

Contents

1 Introduction	2
2 Usage	2
3 Examples	2
3.1 Some groups and sequences	2
3.2 Lifelines	3
3.3 Found and lost messages	4
3.4 Custom images	5
4 Reference	6
4.1 Participants	6
4.1.1 _par	6
4.1.2 SHAPES	7
4.2 Sequences	8
4.2.1 _evt	8
4.2.2 _seq	8
4.2.3 EVENTS	11
4.2.4 tips	11
4.2.5 comment-align	12
4.3 Groups	13
4.3.1 _grp	13
4.3.2 _sync	14
4.4 Gaps and separators	15
4.4.1 _gap	15
4.4.2 _sep	15
4.5 Notes	16
4.5.1 _note	16
4.5.2 SHAPES	17
4.5.3 SIDES	17

1 Introduction

This package lets you create nice sequence diagrams using the CeTZ package.

2 Usage

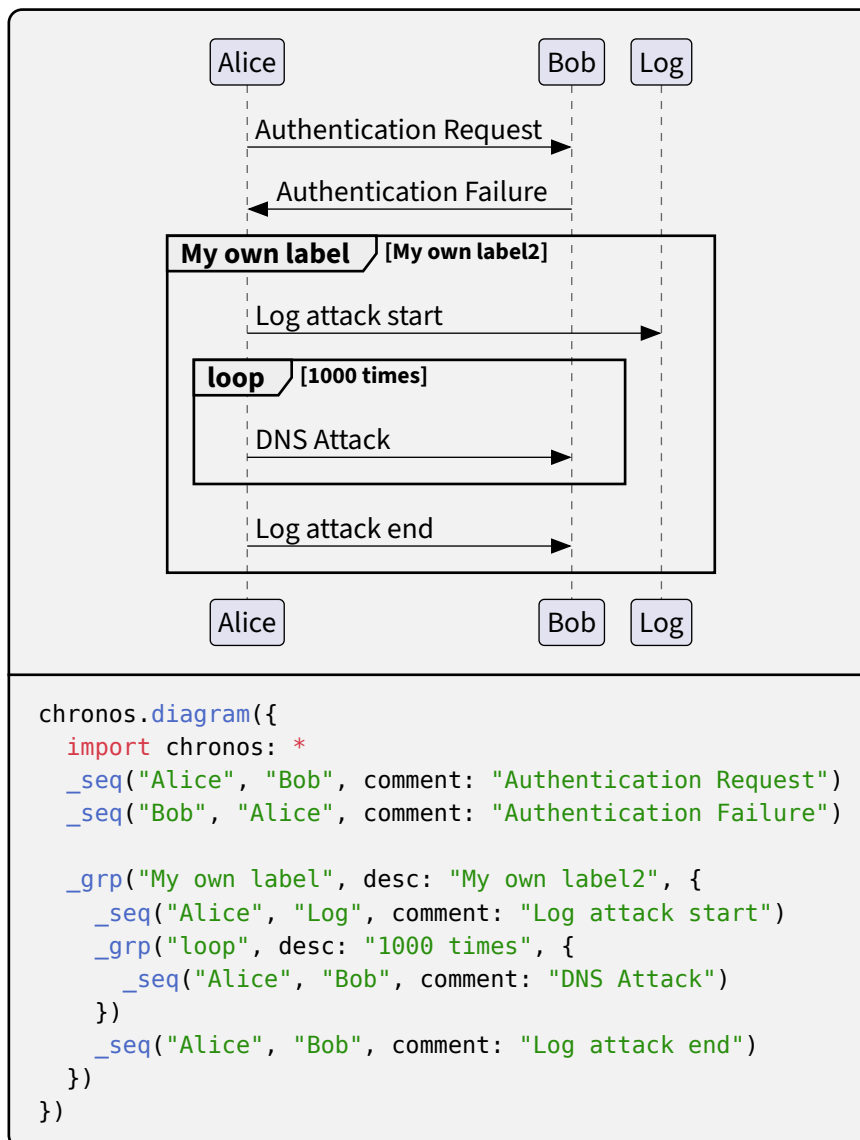
Simply import `chronos` and call the `diagram` function:

```
#import "@preview/chronos:0.1.0"
#chronos.diagram({
  import chronos: *
  ...
})
```

