to configure the scanner to generate timestamps in a locale-insensitive format.

If the timestamp is coming from the document being scanned, then you need to work out who is generating the document. If the document was generated by the same program that is trying to parse the result back (which the sample code seems to be suggesting), then you should fix the program that generates the document so it uses a locale-insensitive format. For human readability, you could have it generate a locale-sensitive version of the date next to the locale-invariant version. On the other hand, if the document was generated by an external source, then you may want to implement a custom parser that handles the date format that the external source uses.

And if you don't know what date format the external source is using, then you're kind of stuck. After all, a date of the form `12-05-2010` is ambiguous. It might be generated by somebody whose locale settings specify a date format of `MM-DD-YYYY`, or somebody whose locale settings specify a date format of `DD-MM-YYYY`.

Okay, so we've addressed the customer's problem of not being able to round-trip a date-to-string-to-date conversion. But why specifically does changing the date separator from "period" to "period and space" cause `VarDateFromStr` to be unable to parse back a string that it generated itself?

The string `2010. 12. 05.` parses back like this:

- `"2010"` is a year, no problem there.
- `". "` is a period followed by a space, no problem there.
- `"12"` is a month, no problem there.
- `". "` is a period followed by a space, no problem there.
- `"05"` is a day, no problem there.

- `"."` is a period *not* followed by a space, which does not match the date separator, so this parse is rejected.
- Next, a special-case rule for `"."` kicks in and says, "Okay, well, if normal parsing rules failed, but I see a period after a complete date, then treat it as a time separator."
- And then parsing fails, because a time separator is not allowed due to the `VAR_DATE-VALUEONLY` flag.

There is also some special-case code for Hungarian trailing period, but that code path is no longer being hit, probably because of the change from a one-character date separator to a two-character date separator.

It turns out that the date parsing code has a ton of special-case rules for various languages. (I'm looking at you, Polish, with your genitive month forms.)

Now it looks like it needs a ton plus one.

Raymond Chen

Follow