Getting To Know My Enemies
Time Classes and Time Zones in Rails

B'more on Rails
December 11, 2025

Rosanna Speller

github.com/r-spell
gitlab.com/r-spell
linkedin.com/in/rlspeller

These slides are at http://bit.ly/44Y6iYf

About me

- Learned to code in Python & Java
- 8 years working in Rails (with some JS)
  - 6 years as the "Rails Expert" of my team
  - Work at a smallish tech non-profit (launchcode.org)
- This is my first tech talk!

My enemies

- Timezones
- Daylight savings time
- The letter S

Versions

- Rails 8.1
- Ruby 3.4

Examples

Roughly based on Rails 8.1 "Getting Started" App[1]

- An online store
- Products and subscribers

[1] Rails Guide Getting Started

Application Config

boiler plate info

In `config/application.rb`

```
# These settings can be overridden in specific
↪  environments using the files
# in config/environments, which are processed later.
#
# config.time_zone = "Central Time (US & Canada)"
# config.eager_load_paths << Rails.root.join("extras")
```

Application Config

Set CA Time Zone

In `config/application.rb`

```
    # These settings can be overridden in specific
    ↪  environments using the files
    # in config/environments, which are processed later.
    #
    config.time_zone = "Pacific Time (US & Canada)"
    # config.eager_load_paths << Rails.root.join("extras")
```

Application Config

Set CA Time Zone

Image from https://imgflip.com/memegenerator/7923908/fire-girl

date, datetime, time in the database migration

```ruby
class AddColumnsToProducts < ActiveRecord::Migration[8.1]
  def change
    add_column :products, :launch_date, :date
    add_column :products, :launch_datetime, :datetime
    add_column :products, :launch_time, :time
  end
end
```

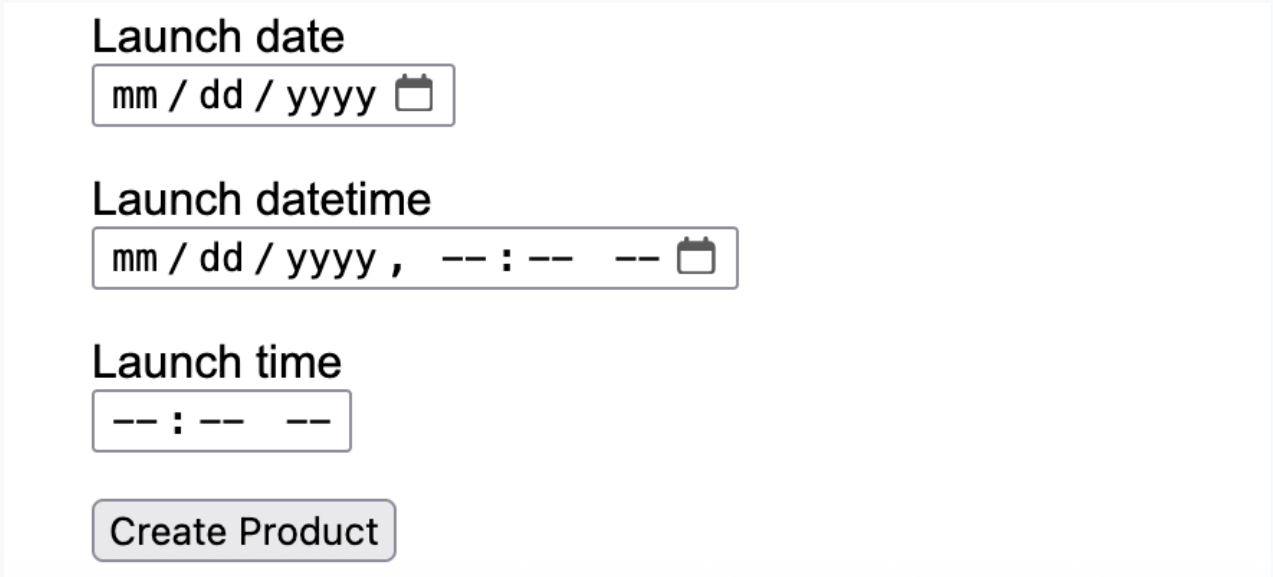
date, datetime, time in the database schema

```
# db/schema.rb
# ...
  create_table "products", force: :cascade do |t|
    t.datetime "created_at", null: false
    t.integer "inventory_count"
    t.date "launch_date"
    t.datetime "launch_datetime"
    t.time "launch_time"
    t.string "name"
    t.datetime "updated_at", null: false
  end
```

