What’s the time Mr Matsumoto?

Why is this important?

Thinking across timezones is hard

Think global

UTC isn’t enough

Some common ground

What’s UTC?

Coordinated Universal Time (blame the French)

Successor to GMT

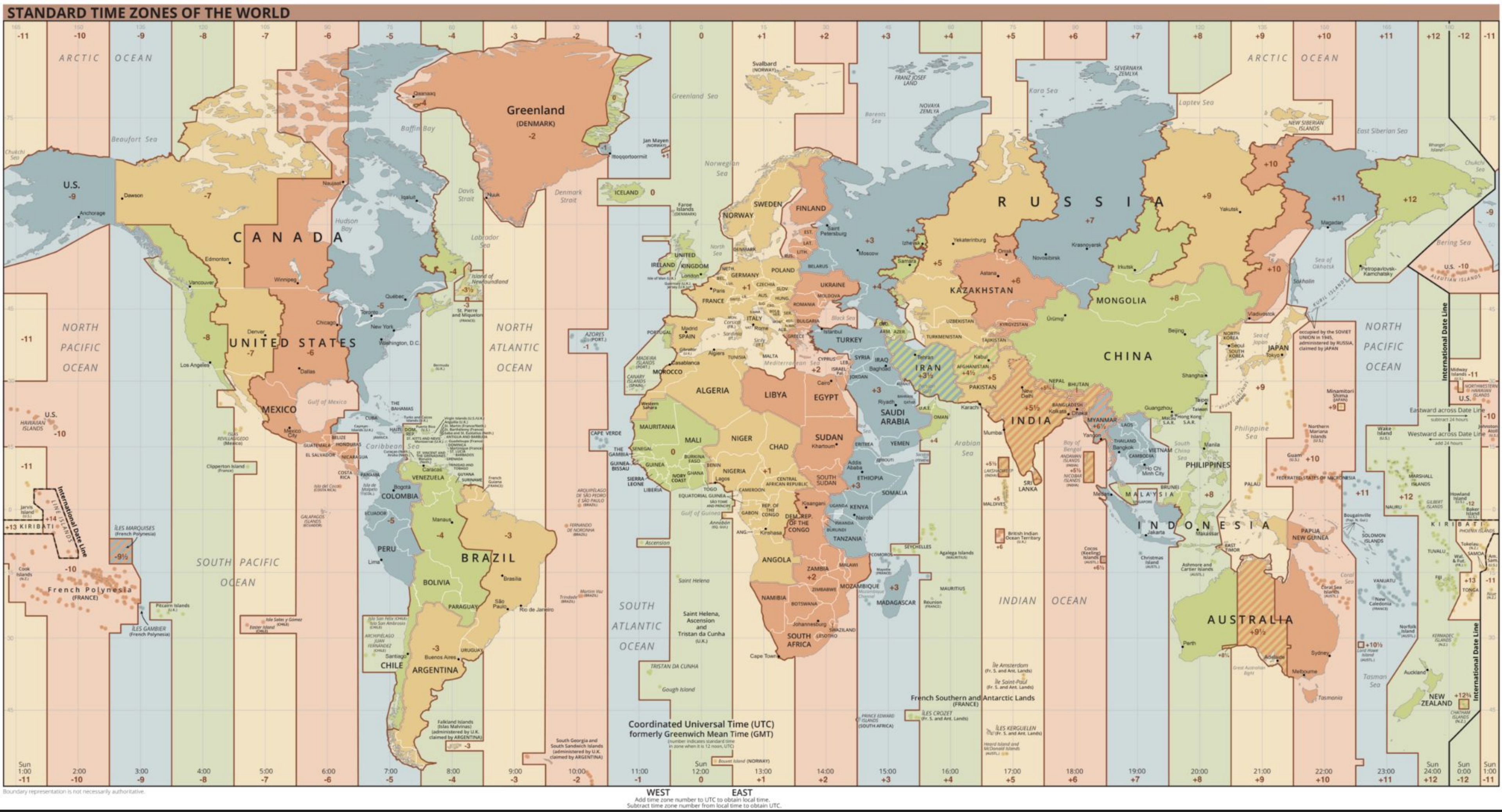
A datum timezone at 0° longitude

Is not adjusted for DST

Is commonly used by computers (e.g. crontab)

Is rarely used by humans

At a given time we can map timezones to UTC offsets, but these are <u>not</u> constant



Attribution: TimeZonesBoy, Public domain, via Wikimedia Commons

Some common ground

For processes that don’t map to real world activity, UTC is generally fine

For example, I want to run this job once an hour

For processes that map to human activity, not so much

For example, I want to run this job at 02:00

02:00 where?