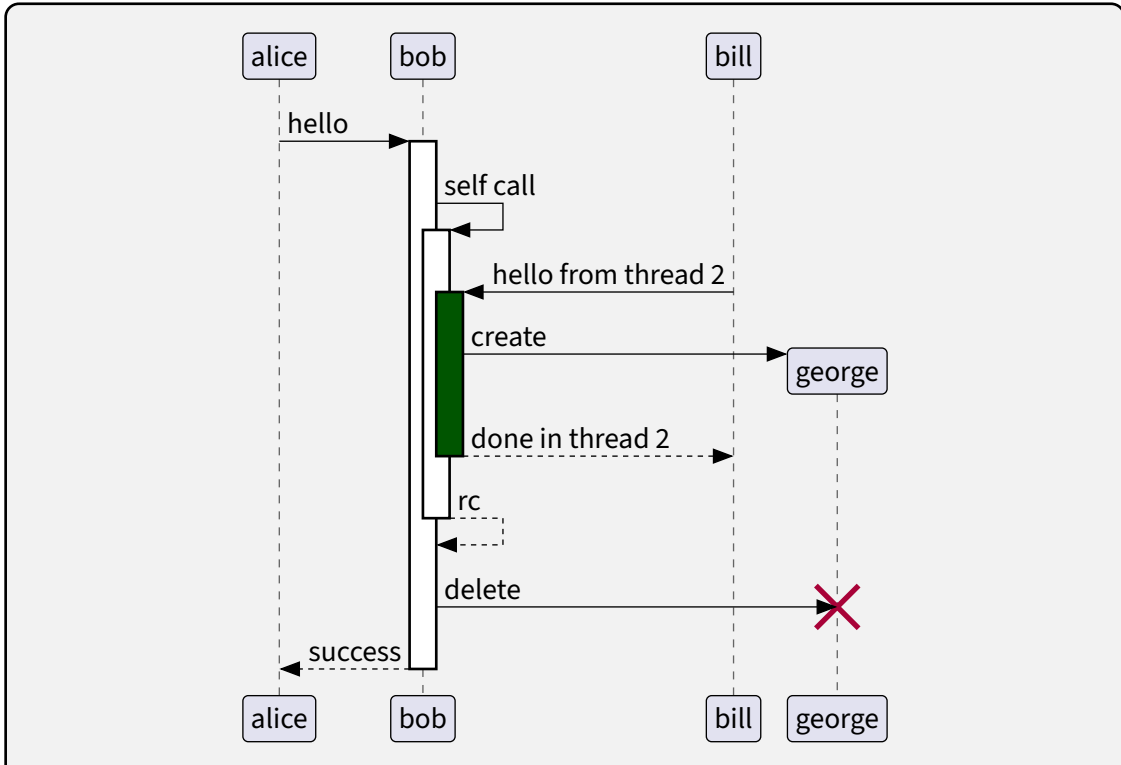
3 Examples

You can find the following examples and more in the [gallery](#) directory

3.1 Some groups and sequences



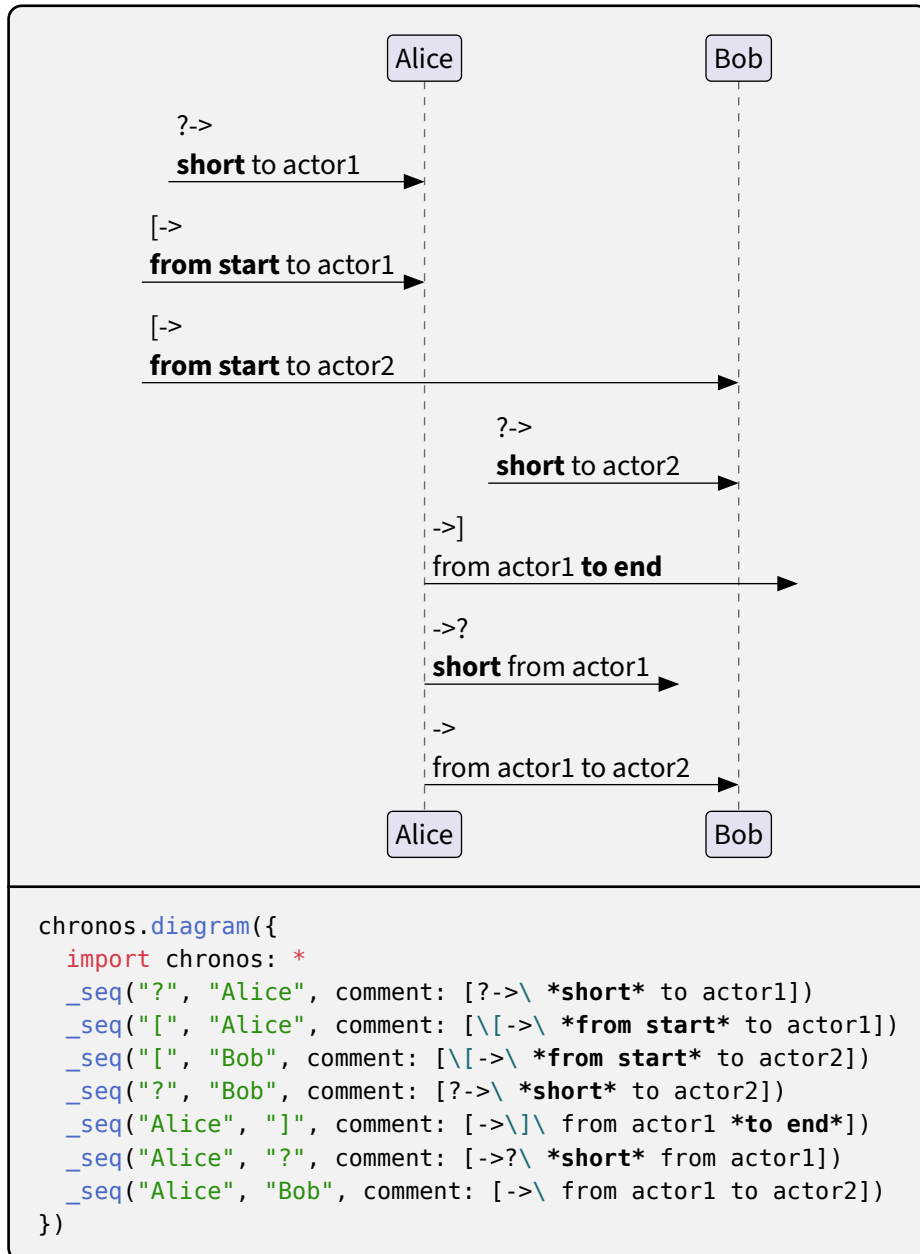
3.2 Lifelines



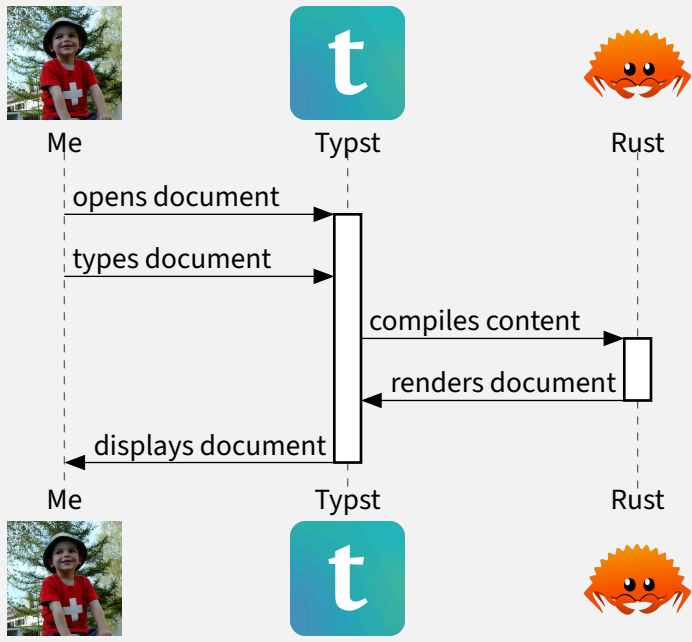
```

chronos.diagram({
  import chronos: *
  _seq("alice", "bob", comment: "hello", enable-dst: true)
  _seq("bob", "bob", comment: "self call", enable-dst: true)
  _seq(
    "bill", "bob",
    comment: "hello from thread 2",
    enable-dst: true,
    lifeline-style: (fill: rgb("#005500"))
  )
  _seq("bob", "george", comment: "create", create-dst: true)
  _seq(
    "bob", "bill",
    comment: "done in thread 2",
    disable-src: true,
    dashed: true
  )
  _seq("bob", "bob", comment: "rc", disable-src: true, dashed: true)
  _seq("bob", "george", comment: "delete", destroy-dst: true)
  _seq("bob", "alice", comment: "success", disable-src: true, dashed: true)
})
  
```

3.3 Found and lost messages



3.4 Custom images



```

let load-img(path) = image(path, width: 1.5cm, height: 1.5cm, fit:"contain")
let TYPST = load-img("../gallery/typst.png")
let FERRIS = load-img("../gallery/ferris.png")
let ME = load-img("../gallery/me.jpg")

chronos.diagram({
  import chronos: *
  _par("me", display-name: "Me", shape: "custom", custom-image: ME)
  _par("typst", display-name: "Typst", shape: "custom", custom-image: TYPST)
  _par("rust", display-name: "Rust", shape: "custom", custom-image: FERRIS)

  _seq("me", "typst", comment: "opens document", enable-dst: true)
  _seq("me", "typst", comment: "types document")
  _seq("typst", "rust", comment: "compiles content", enable-dst: true)
  _seq("rust", "typst", comment: "renders document", disable-src: true)
  _seq("typst", "me", comment: "displays document", disable-src: true)
})

```

4 Reference

4.1 Participants

4.1.1 `_par`

Creates a new participant

Parameters

```
_par(  
  name: str,  
  display-name: auto content,  
  from-start: bool,  
  invisible: bool,  
  shape: str,  
  color: color,  
  custom-image: none image,  
  show-bottom: bool,  
  show-top: bool  
) -> array
```

name str

Unique participant name used as reference in other functions

display-name auto or content

Name to display in the diagram. If set to auto, name is used

Default: auto

from-start bool

If set to true, the participant is created at the top of the diagram. Otherwise, it is created at the first reference

Default: true

invisible bool

If set to true, the participant will not be shown

Default: false

shape str

The shape of the participant. Possible values in [SHAPES](#)

Default: "participant"

color `color`

The participant's color

Default: `rgb("#E2E2F0")`

custom-image `none` or `image`

If shape is 'custom', sets the custom image to display

Default: `none`

show-bottom `bool`

Whether to display the bottom shape

Default: `true`

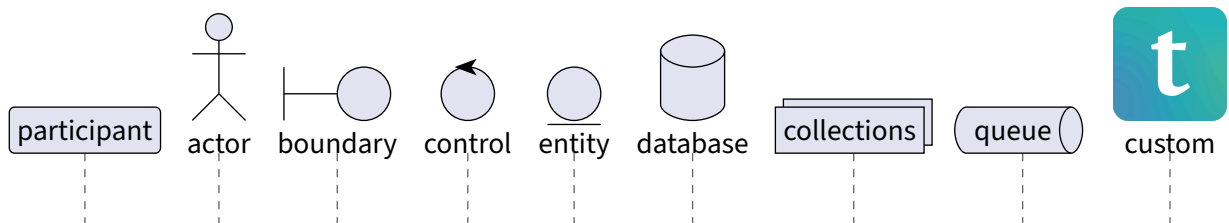
show-top `bool`

Whether to display the top shape

Default: `true`

4.1.2 SHAPES

Possible participant shapes



4.2 Sequences

4.2.1 _evt

Manually adds an event to the given participant

Parameters

```
_evt(  
  participant: str,  
  event: str  
)
```

participant str

The participant concerned by the event

event str

The event type (see [EVENTS](#) for ccepted values)

4.2.2 _seq

Creates a sequence / message between two participants

Parameters

```
_seq(  
  p1: str,  
  p2: str,  
  comment: none content,  
  comment-align: str,  
  dashed: bool,  
  start-tip: str,  
  end-tip: str,  
  color: color,  
  flip: bool,  
  enable-dst: bool,  
  create-dst: bool,  
  disable-dst: bool,  
  destroy-dst: bool,  
  disable-src: bool,  
  destroy-src: bool,  
  lifeline-style: auto dict,  
  slant: none int  
) -> array
```

p1 str

Start participant

p2 `str`

End participant

comment `none` or `content`

Optional comment to display along the arrow

Default: `none`

comment-align `str`

Where to align the comment with respect to the arrow (see `comment-align` for accepted values)

Default: `"left"`

dashed `bool`

Whether the arrow's stroke is dashed or not

Default: `false`

start-tip `str`

Start arrow tip (see `tips` for accepted values)

Default: `" "`

end-tip `str`

End arrow tip (see `tips` for accepted values)

Default: `">"`

color `color`

Arrow's color

Default: `black`

flip `bool`

If true, the arrow is flipped (goes from end to start). This is particularly useful for self calls, to change the side on which the arrow appears

Default: `false`

enable-dst bool

If true, enables the destination lifeline

Default: `false`

create-dst bool

If true, creates the destination lifeline and participant

Default: `false`

disable-dst bool

If true, disables the destination lifeline

Default: `false`

destroy-dst bool

If true, destroys the destination lifeline and participant

Default: `false`

disable-src bool

If true, disables the source lifeline

Default: `false`

destroy-src bool

If true, destroy the source lifeline and participant

Default: `false`

lifeline-style auto or dict

Optional styling options for lifeline rectangles (see CeTZ documentation for more information on all possible values)

