Visual Studio Magazine Online
Classic VB Corner

The Next Y2K

There's another one coming, and this one's still flying pretty far below the radar. Classic VB apps are only vulnerable to the extent they interact with others.

February 16, 2010 · by Karl E. Peterson

Ever heard the phrase "Unix time"? That's the most often-used description of the method of marking time by the number of seconds that have passed since January 1, 1970. A question arose the other day about this, and how negative values were starting to creep into VB calculations, ostensibly because of the way VB uses the high-bit to indicate sign.

Simple fix, thinks I! Just ignore the high-bit, treating the Long as unsigned. I started with some long archived routines I wrote back in the days before the Date variable type was introduced (everyone knows that ClassicVB Date variables are just Doubles in drag, right?):

```
Public Function NetTimeToVbTime(ByVal NetDate As Long) As Double
   Const BaseDate# = 25569   'DateSerial(1970, 1, 1)
   Const SecsPerDay# = 86400
   NetTimeToVbTime = BaseDate + (CDbl(NetDate) / SecsPerDay)
End Function

Public Function VbTimeToNetTime(ByVal VbDate As Double) As Long
   Const BaseDate# = 25569   'DateSerial(1970, 1, 1)
   Const SecsPerDay# = 86400
   VbTimeToNetTime = (VbDate - BaseDate) * SecsPerDay
End Function
```

The functions got their names because I was working with the various Net API functions at the time. Functions, such as NetUserGetInfo, provide dates for things such as last logon or password change using this "Unix time" method. That was okay when they were designed, because Windows as we know it didn't exist before 1970 and a DWORD holds a pretty big number of seconds.

Back to ignoring the high-bit. It turns out that if you can use only the first 31-bits, rather than all 32, the clock ticks down to **KABOOM** early in the morning on January 19, 2038. The 32nd bit effectively doubles that date with destiny, pushing it all the way out to February 7, 2106. Simple math:

```
' Clock rolls over on 1/19/2038 at 3:14:07 AM, if the
' basedate is the commonly used midnight on 1/1/1970.
Private Const BaseDate As Date = #1/1/1970#
Private Const Bit31 As Double = 2147483648#    ' 0x80000000
Private Const Bit32 As Double = 4294967296#    ' 0x100000000
```

```
Public Function NetTimeToVbTime(ByVal NetDate As Long) As Date
   Dim Seconds As Double
   Const SecsPerDay As Double = 86400
   ' Be aware that many "Unix" times are expressed in GMT, so
   ' you may also need to adjust for local offset as needed.
   ' See: http://vb.mvps.org/samples/TimeZone
   If NetDate >= 0 Then
      Seconds = CDbl(NetDate)
   Else
      Seconds = CDbl(NetDate) + Bit32
   End If
   NetTimeToVbTime = CDbl(BaseDate) + (Seconds / SecsPerDay)
End Function

Public Function VbTimeToNetTime(ByVal VbDate As Date) As Long
   Dim Seconds As Double
   ' Supports VB Dates through: 2/7/2106 6:28:15 AM
   Seconds = DateDiff("s", BaseDate, VbDate)
   If Seconds >= Bit32 Then
      ' Houston, we have a problem!
   ElseIf Seconds >= Bit31 Then
      Seconds = Seconds - Bit32
   End If
   VbTimeToNetTime = Seconds
End Function
```

Problem solved! For now, anyway, right? Well, not so fast. It turns out that, even though Windows wasn't around back then, Unix was indeed around before 1970 and the negative numbers are by design. D'oh! Of course there's a need to express dates in the past, too. The explanation offered by Dennis Ritchie, the R of K&R fame, is that he thought it'd "be nice" to be able to represent his entire lifetime. Okay, he's perhaps earned that.

So then, this is just a heads-up, for those of you unaware as I was, of this impending global hand-wringing event we'll all be witnessing at some point down the road. The C/C++ runtimes are full of the same sort of date logic, of course. Just search MSDN for January 1, 1970 for some examples. Yes, there are some provisions for 64-bit logic, but those seem to be creeping in rather slowly. Meanwhile, there's oodles of code in the wild ticking down to Y2K38, and not all of it is even patchable, much less accessible. Just consider all the embedded devices that have long-term projected life spans.

This will be a "disaster" ClassicVB folks can pretty much rubber-neck, thankfully, due to the very long range built into our Date variables. To the extent that you're interacting with alien code bearing these limitations, however, you do need to be aware. You may start seeing negative values at any point, or you may be seeing them already. If your application could be working with date values just three decades from now, and you're swapping data with outside sources, it makes sense to start testing for these potential problems now.

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.