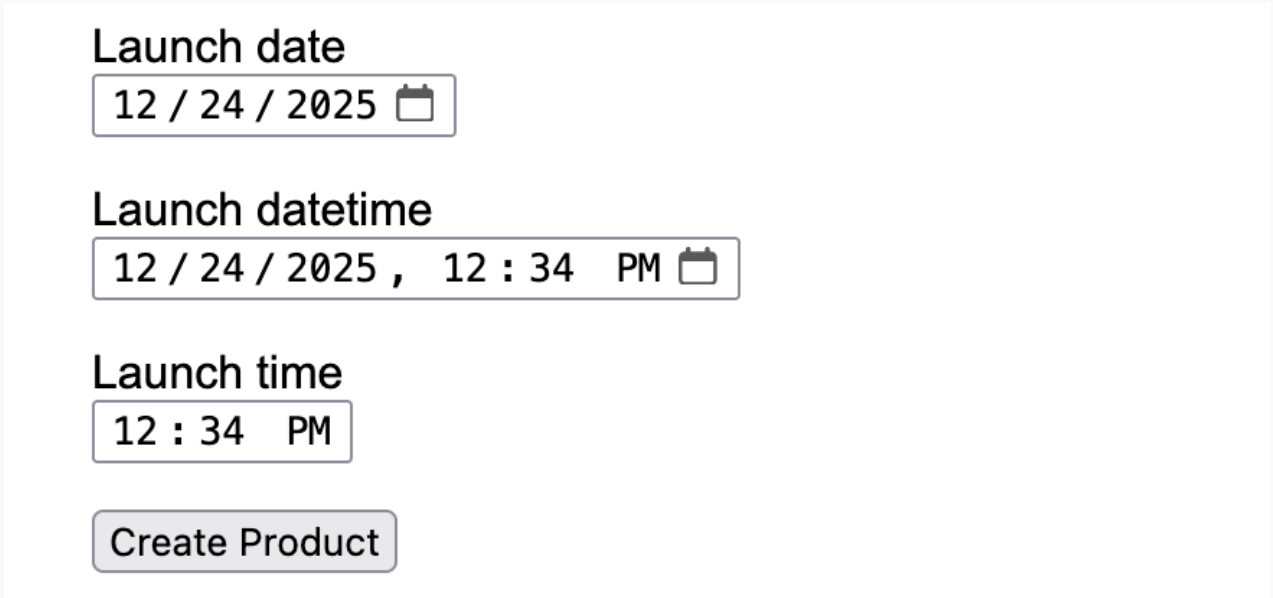
date, datetime, time in the form helpers

```
<div>
  <%= form.label :launch_date, style: "display: block" %>
  <%= form.date_field :launch_date %>
</div>

<div>
  <%= form.label :launch_datetime, style: "display: block"
  <%= form.datetime_field :launch_datetime %>
</div>

<div>
  <%= form.label :launch_time, style: "display: block" %>
  <%= form.time_field :launch_time %>
</div>
```

date, datetime, time in the form helpers (part 2)



date, datetime, time in the form helpers (part 3)



date, datetime, time what about the parameters?

Using the Rails debug gem:[2]

```
[45, 53] in ~/projects/timewarp/app/controllers/products_controller.rb
    45|
    46|     def product_params
    47|       x = params.expect(product:
    48|         [ :name,:description, :featured_image, :inventory_count,
    ↪   :launch_date, :launch_datetime, :launch_time ]
    49|       )
=>  50|       debugger
    51|       x
    52|     end
    53| end
```

[2] Rails Guide: Debug Gem

date, datetime, time what about the parameters?