Does it need to adjust for DST?

What happens if it runs early/late?

What happens if 02:00 doesn’t exist?

What tools do we have in Ruby?

Time
Date
DateTime (deprecated in favour of Time)

Note that this does not include a TimeZone class

```
> time = Time.now
=> 2023-04-06 16:25:08.108773 +0100

> time.zone
=> "BST"

> require 'date'
> date = Date.today
=> #<Date: 2023-04-06 ((2460041j,0s,0n),+0s,2299161j)>
```

Enter ActiveSupport

```
> require 'active_support'
> require 'active_support/time'
> Time.zone
=> nil
> Time.zone = Time.find_zone!("UTC")
> Time.zone
=> #<ActiveSupport::TimeZone:0x0000000105f456b8
     @name="UTC",
     @tzinfo=#<TZInfo::DataTimezone: Etc/UTC>,
     @utc_offset=nil>
> Time.zone.now
=> Thu, 06 Apr 2023 15:32:55.127999000 UTC +00:00
> Time.zone.now + 1.day + 1.hour
=> Fri, 07 Apr 2023 16:33:05.759982000 UTC +00:00
> Time.zone.now.in_time_zone("Australia/Sydney")
=> Fri, 07 Apr 2023 01:34:12.413260000 AEST +10:00
```

What tools do we have in ActiveSupport?

```
ActiveSupport::TimeZone
ActiveSupport::TimeWithZone
core_ext Time
core_ext Date
core_ext DateTime
```

Configuring our timezone

```
> require 'active_support'
> require 'active_support/time'
> Time.zone
=> nil

> Time.zone = Time.find_zone!("UTC")
> Time.zone.class
=> ActiveSupport::TimeZone

> Time.zone.now
=> Thu, 06 Apr 2023 15:32:55.127999000 UTC +00:00

> Time.zone.now.class
=> ActiveSupport::TimeWithZone
```

A note on timezones

```
Where do timezones come from?

> tree /usr/share/zoneinfo/
/usr/share/zoneinfo/
├── +VERSION
├── Africa
│   ├── Abidjan
│   ├── Accra
│   ├── Addis_Ababa
│   ├── Algiers
│   ├── Asmara
│   ├── Asmera
│   ├── Bamako
│   ├── Bangui
│   ├── Banjul
│   ├── Bissau
│   ├── Blantyre
```

What do these tell us?

```
> zdump -v /usr/share/zoneinfo/Europe/London | tail -n 6
/usr/share/zoneinfo/Europe/London  Sun Mar 29 00:59:59 2037 UTC =
Sun Mar 29 00:59:59 2037 GMT isdst=0
/usr/share/zoneinfo/Europe/London  Sun Mar 29 01:00:00 2037 UTC =
Sun Mar 29 02:00:00 2037 BST isdst=1
/usr/share/zoneinfo/Europe/London  Sun Oct 25 00:59:59 2037 UTC =
Sun Oct 25 01:59:59 2037 BST isdst=1
/usr/share/zoneinfo/Europe/London  Sun Oct 25 01:00:00 2037 UTC =
Sun Oct 25 01:00:00 2037 GMT isdst=0
/usr/share/zoneinfo/Europe/London  Mon Jan 18 03:14:07 2038 UTC =
Mon Jan 18 03:14:07 2038 GMT isdst=0
/usr/share/zoneinfo/Europe/London  Tue Jan 19 03:14:07 2038 UTC =
Tue Jan 19 03:14:07 2038 GMT isdst=0
```

If you’re curious about the last two lines, Google the “Epochalypse” (yes, that’s a thing)
By default TZInfo actually uses the tzinfo-data gem as its source
You can force it to use zoneinfo with TZInfo::DataSource.set(:zoneinfo)

Accessible via the tzinfo gem

```
> require "tzinfo"
> london = TZInfo::Timezone.get("Europe/London")
=> #<TZInfo::DataTimezone: Europe/London>
> london.canonical_identifier
=> "Europe/London"
> london.observed_utc_offset
=> 3600
> london.now
=> 2023-04-07 22:25:16.077745 +0100
```

