Ruby master - Bug #14879

Time#+ and Time#- do not preserve receiver's utc_offset if ENV['TZ'] is modified after receiver is created

06/29/2018 04:19 AM - ioquatix (Samuel Williams)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

I have been having some problems with Time. When I add or subtract seconds, sometimes the utc_offset is changed unexpectedly.

```
$ ruby -e 'require "time"; puts Time.parse("5pm NZT")'
2018-06-29 17:00:00 +1200

$ ruby -e 'require "time"; puts Time.parse("5pm NZT") + 1'
2018-06-29 17:00:01 +1200

$ TZ=UTC ruby -e 'require "time"; puts Time.parse("5pm NZT") + 1'
2018-06-29 17:00:01 +0000
```

This seems like strange behaviour. The utc_offset shouldn't change IMHO.

History

#1 - 06/29/2018 04:47 AM - ioquatix (Samuel Williams)

I have added some specs for this behaviour here:
https://github.com/ioquatix/time-zone/blob/cdde65cd8f29d6d9fc645e2182093a7146048b44/spec/time/zone_spec.rb#L91-L99

You can see test results here: https://travis-ci.org/ioquatix/time-zone

The work around for this bug was to convert whatever is the resulting time back into the correct timezone:
https://github.com/ioquatix/time-zone/blob/cdde65cd8f29d6d9fc645e2182093a7146048b44/lib/time/zone/timestamp.rb#L79-L82

I'm open to suggestions for ways to improve this.

#2 - 06/29/2018 05:30 AM - ioquatix (Samuel Williams)

I checked the specs and JRuby doesn't have these issues, just FYI.

#3 - 06/29/2018 08:19 AM - Hanmac (Hans Mackowiak)

the problem might not be the add or substract

what does this show for you?
TZ=UTC ruby -e 'require "time"; puts Time.parse("5pm NZT")'

because on my system it shows +0000 too

#4 - 06/29/2018 08:51 AM - ioquatix (Samuel Williams)

I see, you may well be right! I will check on my end because this was supposed to be a simplified version of the behaviour I was seeing.

#5 - 06/29/2018 10:13 AM - ioquatix (Samuel Williams)

Okay, I tried more complex example which was my original repro.

```
git clone https://github.com/ioquatix/time-zone
cd time-zone

^_^ > rake console
[1] pry(main)> time, zone = Time::Zone.parse("5pm", "Pacific/Auckland")
=> [2018-06-29 17:00:00 +1200, "Pacific/Auckland"]
[2] pry(main)> time
```

```
=> 2018-06-29 17:00:00 +1200
[3] pry(main)> time + 1
=> 2018-06-29 17:00:01 +1200
[4] pry(main)> time - 1
=> 2018-06-29 16:59:59 +1200
[5] pry(main)>
^_^ > TZ=UTC rake console
[1] pry(main)> time, zone = Time::Zone.parse("5pm", "Pacific/Auckland")
=> [2018-06-29 17:00:00 +1200, "Pacific/Auckland"]
[2] pry(main)> time + 1
=> 2018-06-29 05:00:01 +0000
[3] pry(main)> time - 1
=> 2018-06-29 04:59:59 +0000
[4] pry(main)>
```

It does seem like something odd is going on here. My apologies if I've overlooked something. Thanks for your input so far.

#6 - 06/29/2018 02:24 PM - jeremyevans0 (Jeremy Evans)

As Hans correctly showed, addition/subtraction does not change the utc_offset, the utc_offset is already set to 0 before the addition/subtraction. That is because TZ=UTC and the fact that NZT is not a recognized time zone by the time library (the time library only supports a few time zones specified by ISO 8601 and RFC 2822). You can add time zones if you want to support them:

```
TZ=UTC ruby -e 'require "time"; Time.singleton_class::ZoneOffset["NZT"] = 12; puts Time.parse("5pm NZT")
2018-06-30 17:00:00 +1200
```

I think this should be closed.

#7 - 06/29/2018 09:45 PM - ioquatix (Samuel Williams)

Jeremy, thanks for your interest.

Here is a minimal repro:

```
#!/usr/bin/env ruby

# https://bugs.ruby-lang.org/issues/14879
require 'time'

ENV['TZ'] = "Pacific/Auckland"
time = Time.parse("5pm")
ENV['TZ'] = "UTC"

puts time
puts time.utc_offset
puts (time + 1).utc_offset
puts time + 1
```

Thanks for the suggestion regarding ZoneOffset. However, that does not seem sufficient for capturing DST rules?

#8 - 06/29/2018 09:46 PM - ioquatix (Samuel Williams)

Here is the output of running the minimal repro:

```
2018-06-30 17:00:00 +1200
43200
0
2018-06-30 05:00:01 +0000
```

#9 - 06/29/2018 10:17 PM - jeremyevans0 (Jeremy Evans)

- Subject changed from Adding/subtracting from time can change utc_offset unexpectedly. to Time#+ and Time#- do not preserve receiver's utc_offset if ENV['TZ'] is modified after receiver is created

ioquatix (Samuel Williams) wrote:

> Jeremy, thanks for your interest.
>
> Here is a minimal repro:
>
> ```
> #!/usr/bin/env ruby
>
> # https://bugs.ruby-lang.org/issues/14879
> require 'time'
> ```

```
ENV['TZ'] = "Pacific/Auckland"
time = Time.parse("5pm")
ENV['TZ'] = "UTC"

puts time
puts time.utc_offset
puts (time + 1).utc_offset
puts time + 1
```

> Thanks for the suggestion regarding ZoneOffset. However, that does not seem sufficient for capturing DST rules?