See the params x using the debugger:

```
=>  50|       debugger
    51|       x
    52|     end
    53| end
=>#0        ProductsController#product_params at
↪  ~/projects/timewarp/app/controllers/products_controller.rb:50
  #1        ProductsController#create at
  ↪  ~/projects/timewarp/app/controllers/products_controller.rb:17
  # and 81 frames (use `bt' command for all frames)
(rdbg) x
#<ActionController::Parameters {"name" => "Example Product", "description" =>
↪  "<div>Blah</div>", "inventory_count" => "2", "launch_date" => "2025-12-24",
↪  "launch_datetime" => "2025-12-24T12:34", "launch_time" => "12:34"}
↪  permitted: true>
(rdbg)
```

date, datetime, time what about the parameters?

Use command `continue` to complete the action:

```
=>  50|        debugger
    51|        x
    52|      end
    53|  end
=>#0        ProductsController#product_params at
↪  ~/projects/timewarp/app/controllers/products_controller.rb:50
  #1        ProductsController#create at
  ↪  ~/projects/timewarp/app/controllers/products_controller.rb:17
  # and 81 frames (use `bt' command for all frames)
(rdbg) x
#<ActionController::Parameters {"name" => "Example Product", "description" =>
↪  "<div>Blah</div>", "inventory_count" => "2", "launch_date" => "2025-12-24",
↪  "launch_datetime" => "2025-12-24T12:34", "launch_time" => "12:34"}
↪  permitted: true>
(rdbg) continue    # command
  TRANSACTION (0.0ms)  BEGIN immediate TRANSACTION
  ↪  /*action='create',application='Timewarp',controller='products'*/
```

date, datetime, time in the Rails Console

```
timewarp(dev):001> Product.last
  Product Load (0.3ms)  SELECT "products".* FROM "products"
    ↪  ORDER BY "products"."id" DESC LIMIT 1
    ↪  /*application='Timewarp'*/
=>
#<Product:0x0000000129b8fd60
 id: 10,
 created_at: "2025-12-08 10:09:53.662802000 -0800",
 name: "Example Product",
 updated_at: "2025-12-08 10:09:53.668152000 -0800",
 inventory_count: 2,
 launch_date: "2025-12-24",
 launch_datetime: "2025-12-24 12:34:00.000000000 -0800",
 launch_time: "2000-01-01 12:34:00.000000000 -0800">
```

Classes

Date, DateTime, Time

We looked at the form inputs and column types:

- `date`
- `datetime`
- `time`

There's also Classes:

- `Date`
- `DateTime`
- `Time`

Date, DateTime, Time

In Ruby and Rails

`Date`, `DateTime` and `Time` are Ruby classes.[3]

Rails has `ActiveSupport` core extensions for these 3 classes.[4]

[3] Ruby Docs for `Date`, `DateTime` and `Time`

[4] Rails Docs for `Date`, `DateTime` and `Time`

Date, DateTime, Time in the console

```
timewarp(dev):013> date = Date.today
=> Sun, 07 Dec 2025
timewarp(dev):014> date_time = DateTime.now
=> Sun, 07 Dec 2025 17:03:13 -0500
timewarp(dev):015> time = Time.now
=> 2025-12-07 17:03:17.282426 -0500
```

(I made the examples on Sunday)

Date, DateTime, Time in the console (cont.)

```
timewarp(dev):013> date = Date.today
=> Sun, 07 Dec 2025
timewarp(dev):014> date_time = DateTime.now
=> Sun, 07 Dec 2025 17:03:13 -0500
timewarp(dev):015> time = Time.now
=> 2025-12-07 17:03:17.282426 -0500
timewarp(dev):016> date.acts_like?(:date)
=> true
timewarp(dev):017> date.acts_like?(:time)
=> false
timewarp(dev):018> date_time.acts_like?(:date)
=> true
timewarp(dev):019> date_time.acts_like?(:time)
=> true
timewarp(dev):020> time.acts_like?(:time)
=> true
```