Also available on an ActiveSupport::TimeZone:

```
> london = Time.find_zone!("Europe/London")
> london.tzinfo
=> #<TZInfo::DataTimezone: Europe/London>
```

Accessible via the tzinfo gem

```
# The TimeZone class serves as a wrapper around TZInfo::Timezone instances.
# It allows us to do the following:
#
# * Limit the set of zones provided by TZInfo to a meaningful subset of 134
#   zones.
#
# * Retrieve and display zones with a friendlier name
#   (e.g., "Eastern Time (US & Canada)" instead of "America/New_York").
#
# * Lazily load TZInfo::Timezone instances only when they're needed.
#
# * Create ActiveSupport::TimeWithZone instances via TimeZone's local,
#   parse, at, and now methods.
```

https://github.com/rails/rails/blob/main/activesupport/lib/active_support/values/time_zone.rb

What about ActiveRecord?

```
> rails generate model widget \
  title:string utc_timestamp:timestamp tz_timestamp:timestamptz

---

class CreateWidgets < ActiveRecord::Migration[7.0]
  def change
    create_table :widgets do |t|
      t.string :title
      t.timestamp :utc_timestamp
      t.timestamptz :tz_timestamp

      t.timestamps
    end
  end
end
```

What about ActiveRecord?

```
     Column     |              Type
----------------+-------------------------------
 id             | bigint
 title          | character varying
 utc_timestamp  | timestamp without time zone
 tz_timestamp   | timestamp with time zone
 created_at     | timestamp(6) without time zone
 updated_at     | timestamp(6) without time zone
```

What about ActiveRecord?

```
rails_test_development=> SELECT * FROM widgets;
-[ RECORD 1 ]-+-------------------------------
id            | 1
title         | my_widget
utc_timestamp | 2023-04-08 20:07:34.545792
tz_timestamp  | 2023-04-08 21:07:34.545792+01

rails_test_development=> SHOW TIMEZONE;
-[ RECORD 1 ]-----------
TimeZone | Europe/London
```

What about ActiveRecord?

```
now = Time.find_zone!("Australia/Sydney").now

Widget.create(
  title: "my_widget",
  tz_timestamp: now,
  utc_timestamp: now,
)

#<Widget:0x0000000106d14780
 id: 1,
 title: "my_widget",
 utc_timestamp: Sat, 08 Apr 2023 20:07:34.545792000 UTC +00:00,
 tz_timestamp: Sat, 08 Apr 2023 20:07:34.545792000 UTC +00:00,
 created_at: Sat, 08 Apr 2023 20:07:57.445968000 UTC +00:00,
 updated_at: Sat, 08 Apr 2023 20:07:57.445968000 UTC +00:00>
```

What about ActiveRecord?

```
Widget.create(
  title: "my_widget",
  tz_timestamp: Time.find_zone!("Europe/London").now,
  utc_timestamp: Time.find_zone!("Australia/Sydney").now,
)

INSERT INTO "widgets" ("title", "utc_timestamp", "tz_timestamp", "created_at",
"updated_at")
VALUES ($1, $2, $3, $4, $5) RETURNING "id"
[
  ["title", "my_widget"],
  ["utc_timestamp", "2023-04-27 15:25:04.389264"],
  ["tz_timestamp", "2023-04-27 15:25:04.382907"],
  ["created_at", "2023-04-27 15:25:04.392253"],
  ["updated_at", "2023-04-27 15:25:04.392253"]
]
```

What about ActiveRecord?

```
> Time.zone = Time.find_zone!("Australia/Melbourne")
> Widget.last
  Widget Load (1.1ms)  SELECT "widgets".* FROM "widgets" ORDER BY "widgets"."id"
DESC LIMIT $1  [["LIMIT", 1]]
=>
#<Widget:0x0000000106d77308
 id: 1,
 title: "my_widget",
 utc_timestamp: Fri, 28 Apr 2023 01:25:04.389264000 AEST +10:00,
 tz_timestamp: Fri, 28 Apr 2023 01:25:04.382907000 AEST +10:00,
 created_at: Fri, 28 Apr 2023 01:25:04.392253000 AEST +10:00,
 updated_at: Fri, 28 Apr 2023 01:25:04.392253000 AEST +10:00>
```

