How do I generate a unique 32-bit value for a time zone?

devblogs.microsoft.com/oldnewthing/20111104-00

November 4, 2011

Raymond Chen

Public Service Announcement: Daylight Saving Time ends in most parts of the United States this weekend. Other parts of the world may change on a different day from the United States.

A customer asked the following question:

> Given two `TIME_ZONE_INFORMATION` structures, I would like to compute a `LONG` for each that I can then compare to determine whether they represent the same time zone. When I say the same, I mean that when the two are passed to `SystemTimeToTzSpecificLocalTime` with the same `LPSYSTEMTIME` input, the output is the same.

A `TIME_ZONE_INFORMATION` structure contains more information than can be packed into a 32-bit value. (At least there's no obvious way to pack it into a 32-bit value.) You're not going to be able to squeeze the entire structure into a 32-bit value that is unique for each time zone, so that comparing the 32-bit values will tell you whether the time zones are the same or not.

Fortunately, the customer also provided context for the question, explaining their underlying problem. And as is often the case, the customer had broken down the problem into two parts, one easy and one impossible. The customer solved the easy part and was asking for help with the impossible part.

But on closer inspection, the problem wasn't so much impossible as it was improperly specified:

> The bigger problem I'm actually trying to solve is that we call `SystemTimeToTzSpecific-LocalTime` inside a deeply nested loop. I would like to cache the results for performance, using the time zone as a key to a `CAtlMap` which would hold the cached results for each time zone. I'm looking for help coming up with what combinaion of the structure members to use to uniquely identify the time zone.

Okay, the customer appears to be a bit confused about hash keys. Hash keys do not need to be unique for each time zone. It is perfectly legitimate for two different items to result in the same hash value; that's why we have the term *hash collision*. Of course, you want to take

reasonable steps to minimize collisions, but when you don't control the domain space, hash collisions are a part of life.

> From looking at some time zone data, it looks like `(Bias + StandardBias)` is unique for any time zone, but I know that there are a lot of complicated issues when dealing with time zones so I wanted to check if I could be sure of that.
>
> ```
> LONG CTimeZoneTraits::GetHash(const TIME_ZONE_INFORMATION& tz)
> {
>  return tz.Bias + tz.StandardBias;
> }
> int CTimeZoneTraits::Equals(const TIME_ZONE_INFORMATION& tz1,
>                             const TIME_ZONE_INFORMATION& tz2)
> {
>  return tz1.Bias         == tz2.Bias &&
>         tz1.StandardBias == tz2.StandardBias &&
>         tz1.DaylightBias == tz2.DaylightBias &&
>         memcmp(&tz1.StandardDate, &tz2.StandardDate,
>                                   sizeof(tz1.StandardDate) &&
>         memcmp(&tz1.DaylightDate, &tz2.DaylightDate,
>                                   sizeof(tz1.DaylightDate);
> }
> ```

If you think it about it, it's clear that `(Bias + StandardBias)` does not always uniquely identify a time zone. Consider two cities at the same longitude in the same hemisphere in the middle of winter: They will have the same `StandardBias` (because they have the same longitude) and the same `Bias` (because Daylight Saving Time is not applicable during the winter), but if the cities are in different countries (or sometimes, even different parts of the same country), they will transition to/from Daylight Saving Time differently and consequently do not belong to the same time zone.

On the other hand, since this is being used simply as a hash key, uniqueness is not an absolute requirement, so even a bad hash function will still "work"; it'll just be slower than a good hash function.

If it were up to me, I would choose as a hash function something like this:

```
LONG CTimeZoneTraits::GetHash(const TIME_ZONE_INFORMATION& tz)
{
 return tz.StandardBias +
        tz.StandardDate.wDay +
        (tz.StandardDate.wDayOfWeek << 16) +
        (tz.StandardDate.wMonth << 24);
}
```

I wouldn't use the `Bias` in the hash code because the `Bias` changes over time. If the hash table lifetime extends across a daylight saving time transition, then the `Bias` will change.

For the hash, I use the `StandardBias`, which is the number of minutes east of UTC. In practice this does not exceed 60 × 25 = 1500, and it's a multiple of 30. (But not necessarily a multiple of 60.) The `wDay` is typically in the range [0,5], though it can go as high as 31 if the transition is based on a specific day. Therefore, I'll simply add it to the `StandardBias`, taking advantage of the fact that the `StandardBias` is a multiple of 30. The month and day of the week are thrown into the upper 16 bits.

Now, this hash function will still have collisions: If there are two time zones at the same longitude which transition to Standard time with the same rule, but which transition to Daylight time according to different rules, then we will still have a collision.

> I would like to reduce the number of collisions by understanding how often two equal values of `(Bias + StandardBias)` could represent different time zones.

How likely is such a collision? You can answer this question yourself: Take all the time zones currently known to the system and hash them all to see what happens. Of course, time zones change all the time, so don't assume that your results will hold true in perpetuity, but if you're just looking for a rough guide, calculating against the current state of affairs is a pretty good one. It's true that time zones change all the time, but they typically don't change by much.

Raymond Chen

Follow