Date, DateTime, Time in the console (cont.)

```
timewarp(dev):013> date = Date.today
=> Sun, 07 Dec 2025
timewarp(dev):014> date_time = DateTime.now
=> Sun, 07 Dec 2025 17:03:13 -0500
timewarp(dev):015> time = Time.now
=> 2025-12-07 17:03:17.282426 -0500
timewarp(dev):016> date.acts_like?(:date)
=> true
timewarp(dev):017> date.acts_like?(:time)
=> false
timewarp(dev):018> date_time.acts_like?(:date)
=> true
timewarp(dev):019> date_time.acts_like?(:time)
=> true
timewarp(dev):020> time.acts_like?(:time)
=> true
timewarp(dev):021> time.acts_like?(:date)
=> false
```

`DateTime` acts like a date and a time, so that must be the best one, if I want to give a date and a time, right?

NO!

`DateTime` is actually **deprecated!**

DateTime vs. Time

Prefer Time to DateTime

At least as far back as Ruby 3.0, Ruby says:

> *`DateTime` class is considered deprecated. Use `Time` class.* [5]

[5] Ruby Docs

DateTime vs. Time

Except when you prefer DateTime

Current (3.4) Ruby Docs still say that `DateTime` is preferable to `Time` for **historical** dates.

See Ruby Docs for some interesting examples![6]

[6] Read the Ruby Docs!

DateTime vs. Time

Why different?

`DateTime`

- subclass of `Date`
- can use old calendars (eg. Gregorian vs. Julian)
- doesn't deal with leap seconds or day light savings[7]

`Time`

- based internally on nanoseconds since "Unix Epoch"
- mostly deals with leap seconds
- deals with day light savings[8]

---

[7] Ruby `DateTime` Docs
[8] Ruby `Time` Docs

date, datetime, time

back in the console

Let's look at the classes of the values we have for `launch_date`, `launch_time` and `launch_datetime`.

```
timewarp(dev):011> product = Product.last
timewarp(dev):012> product.launch_date.class
=> Date
timewarp(dev):013> product.launch_time.class
=> ActiveSupport::TimeWithZone
```

date, datetime, time

back in the console

Let's look at the classes of the values we have for `launch_date`, `launch_time` and `launch_datetime`.

```
timewarp(dev):011> product = Product.last
timewarp(dev):012> product.launch_date.class
=> Date
timewarp(dev):013> product.launch_time.class
=> ActiveSupport::TimeWithZone
timewarp(dev):014> product.launch_datetime.class
=> ActiveSupport::TimeWithZone
```

TimeWithZone

What's the zone?

`launch_time` and `launch_datetime` are Rails `ActiveSupport::TimeWithZone`s.

Let's look at the `zone`s:

```
timewarp(dev):013> product.launch_time.class
=> ActiveSupport::TimeWithZone
timewarp(dev):014> product.launch_datetime.class
=> ActiveSupport::TimeWithZone
timewarp(dev):015> product.launch_datetime.zone
=> "PST"
timewarp(dev):016> product.launch_time.zone
=> "PST"
```

Time.now and Time.current in the console

```
timewarp(dev):017> now = Time.now
=> 2025-12-07 20:18:32.345682 -0500
timewarp(dev):018> now.class
=> Time
timewarp(dev):019> current = Time.current
=> 2025-12-07 17:18:50.066319000 PST -08:00
timewarp(dev):020> current.class
=> ActiveSupport::TimeWithZone
timewarp(dev):021> current.zone
=> "PST"
```

Time.now and Time.current in the console cont.

```
timewarp(dev):017> now = Time.now
=> 2025-12-07 20:18:32.345682 -0500
timewarp(dev):018> now.class
=> Time
timewarp(dev):019> current = Time.current
=> 2025-12-07 17:18:50.066319000 PST -08:00
timewarp(dev):020> current.class
=> ActiveSupport::TimeWithZone
timewarp(dev):021> current.zone
=> "PST"
timewarp(dev):022> now.zone
=> "EST"
```

Time.zone

`Time.zone`, by default, will use the `TimeZone` set in the application config.[9]

[9] See `TimeZone` Documentation. Also look at the definition of the `zone` method in the Rails `Time` documentation. It is also possible to set it as something else on a per request basis.