What about ActiveRecord?

```
> ActiveRecord::Base.transaction(requires_new: true) do
    ActiveRecord::Base.connection.execute("SET TIMEZONE TO 'Australia/Sydney';")
    Widget.last
  end
=>
#<Widget:0x000000010793e880
 id: 1,
 title: "my_widget",
 utc_timestamp: Thu, 27 Apr 2023 15:25:04.389264000 UTC +00:00,
 tz_timestamp: Thu, 27 Apr 2023 15:25:04.382907000 UTC +00:00,
 created_at: Thu, 27 Apr 2023 15:25:04.392253000 UTC +00:00,
 updated_at: Thu, 27 Apr 2023 15:25:04.392253000 UTC +00:00>
```

What about ActiveRecord?

```
rails_test_development=> SET TIMEZONE TO 'Europe/London';
SET

rails_test_development=> SELECT * FROM widgets;
-[ RECORD 1 ]-+------------------------------
id            | 1
title         | my_widget
utc_timestamp | 2023-04-08 20:07:34.545792
tz_timestamp  | 2023-04-08 21:07:34.545792+01
```

What about ActiveRecord?

```
rails_test_development=> SET TIMEZONE TO 'Australia/Sydney';
SET

rails_test_development=> SELECT * FROM widgets;
-[ RECORD 1 ]-+--------------------------------
id            | 1
title         | my_widget
utc_timestamp | 2023-04-08 20:07:34.545792
tz_timestamp  | 2023-04-09 06:07:34.545792+10
```

What about in SQL?

```
=> INSERT INTO widgets (id, title, utc_timestamp, tz_timestamp, created_at,
updated_at) VALUES (999, 'test_widget', '2011-01-01 00:00:00+03', '2011-01-01
00:00:00+03', NOW(), NOW());

rails_test_development=> SELECT * FROM widgets WHERE id = 999;
-[ RECORD 1 ]-+---------------------------
id            | 999
title         | test_widget
utc_timestamp | 2011-01-01 00:00:00
tz_timestamp  | 2010-12-31 21:00:00+00
created_at    | 2023-04-08 23:21:07.424189
updated_at    | 2023-04-08 23:21:07.424189

=> show timezone;
-[ RECORD 1 ]-----------
TimeZone | Europe/London
```

What about in SQL?

```
-- Add timezone to a timestamp without zone
> SELECT '2023-05-01 12:00'::timestamp AT TIME ZONE 'Europe/London';
        timezone
------------------------
 2023-05-01 12:00:00+01

-- Convert timestamp with zone to the timezone
> SELECT '2023-05-01 12:00+01'::timestamptz AT TIME ZONE 'Europe/London';
      timezone
---------------------
 2023-05-01 12:00:00
```

Note that our database timezone is still Europe/London, so that’s the display timezone

What about in SQL?

```
-- Add timezone to a timestamp without zone
> SELECT '2023-05-01 12:00'::timestamp AT TIME ZONE 'America/New_York';
        timezone
------------------------
 2023-05-01 17:00:00+01
(1 row)

-- Convert timestamp with zone to the timezone
> SELECT '2023-05-01 12:00+01'::timestamptz AT TIME ZONE 'America/New_York';
      timezone
---------------------
 2023-05-01 07:00:00
(1 row)
```

Best Practices - use ActiveSupport

```
require "active_support"
require "active_support/time"

Time.zone = Time.find_zone!("UTC")
```

Things to avoid - manual UTC offsetting

Especially don’t do this:

```
> Time.find_zone!("Europe/London").utc_offset
=> 0

> Date.today
=> Fri, 07 Apr 2023

> Timecop.freeze("01 November 2023") do
    Time.find_zone!("Europe/London").utc_offset
  end
=> 0
```

This is equivalent to the constant:

```
> Time.find_zone!("Europe/London").tzinfo.base_utc_offset
=> 0
```

Things to avoid - manual UTC offsetting

```
If you have to do this, use observed_utc_offset:

> Time.find_zone!("Europe/London").tzinfo.observed_utc_offset
=> 3600

> Date.today
=> Fri, 07 Apr 2023

> Timecop.freeze("01 November 2023") do
    Time.find_zone!("Europe/London").tzinfo.observed_utc_offset
  end
=> 0
```

