
business_time



ActiveSupport gives us some great helpers so we can do things like:

```
1 5.days.ago
```

and

```
1 8.hours.from_now
```

as well as helpers to do that from any provided date or time.

I needed this, but taking into account business hours/days and holidays.

Usage

install the gem

```
1 gem install business_time
```

open up your console

```
1 # if in irb, add these lines:
2
3 require 'business_time'
4
5 # try these examples, using the current time:
6
7 1.business_hour.from_now
8 4.business_hours.from_now
9 8.business_hours.from_now
10
11 1.business_hour.ago
12 4.business_hours.ago
13 8.business_hours.ago
14
15 1.business_day.from_now
16 4.business_days.from_now
17 8.business_days.from_now
18
19 1.business_day.ago
20 4.business_days.ago
21 8.business_days.ago
22
```

```
23 Date.today.workday?
24 Date.parse("2015-12-09").workday?
25 Date.parse("2015-12-12").workday?
```

And we can do it from any Date or Time object.

```
1 my_birthday = Date.parse("August 4th, 1969")
2 8.business_days.after(my_birthday)
3 8.business_days.before(my_birthday)
4
5 my_birthday = Time.parse("August 4th, 1969, 8:32 am")
6 8.business_days.after(my_birthday)
7 8.business_days.before(my_birthday)
```

We can adjust the start and end time of our business hours

```
1 BusinessTime::Config.beginning_of_workday = "8:30 am"
2 BusinessTime::Config.end_of_workday = "5:30 pm"
```

Or we can temporarily override the configured values

```
1 BusinessTime::Config.with(beginning_of_workday: "8 am", end_of_workday:
  "6 pm") do
2   1.business_hour.from_now
3 end
```

and we can add holidays that don't count as business days

July 5 in 2010 is a monday that the U.S. takes off because our independence day falls on that Sunday.

```
1 three_day_weekend = Date.parse("July 5th, 2010")
2 BusinessTime::Config.holidays << three_day_weekend
3 friday_afternoon = Time.parse("July 2nd, 2010, 4:50 pm")
4 tuesday_morning = 1.business_hour.after(friday_afternoon)
```

plus, we can change the work week:

```
1 # July 9th in 2010 is a Friday.
2 BusinessTime::Config.work_week = [:sun, :mon, :tue, :wed, :thu]
3 thursday_afternoon = Time.parse("July 8th, 2010, 4:50 pm")
4 sunday_morning = 1.business_hour.after(thursday_afternoon)
```

As alternative we also can change the business hours for each work day:

```
1 BusinessTime::Config.work_hours = {
2   mon: ["9:00", "17:00"],
3   fri: ["9:00", "17:00"],
4   sat: ["10:00", "15:00"]
5 }
6 friday = Time.parse("December 24, 2010 15:00")
7 monday = Time.parse("December 27, 2010 11:00")
```

```
8 working_hours = friday.business_time_until(monday) # 9.hours
```

You can also calculate business duration between two dates

```
1 friday = Date.parse("December 24, 2010")
2 monday = Date.parse("December 27, 2010")
3 friday.business_days_until(monday) #=> 1
```

Or you can calculate business duration between two Time objects

```
1 ticket_reported = Time.parse("February 3, 2012, 10:40 am")
2 ticket_resolved = Time.parse("February 4, 2012, 10:50 am")
3 ticket_reported.business_time_until(ticket_resolved) #=> 8.hours + 10.
  minutes
```

You can also determine if a given time is within business hours

```
1 Time.parse("February 3, 2012, 10:00 am").during_business_hours?
```

Note that counterintuitively, durations might not be quite what you expect when involving weekends. Consider the following example:

```
1 ticket_reported = Time.parse("February 3, 2012, 10:40 am")
2 ticket_resolved = Time.parse("February 4, 2012, 10:40 am")
3 ticket_reported.business_time_until(ticket_resolved) # will equal 6
  hours and 20 minutes!
```

Why does this happen? Feb 4 2012 is a Saturday. That time will roll over to Monday, Feb 6th 2012, 9:00am. The business time between 10:40am friday and 9am monday is 6 hours and 20 minutes. From a quick inspection of the code, it looks like it should be 8 hours.

Or you can calculate business dates between two dates

```
1 monday = Date.parse("December 20, 2010")
2 wednesday = Date.parse("December 22, 2010")
3 monday.business_dates_until(wednesday) #=> [Mon, 20 Dec 2010, Tue, 21
  Dec 2010]
```

You can get the first workday after a time or return itself if it is a workday

```
1 saturday = Time.parse("Sat Aug 9, 18:00:00, 2014")
2 monday = Time.parse("Mon Aug 11, 18:00:00, 2014")
3 Time.first_business_day(saturday) #=> "Mon Aug 11, 18:00:00, 2014"
4 Time.first_business_day(monday) #=> "Mon Aug 11, 18:00:00, 2014"
5
6 # similar to Time#first_business_day Time#previous_business_day only
  cares about
7 # workdays:
8 saturday = Time.parse("Sat Aug 9, 18:00:00, 2014")
9 monday = Time.parse("Mon Aug 11, 18:00:00, 2014")
```

```
10 Time.previous_business_day(saturday) #=> "Fri Aug 8, 18:00:00, 2014"
11 Time.previous_business_day(monday)   #=> "Mon Aug 11, 18:00:00, 2014"
```

Rails generator

```
1 rails generate business_time:config
```

The generator will add a `./config/business_time.yml` and a `./config/initializers/business_time.rb` file that will cause the start of business day, the end of business day, and your holidays to be loaded from the yml file.

You might want to programatically load your holidays from a database table, but you will want to pay attention to how the initializer works - you will want to make sure that the initializer sets stuff up appropriately so rails instances on mongrels or passenger will have the appropriate data as they come up and down.

Timezone support

This gem strives to be timezone-agnostic. Due to some complications in the handling of timezones in the built in Time class, and some complexities (bugs?) in the `timeWithZone` class, this was harder than expected... but here's the idea:

- When you configure the gem with something like 9:00am as the start time, this is agnostic of time zone.
- When you are dealing with a Time or TimeWithZone class, the timezone is preserved and the beginning and end of times for the business day are referenced in that time zone.

This can lead to some weird looking effects if, say, you are in the Eastern time zone but doing everything in UTC times... Your business day will appear to start and end at 9:00 and 5:00 UTC.

If this seems perplexing to you, I can almost guarantee you are in over your head with timezones in other ways too, this is just the first place you encountered it.

Timezone relative date handling gets more and more complicated every time you look at it and takes a long time before it starts to seem simple again.

Integration with the Holidays gem

Chris Wise wrote up a great article[<http://murmurinfo.wordpress.com/2012/01/11/handling-holidays-and-business-hours/>] on using the `business_time` gem with the `holidays`[<https://github.com/alexduane/holidays>] gem. It boils down to this:

```
1 Holidays.between(Date.civil(2013, 1, 1), 2.years.from_now, :ca_on, :
   observed).map do |holiday|
2   BusinessTime::Config.holidays << holiday[:date]
3   # Implement long weekends if they apply to the region, eg:
4   # BusinessTime::Config.holidays << holiday[:date].next_week if !
     holiday[:date].weekday?
5 end
```

You can pass holidays as an option In addition to having static holidays in `BusinessTime::Config.holidays`, you can also pass holidays as an option (date, time, string, or array of dates, times and strings). If, for example, you are processing calculations for many regions with different holidays, you can pass in the appropriate holidays for the region with each calculation.

```
1 three_day_weekend = Date.parse("July 5th, 2010")
2 friday_afternoon = Time.parse("July 2nd, 2010, 4:50 pm")
3 tuesday_morning = 1.business_hour.after(friday_afternoon, holidays:
   three_day_weekend)
```

Contributors

- David Bock <http://github.com/bokmann>
- Ryan McGeary <http://github.com/rmm5t>
- Enrico Bianco <http://github.com/enricob>
- Arild Shirazi <http://github.com/ashirazi>
- Piotr Jakubowski <http://github.com/piotrj>
- Glenn Vanderburg <http://github.com/glv>
- Michael Grosser <http://github.com/grosser>
- Michael Curtis <http://github.com/mcurtis>
- Brian Ewins <http://github.com/bazzargh>
- Mstate <http://github.com/mstate>
- Caden Westmoreland <http://github.com/cadenforrest>

(Special thanks for Arild on the complexities of dealing with `TimeWithZone`)

Note on Patches/Pull Requests

- Fork the project.
- Make your feature addition or bug fix.
- Add tests for it. This is important so I don't break it in a future version unintentionally.
- Commit, do not mess with rakefile, version, or history. (if you want to have your own version, that is fine but bump version in a commit by itself I can ignore when I pull)

-
- Send me a pull request. Bonus points for topic branches.

TODO

- Arild has pointed out that there may be some logical inconsistencies regarding the `beginning_of_workday` and `end_of_workday` times not actually being considered inside of the workday. I'd like to make sure that they work as if the `beginning_of_workday` is included and the `end_of_workday` is not included, just like the `'...'` range operator in Ruby.

NOT TODO

- I spent way too much time in my previous java-programmer life building frameworks that worshipped complexity, always trying to give the developer-user ultimate flexibility at the expense of the 'surface area' of the api. Never again - I will sooner limit functionality to 80% so that something stays usable and let people fork.
- While there have been requests to add 'business minutes' and even 'business seconds' to this gem, I won't entertain a pull request with such things. If you find it useful, great. Most users won't, and they don't need the baggage.

A note on stability and change

Sometimes people ask me why this gem doesn't release more often. My opinions on that are best discussed in person in a friendly discussion, but I'll attempt some of that here.

First, a big part of the reason is that the projects I do use this gem on are happy with it's current functionality. It is 'suitable for the purpose' for which I released it, and as such, maintenance I do on this gem is a gift to the community.

Second, out of the ~1.3 million downloads (according to rubygems.org), the number of real 'issues' with this gem have been minimal. Most of the issues that are opened are really people with slightly different requirements than I have regarding whether 'off hours' work counts as the previous or the next business day, a disagreement on the semantics of days vs. hours, etc. I take care to try to explain these choices in the open issues, but to my mind, they aren't true issues if it's just a difference of opinion. Even so, I'll gladly accept pull requests that resolve this difference of opinion as a configuration option... just don't expect me to do your job for you. I've already given you 90% of what you need.

Third, a business time gem is, well, relevant to businesses. Many businesses don't move quickly. My government clients move even more slowly. Stability is favored in these environments.

Fourth, new features can wait. To the person that adds them they can be mission critical, but with modern packaging processes, they can use their version without waiting for their changes to be included in the upstream version. Their changes don't break your code.

I'm proud of the work in this gem; the stability is a big part of that. This gem has lived longer than many others that have attempted to do the same thing. I expect it to be here chugging away when Ruby has become the next COBOL.

Copyright

Copyright (c) 2010-2022 bokmann. See LICENSE for details.