Ruby master - Bug #10677

Regression: Time#parse no longer automatically converts to localtime

12/30/2014 01:30 AM - parkr (Parker M)

Status:	Rejected		
Priority:	Normal		
Assignee:	zzak (Zachary Scott)		
Target version:	2.2.0		
ruby -v:	ruby 2.2.0p0 (2014-12-25 revision 49005) [x86_64-darwin14]	**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description

In Ruby 2.1 and before, Time#parse automatically converted to the localtime:

Ruby 2.1:

```
>> require 'time'
=> true
>> ENV['TZ'] = 'Australia/Melbourne'
=> "Australia/Melbourne"
>> Time.parse("2014-12-29 20:16:32 -0400")
=> 2014-12-30 11:16:32 +1100
```

But in Ruby 2.2, this is not the case:

```
>> require 'time'
>> ENV['TZ'] = 'Australia/Melbourne'
>> Time.parse("2014-12-29 20:16:32 -0400")
=> 2014-12-29 20:16:32 -0400 # !!
>> Time.parse("2014-12-29 20:16:32 -0400").localtime
=> 2014-12-30 11:16:32 +1100
```

This seems to be a regression, as this is a change in default behaviour without a MAJOR version bump, violating semver.

History

#1 - 12/30/2014 03:10 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

It's a bug fix.

#2 - 12/30/2014 03:36 AM - parkr (Parker M)

Ok, thanks Nobuyoshi. Is this documented somewhere? It caused a lot of strife for me, and I don't think I'll be the only one. I wrote a short post about it as I had no clue this was scheduled as a bug fix. I'll update my post with a link to the original bug and the related fixes. Thanks!

#3 - 01/01/2015 03:21 PM - akr (Akira Tanaka)

This change is intentional to preserve the original information.

Please use localtime method if you need a Time object in your local time.

#4 - 01/02/2015 09:58 PM - parkr (Parker M)

I hear you, Akira. I am asking for a link to the issue or conversation that tracked this change. I want to know why the change was made in more detail, so I would like to read the discussion.

#5 - 01/02/2015 10:42 PM - akr (Akira Tanaka)

There is no direct issue.

It is inspired by [Bug #9794].

#6 - 01/03/2015 07:01 PM - binarylogic (Ben Johnson)

Akira Tanaka wrote:

> There is no direct issue.
>
> It is inspired by [Bug #9794].

Can we start a discussion? I'm surprised major changes like this are made without one. I didn't see anything in the changelogs either. This will undoubtedly cause significant problems. I've tried upgrading and came across issues with rails, multiple gems, etc.

#7 - 01/04/2015 06:50 AM - binarylogic (Ben Johnson)

Akira Tanaka wrote:

> There is no direct issue.
>
> It is inspired by [Bug #9794].

I'd also like to add that Parker's post, and the explanation in the bottom half, is spot on (https://byparker.com/blog/2014/ruby-2-2-0-time-parse-localtime-regression/).

I have a strong feeling this is going to be a **major** problem as people try to move forward. Adding "local" time everywhere you use Time.parse simply is not feasible. This change is also outside of the scope of a "minor" version change for ruby.

Finally, the amount of dependencies a typical ruby project has (gems). There is so much of this behavior we can not control. I can't even think of a backwards compatible patch, as gems start to adopt this new behavior, I'm not sure how we'd distinguish the "before" and "after" gems.

#8 - 01/04/2015 07:47 AM - nobu (Nobuyoshi Nakada)

Ben Johnson wrote:

> I have a strong feeling this is going to be a **major** problem as people try to move forward. Adding "local" time everywhere you use Time.parse simply is not feasible. This change is also outside of the scope of a "minor" version change for ruby.

I strongly disagree.
It's a bug that had ignored a part of the input which should not be ignored, and should be fixed as usual.

#9 - 01/05/2015 03:58 AM - naruse (Yui NARUSE)

It seems a small issue that no one notice before 2.2.0 released

You may know Ruby sometimes break compatibility to achieve better specs/functions.
For Time class it didn't save its time zone before.
After 1.9 whose time objects can save its timezone, Time methods supports time zone gradually.
This change is considered as a part of them.
see also https://www.ruby-lang.org/en/news/2013/12/21/ruby-version-policy-changes-with-2-1-0/

Of course such changes may cause trouble.
Therefore we provide preview releases and release candidates to test changes.
Since Travis supports such previews, you can test your applications easily.
http://rubies.travis-ci.org/
Heroku also supports preview versions.
You can report if a change is too large.
We may withdraw the change or provide more graceful transition.

Thanks,

#10 - 01/05/2015 05:05 AM - binarylogic (Ben Johnson)

Thank you for the explanation. I'll continue to debug and see if I can help measure it's impact. I more clearly understand the issue, and agree with the change. Unfortunately, I feel it's going to have a bigger impact than anticipated. If so, we should discuss a backwards compatible method of introducing this change.

#11 - 01/29/2015 11:59 AM - rohandaxini (Rohan Daxini)

Ben Johnson wrote:

> Thank you for the explanation. I'll continue to debug and see if I can help measure it's impact. I more clearly understand the issue, and agree with the change. Unfortunately, I feel it's going to have a bigger impact than anticipated. If so, we should discuss a backwards compatible method of introducing this change.

Something similar happened to me, but we use Time.zone.parse.
Refer the ticket here - https://bugs.ruby-lang.org/issues/10758

#12 - 03/07/2015 04:21 PM - blmlcu (Luca B)

Yui NARUSE wrote:

> After 1.9 whose time objects can save its timezone,

I welcome the change, even if it's backward incompatible.
However, I noticed that although the object is meant to save its own timezone, it does not return it any more:

```
require 'time'
                                                # 1.9.3 -> 2.1.5 | 2.2.0 and 2.2.1
                                                # ---------------|-----------------
p Time.parse("2014-12-31 20:16:32 -0400").zone  #    "GMT"       |        nil
ENV['TZ'] = "Australia/Melbourne"               #                |
p Time.parse("2014-12-31 20:16:32 -0400").zone  #    "AEDT"      |        nil
```

It seems to me that to be consistent with the spirit of the change, #zone should still return the same value.
Is this a bug? If not, what is the reason?

#13 - 03/08/2015 03:26 AM - nobu (Nobuyoshi Nakada)

You don't provide *timezone*, but *offset* only.
Try utc_offset.