Computing the interval between two moments in time

devblogs.microsoft.com/oldnewthing/20050414-39

April 14, 2005

Raymond Chen

Computing the interval between two moments in time is easy: It's just subtraction, but subtraction may not be what you want.

If you are displaying time units on the order of months and years, then you run into the problem that a month is of variable length. some people just take the value relative to a base date of January 1 and extract the year and month counts.

Unfortunately, this results in somewhat non-intuitive results. Let's illustrate with some examples. I'm going to write this in C# because it lets me focus on the algorithm instead of getting distracted by "oh dear how do I convert between SYSTEMTIME and FILETIME?" issues, and because it hightlights some new issues.

```
// Remember, code in italics is wrong
using System;
using SC = System.Console;
class Program {
 static void PrintAge(DateTime bday, DateTime asof)
 {
  TimeSpan span = asof - bday;
  SC.WriteLine(span);
 }
 public static void Main(string[] args) {
  DateTime bday = DateTime.Parse(args[0]);
  DateTime asof = DateTime.Parse(args[1]);
  if (bday > asof) { SC.WriteLine("not born yet"); return; }
  PrintAge(bday, asof);
 }
}
```

The two parameters to the program are the victim's birthday and the date as of which you want to compute the victim's age.

Here's a sample run:

```
> howold 1/1/2001 1/1/2002
365.00:00:00
```

Observe that the `TimeSpan` structure does not attempt to produce results in any unit larger than a day, since the authors of `TimeSpan` realized that months and years are variable-length.

A naive implementation might go like this:

```
static void PrintAge(DateTime bday, DateTime asof)
{
 TimeSpan span = asof - bday;
 DateTime dt = (new DateTime(1900, 1, 1)).Add(span);
 SC.WriteLine("{0} years, {1} months, {2} days",
              dt.Year - 1900, dt.Month - 1, dt.Day - 1);
}
```

Try it with some command lines and see what happens:

```
> howold 1/1/2001 1/1/2002
1 years, 0 months, 0 days // good
> howold 1/1/2001 3/1/2001
0 years, 2 months, 0 days // good
> howold 1/1/2000 1/1/2001
1 years, 0 months, 1 days // wrong
> howold 9/1/2000 11/1/2000
0 years, 2 months, 2 days // wrong
```

Why does it say that a person born on January 1, 2000 is one year and one day old on January 1, 2001? The person is clearly exactly one year old on that day. Similarly, it thinks that November first is two months and two days after September first, when it is clearly two months exactly.

The reason is that months and years are variable-length, but our algorithm assumes that they are constant. Specifically, months and years are context-sensitive but the algorithm assumes that they are translation-invariant. The lengths of months and years depend which month and year you're talking about. Leap years are longer than non-leap years. Months have all different lengths.

How do you fix this? Well, first you have to figure out how human beings compute the difference between dates when variable-length units are involved. The most common algorithm is to declare that one year has elapsed when the same month and day have arrived in the year following the starting point. Similarly, a month has elapsed when the same numerical date has arrived in the month following the starting point.

Mentally, you add years until you can't add years any more without overshooting. Then you add as many months as fit, and then finish off with days. (Some people subtract, but the result is the same.)

Now you get to mimic this algorithm in code.

```
static void PrintAge(DateTime bday, DateTime asof)
{
 // Accumulate years without going over.
 int years = asof.Year - bday.Year;
 DateTime t = bday.AddYears(years);
 if (t > asof) { years--; t = bday.AddYears(years); }
 // Accumulate months without going over.
 int months = asof.Month - bday.Month; // fixed 10pm
 if (asof.Day < bday.Day) months--;
 months = (months + 12) % 12;
 t = t.AddMonths(months);
 // Days are constant-length, woo-hoo!
 int days = (asof - t).Days;
 SC.WriteLine("{0} years, {1} months, {2} days",
              years, months, days);
}
```

Notice that this algorithm agrees with the common belief that people born on February 29th have birthdays only once every four years.

Exercise: Explain what goes wrong if you change the line

```
 if (t > asof) { years--; t = bday.AddYears(years); }
```

to

```
 if (t > asof) { years--; t = t.AddYears(-1); }
```

Raymond Chen

Follow