Things to avoid - .parse methods

These are just…really awful

Things to avoid - .parse methods

```
> Date.parse("Monday 26rd September 2016")
=> #<Date: 2016-09-26 ((2457658j,0s,0n),+0s,2299161j)>

> Date.parse("fri1feb3bc4pm+5")
=> #<Date: -0002-02-01 ((1720359j,0s,0n),+0s,2299161j)>

> Date.parse("2O15-O2-01")
=> #<Date: 2022-09-15 ((2459838j,0s,0n),+0s,2299161j)>

> Date.parse("20189")
=> Tue, 07 Jul 2020
```

Things to avoid - .parse methods

```
> Date.iso8601("2022-01-01")
=> Sat, 01 Jan 2022

> Time.zone.iso8601("2022-01-01T12:00")
=> Sat, 01 Jan 2022 12:00:00.000000000 UTC +00:00

> Time.zone.now.iso8601
=> "2023-04-08T15:07:06Z"

> Time.zone.strptime("8 April 2023, 12:00", "%d %B %Y, %k:%M")
=> Sat, 08 Apr 2023 12:00:00.000000000 UTC +00:00
```

Things to avoid - implying timezones, bad conversion

```
> Time.iso8601("2022-06-17T16:00").in_time_zone("Europe/London")
=> Fri, 17 Jun 2022 16:00:00.000000000 BST +01:00
> Time.zone.iso8601("2022-06-17T16:00").in_time_zone("Europe/London")
=> Fri, 17 Jun 2022 17:00:00.000000000 BST +01:00

> Time.find_zone!("Europe/London").iso8601("2022-06-17T16:00")
=> Fri, 17 Jun 2022 16:00:00.000000000 BST +01:00
> Time.zone.iso8601("2022-06-17T16:00")
=> Fri, 17 Jun 2022 16:00:00.000000000 UTC +00:00

> Time.now
=> 2023-04-07 22:40:34.782764 +0100

> Time.local(2023, 1, 1).zone
=> "GMT"
> Time.zone.name
=> "UTC"
```

Best Practices - use TimeWithZone

```
now = Time.zone.now

now_in_london = Time.find_zone!("Europe/London").now
tomorrow = now_in_london.tomorrow
two_hours_ago = now - 2.hours
```

Best Practices - don’t pass around Dates

```
> beginning_of_day = Time.find_zone!("Australia/Sydney").now.beginning_of_day
=> Fri, 07 Apr 2023 00:00:00.000000000 AEST +10:00
> date = beginning_of_day.to_date
=> Fri, 07 Apr 2023

# in some other code

> now_somewhere_else = Time.zone.iso8601(date.iso8601)
=> Fri, 07 Apr 2023 00:00:00.000000000 UTC +00:00
> beginning_of_day.to_i - now_somewhere_else.to_i
=> -36000
```

Best Practices - use strict methods

```
# don’t use this - returns nil
Time.find_zone("Foo/Bar")

# don’t use this - returns nil
ActiveSupport::TimeZone.new("Foo/Bar")

# raises TZInfo::InvalidTimezoneIdentifier
TZInfo::Timezone.get("Foo/Bar")

# raises ArgumentError
Time.find_zone!("Foo/Bar")
```

Best Practices - use strict methods

Using methods that raise errors should always be preferred, as if you aren't checking your return values you can end up with more insidious bugs:

```
> london = Time.find_zone("Europe/Londn")
=> nil
> Time.find_zone!("UTC").now.in_time_zone(london)
=> 2023-04-06 11:11:03.449456 UTC
```

vs.

```
> london = Time.find_zone!("Europe/Londn")
lib/active_support/core_ext/time/zones.rb:85:in `find_zone!': Invalid Timezone:
Europe/Londn (ArgumentError)
```

Best Practices - use specific names

Prefer specific timezone names, which will also be enforced by using strict methods

Avoid using legacy names

Prefer 'America/New_York' to 'US/Eastern' or 'EST5EDT'

This also applies when discussing/specifying timezones

Referring to 'Europe/London' as GMT or 'America/New_York' as EST is wrong about 50% of the year

Best Practices - test in multiple timezones

Don’t fall into the trap of Europe/London or UTC being the “one true timezone”

Test that time-sensitive code works correctly in multiple timezones

Make use of Timecop to account for DST and any other relevant changes over time

Make use of the internet or tzinfo to find out when DST happens

Time’s up!