Correct, Time.parse doesn't support that. It uses different timezones for DST rules (e.g. PST and PDT). In your case you may want NZST and NZDT.

Your example results are due to the fact that you are changing ENV['TZ'] after creating the Time instance, which affects new time instances, and (time + 1) creates a new Time instance.

Here's a modified example:

```
require 'time'

ENV['TZ'] = "Pacific/Auckland"
time = Time.parse("5pm")
puts time
# 2018-06-30 17:00:00 +1200
puts time.utc_offset
# 43200
puts time + 1
# 2018-06-30 17:00:01 +1200
puts (time + 1).utc_offset
# 43200

ENV['TZ'] = "UTC"
time = Time.parse("5pm")
puts time
# 2018-06-29 17:00:00 +0000
puts time.utc_offset
# 0
puts time + 1
# 2018-06-29 17:00:01 +0000
puts (time + 1).utc_offset
# 0
```

I think it is unreasonable to expect Time#+ and Time#- to have special handling for the case where ENV['TZ'] is modified after the Time instance is created, but I'll admit that it is subjective whether the methods should preserve the receiver's utc_offset in such cases. I would say it shouldn't be expected, because Time#+ and Time#- modifies utc_offset when crossing DST boundaries, and keeping the same utc_offset would lead to undesired behavior:

```
Time.now
# => 2018-06-29 15:13:22 -0700
Time.now - 86400*200
# => 2017-12-11 14:13:30 -0800
```

I'm updating the subject to more accurately reflect the issue.

#10 - 06/29/2018 10:58 PM - ioquatix (Samuel Williams)

> In your case you may want NZST and NZDT.

Expecting users to know the time zone in advance is not feasible. User should be able to say "This date, time at this location" and it computes the correct offset. Anyway, it's a separate issue.

> Your example results are due to the fact that you are changing ENV['TZ'] after creating the Time instance, which affects new time instances, and (time + 1) creates a new Time instance.

Yes that is what I suspected.

> I think it is unreasonable to expect Time#+ and Time#- to have special handling for the case where ENV['TZ'] is modified after the Time instance is created, but I'll admit that it is subjective whether the methods should preserve the receiver's utc_offset in such cases. I would say it shouldn't be expected, because Time#+ and Time#- modifies utc_offset when crossing DST boundaries, and keeping the same utc_offset would lead to

undesired behavior.

I think there are two separate cases you discuss. The first one being whether Time#+/- should retain the utc_offset, and whether Time#+/- should be able to change utc_offset when going cross DST changes. You've used the second case to argue against the first.

I think the 1st case is a bug.

I think the 2nd case is almost a bug but in theory is reasonable behaviour.

The problem is, -0700 is a offset from UTC, and doesn't encode enough information to relate to DST rules AFAIK. Let's use Canada as an example. Some areas are -7 hours all year around, other areas use -6 hours for DST. So, how do you know enough information what case it is, considering you only have -0700? (https://en.wikipedia.org/wiki/Time_in_Canada)

So, I'd be interested to know where those DST rules are coming from, because that stuff is hard to encode, subject to the whims of governments, and changes at the drop of a vote.

Coming back to the bug report, I don't think adding/subtracting seconds should change the utc_offset, unless the user has actually specified a physical time zone on which DST rules are known (e.g. "Pacific/Auckland") and the new time crosses a DST boundary which affects the utc_offset. Every other case where the UTC offset changes, I'd assert, is a bug.

#11 - 06/29/2018 11:07 PM - ioquatix (Samuel Williams)

By the way, perhaps it's not clear, but utc_offset is not a time zone, nor is Time.zone, because just stating MST is not enough to disambiguate. You need to specify Canada/Saskatchewan (MST only) or Canada/Mountain (MST/MDT). Practically speaking, the best computation we have for utc_offset is the time zone database (i.e. government rules), a specific date+time in UTC, and a specific zone name from said database. That can then give us the utc_offset. Since several time zones might map to the same UTC offset, it's not easy to go in reverse without additional help.

#12 - 06/19/2019 07:55 PM - jeremyevans0 (Jeremy Evans)

Using Ruby 2.6's new timezone support for Time, I think you can get the behavior you want, so that time calculations do not change based on TZ.

```
ENV['TZ'] = "Pacific/Auckland"
time = Time.at(Time.parse("5pm"), in: TZInfo::Timezone.get('Pacific/Auckland'))
ENV['TZ'] = "UTC"

time
# => 2019-06-20 17:00:00 +1200
time + 1
# => 2019-06-20 17:00:01 +1200
time.utc_offset
# => 43200
(time + 1).utc_offset
# => 43200
```

#13 - 08/13/2019 05:27 AM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed