

Home → Articles

How to parse a time or date in Go

Published Thu 23 Nov, 2023

GO/GOLANG

TIME

Every Go developer trips up on it at some point: parsing a date or time. If you haven't done it before, it can be a frustrating experience.

But.. you're not alone. There are a lot of developers who share your frustration:



"Time parsing is a joke"



"we should open a support group"



"truly awful indeed"

"I always forget if it was Monday or not"

So why does this go so wrong for so often?

It's usually a combination of:

- Misunderstanding the way layouts work.
- The cryptic error messages returned by `time.Parse` when an invalid layout

is provided.

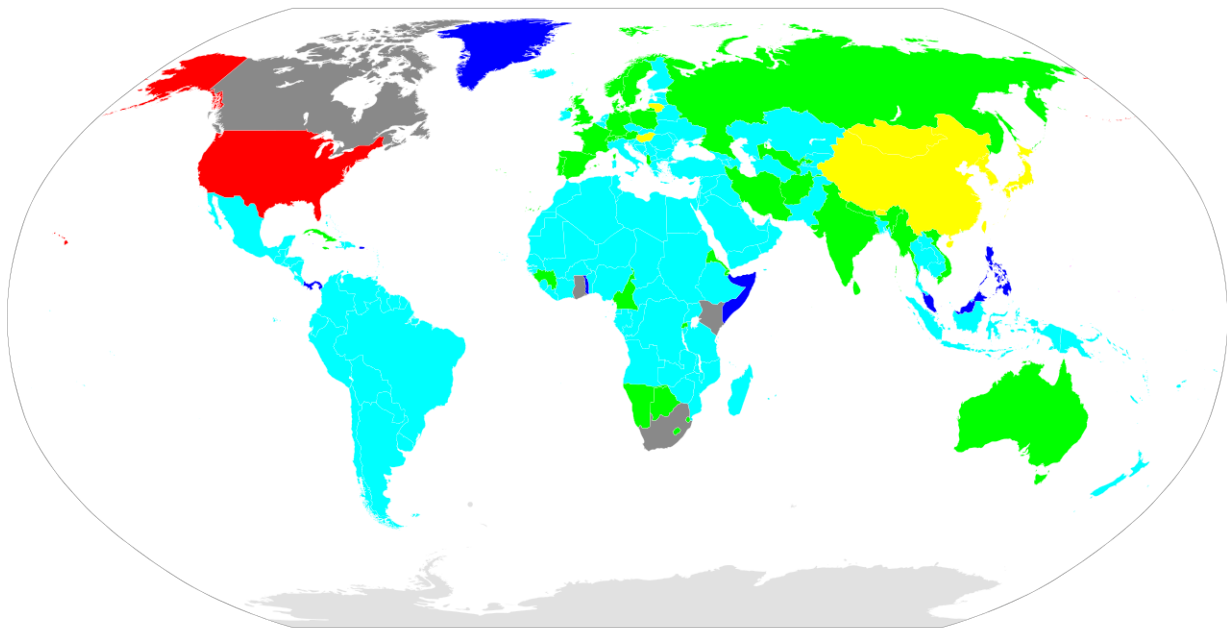
This article addresses both.

But first, some background.

Time parsing and layouts

You can't just pass a raw string to your program and expect it to know what exact time or date it should be.

This doesn't just go for programs, if I gave you the text "5/11/2023" you can interpret that as "5th of November 2023" or "11th of May 2023" depending on which notation you think I used.



Notations by country according to [Wikipedia](#).


This goes for parsers as well, they need to know which “notation” is used in order to get the right result. This “notation” is usually provided as a **layout consisting of symbols**.

These symbols indicate the expected format and position of an element of a

date or time.

For example, symbols might look like this:

- %d to indicate the day of the month.
- %m to indicate the month number.
- %Y to indicate the year, including century.

 These symbols are used by the parsing function in the Standard C library (`libc`). A lot of other languages call out to this library. You can find more symbols [here](#).

To parse a string, we need to create a layout (a “notation”) that places these symbols in the right place.

For example, parsing the string "5/11/2023" with:

- "%d/%m/%Y" as a layout results in the date "5th of November 2023" .
- "%m/%d/%Y" as a layout results in the date "11th of May 2023" .

Most languages use these kind of symbols to define layouts:

- [PHP](#), [Java](#) and [.NET](#) use their own symbols.
- Ruby allows you to [use the libc symbols](#).

Go does things *somewhat different*.

Parsing time in Go

Parsing time in Go is done using the `time.Parse` function. Looking at its signature you will see that it accepts a layout and a raw input parameter:

```
func Parse(layout, value string) (Time, error)
```

However, when we call this function, things get funky:

```
package main

import (
    "fmt"
    "log"
    "time"
)

func main() {
    t, err := time.Parse("1/2/2006", "5/11/2023")
    if err != nil {
        log.Fatal(err)
    }

    year, month, day := t.Date()
    fmt.Printf("year: %d, month: %d, day: %d", year, month, day)
}
```

RUN

FORMAT

Go v1.21

RESET

OUTPUT

What is going on here?

Is "1/2/2006" an example we're providing as a layout?

No.

`time.Parse` doesn't use symbols in its layout, it uses elements of a **reference time**.

The reference time

This reference time is **specifically chosen** so that every element (minutes,

hours, days etc) has a different value. Because every element has an unique value, the parser can use these values as “symbols” to identify which part goes where.

From the example above we can deduce the following values (and their equivalent libc-style symbol):

libc symbol	Go value	Description
"%d"	"2"	The day of the month
"%m"	"1"	The month number
"%Y"	"2006"	The year, including century

This also means that the Go `"01/02/2006"` layout is equivalent to `"%m/%d/%Y"` libc-style layout.

Let’s take a look at the full reference time:

```
// in the time package

// The reference time, in numerical order.
const Layout = "01/02 03:04:05PM '06 -0700"
```

If we write this out we get `"January 2, 15:04:05, 2006, in time zone seven hours west of GMT"`.

At a first glance this reference time might seem random, but it’s not. When you format this time according to American convention (month/day/year), the values increase from low to high.

 It's noted in [the documentation](#) that:

"It is a regrettable historic error that the date uses the American convention of putting the numerical month before the day."

This [came to be](#) because one of Go's designers (Rob Pike) did not realize his computer was configured to use an American locale. He ran the `date`

command locally and used the output from that.

The “reference time” way of specifying a layout is somewhat controversial and can lead to mistakes if you’re not used to it.

More formats

The "1", "2" and "2006" are not the only values we can use. The elements of the reference time can be written as different values to parse different formats.

Element	Possible values
Year	"2006", "06"
Month	"Jan", "January", "01", "1"
Day of the week	"Mon", "Monday"
Day of the month	"2", "_2", "02"
Day of the year	"__2", "002"
Hour	"15", "3", "03" (PM or AM)
Minute	"4", "04"
Second	"5", "05"
AM/PM mark	"PM"

A complete overview (including timezones) can be found in [the documentation](#).

Below we use values from the above table to parse an English month and two-digit year from a string.

```
package main

import (
    "fmt"
    "log"
    "time"
)

func main() {
    t, err := time.Parse("January 06", "March 19")
    if err != nil {
```

RUN

FORMAT

Go v1.21

RESET

OUTPUT

Trouble remembering?

You might want to try out Goland if you have trouble remembering the reference time and possible values.

Goland might help if you're used to symbols. It provides [automatic suggestions](#) that can convert from symbols to reference time values.

Common mistakes

Mistakes happen, and unfortunately, `time.Parse` does not always return clear error messages.

If you run into cryptic errors like:

```
cannot parse "X" as "Y"
```

It's usually one of the following two issues.

Layout is NOT an example

If you're unfamiliar with the reference time it can look like "random" example that `time.Parse` somehow turns into layout. But that's not how it works, you need to **use the specific reference time values** in your layout.

For example, the following example needs to use "01/02/2006" as a layout to make it work.

```
package main

import (
    "fmt"
    "log"
    "time"
)

func main() {
    t, err := time.Parse("9/4/1992", "5/22/2023")
    if err != nil {
        log.Fatal(err)
    }

    year, month, day := t.Date()
    fmt.Printf("year: %d, month: %d, day: %d", year, month, day)
}
```

RUN

FORMAT

Go v1.21

RESET

OUTPUT

Switching layout and input

When you call `time.Parse` with two literal values it can be difficult to glance which input is the layout and which one is the value. The **first parameter** should always be the layout and use the reference time.

In the example below I have switched the layout and input, switch them back to fix the error.

```
package main

import (
    "fmt"
    "log"
    "time"
)
```


RUN

FORMAT

Go v1.21

RESET

OUTPUT

Outro

I hope this article clears up some possible misconceptions around
`time.Parse`. Feel free to reach out if you have any questions or need help :)

Stay in the loop,
subscribe to my newsletter.

Email...

SUBSCRIBE

No spam, ever.

[Privacy policy](#)

```
for _, f := range []string{
    "🧑‍🔬 Exercices and solutions",
    "🔥 Subscriber-only content",
    "📧 New posts in your inbox",
    "💰 Discounts",
} {
    you.Enjoy(f)
}
```



Hello! I'm the Willem behind willem.dev

I created this blog to help newcomers to Go and/or web development. I hope it brings you some value :)

Thanks for reading!