Time.current and Time.zone.now

`Time.current` is generally the same as `Time.zone.now`.

`Time.current` is slightly superior, in that it also handles a scenario where somehow `Time.zone` is `nil`[10]

```
timewarp(dev):001> current = Time.current
=> 2025-12-07 06:26:14.626656000 PST -08:00
timewarp(dev):002> time_in_app_zone = Time.zone.now
=> 2025-12-07 06:26:17.387557000 PST -08:00
```

[10] See definition of `current`

TimeWithZone and TimeZones

Let's see the times in some specified time zones.[11]

```
timewarp(dev):001> current = Time.current
=> 2025-12-07 06:26:14.626656000 PST -08:00
timewarp(dev):002> time_in_app_zone = Time.zone.now
=> 2025-12-07 06:26:17.387557000 PST -08:00
timewarp(dev):003> now_in_central =
↪   Time.now.in_time_zone("Central Time (US & Canada)")
=> 2025-12-07 08:26:37.305944000 CST -06:00
timewarp(dev):004> now_in_central.zone
=> "CST"
timewarp(dev):005> now_in_tehran =
↪   Time.now.in_time_zone("Tehran")
=> 2025-12-07 17:57:08.617554000 +0330 +03:30
timewarp(dev):006> now_in_tehran.zone
=> "+0330"
```

(I made the examples on Sunday)

[11] See TimeZone docs for options

How many possible times/timezones for any one time in any one app?

4

(At least 4)

It can be helpful to think of there being 4 main kinds of time/ time zones[12]

[12] I first saw these 4 in this thoughtbot article but I've also looked into Rails docs to find info which backs this all up

4 Kinds of Times

1. system time
2. application time
3. database time
4. the user's time

4 Kinds of Times

We've already seen the first 2

1. system time
   - My computer timezone is EST
   - An instance of "regular" `Time` had EST `zone`
2. application time
   - We configured our application timezone to be Pacific Time
   - An instance of `TimeWithZone` (without zone specified) had the default, PST, `zone`

More on System time

System time will be the time on the server/ computer/ machine where the app is running. (Can vary!)

Not sure that Rails uses the term "system time" but I think it's what Rails will refer to as:

- `"ENV['TZ']"`
- "Ruby's *process* timezone" [13]
- `:local` (not to be confused with the method `local` on `TimeZone`)[14]

---

[13] I can talk more about this if there's time, but I say this based info here and here

[14] Docs on method `local`

More on database time

Database time by default is in UTC time.[15]

I lean towards leaving it this way.[16]

[15] Rails config guide

[16] See discussion here for issues with changing it

More on user time

You can set specific times for your users.[17]

[17] See Rails example of how to set here

Enough time chaos!

What should we do?

Recommendations

Use `Time` (or `TimeWithZone`) instead of `DateTime`

There's a `rubocop DateTime` cop to help enforce this[18]

[18] See Rubocop Docs about `DateTime` cop

Recommendations

Use `ActiveSupport::TimeWithZone`) over just generic Ruby `Time`.

There is a `rubocop-rails TimeZone` cop that can be enabled to enforce this.[19]

[19] See Rubocop Rails Docs about `TimeZone` cop

Recommendations

Use `datetime` column type for your migration when you want a date and a time.

Note: The default settings for timestamps in Rails adds them `datetime`s in the schema and are `TimeWithZone` values.[20]

[20] See schema examples above, and Timestamp Docs

Recommendations

Remember that you stand on the shoulders of giants who have also struggled with time and time zones!

Thank you!! Questions?

These Beamer slides made with a customized version of the theme Arguelles