Default: `auto`

slant none or int

Optional slant of the arrow

Default: `none`

4.2.3 EVENTS

Accepted values for event argument of `_evt()`

EVENTS = ("create", "destroy", "enable", "disable")

4.2.4 tips

Accepted values for start-tip and end-tip arguments of `_seq()`

	<pre> let _seq = _seq.with(comment-align: "center") _par("a", display-name: "Alice") _par("b", display-name: "Bob") _seq("a", "b", comment: "Various tips", end-tip: "") _seq("a", "b", end-tip: ">", comment: `->`) _seq("a", "b", end-tip: ">>", comment: `->>`) _seq("a", "b", end-tip: "\\", comment: `-\`) _seq("a", "b", end-tip: "\\\", comment: `-\`) _seq("a", "b", end-tip: "/", comment: `-/`) _seq("a", "b", end-tip: "//", comment: `-//`) _seq("a", "b", end-tip: "x", comment: `->x`) _seq("a", "b", start-tip: "x", comment: `x->`) _seq("a", "b", start-tip: "o", comment: `o->`) _seq("a", "b", end-tip: ("o", ">"), comment: `->o`) _seq("a", "b", start-tip: "o", end-tip: ("o", ">"), comment: `o->o`) _seq("a", "b", start-tip: ">", end-tip: ">", comment: `<->`) _seq("a", "b", start-tip: ("o", ">"), end-tip: ("o", ">"), comment: `o<->o`) _seq("a", "b", start-tip: "x", end-tip: "x", comment: `x<->x`) _seq("a", "b", end-tip: ("o", ">>"), comment: `->>o`) _seq("a", "b", end-tip: ("o", "\\\"), comment: `-\o`) _seq("a", "b", end-tip: ("o", "\\\"), comment: `-\o`) _seq("a", "b", end-tip: ("o", "/\"), comment: `-/o`) _seq("a", "b", end-tip: ("o", "//\"), comment: `-//o`) _seq("a", "b", start-tip: "x", end-tip: ("o", ">"), comment: `x->o`) </pre>
--	--

4.2.5 comment-align

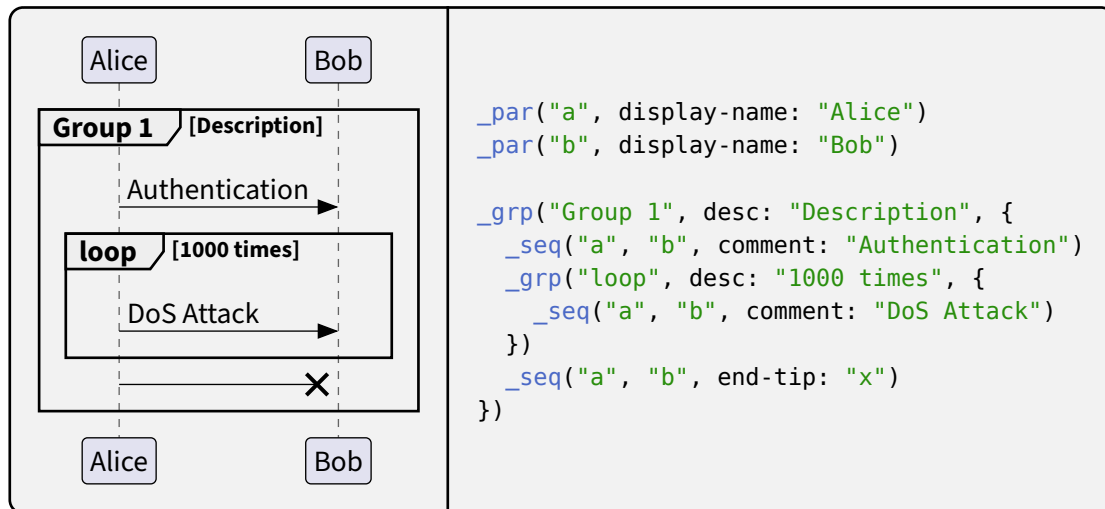
Accepted values for comment-align argument of `_seq()`

	<pre> _par("p1", display-name: "Start participant") _par("p2", display-name: "End participant") let alignments = ("start", "end", "left", "right", "center") for a in alignments { _seq("p2", "p1", comment: raw(a), comment-align: a) } </pre>
--	---

4.3 Groups

4.3.1 `_grp`

Creates a group of sequences



Parameters

```
_grp(
  name: content,
  desc: none | content,
  type: str,
  elmts: array
)
```

name `content`

The group's name

desc `none` or `content`

Optional description

Default: `none`

type `str`

The groups's type (unused for the moment)

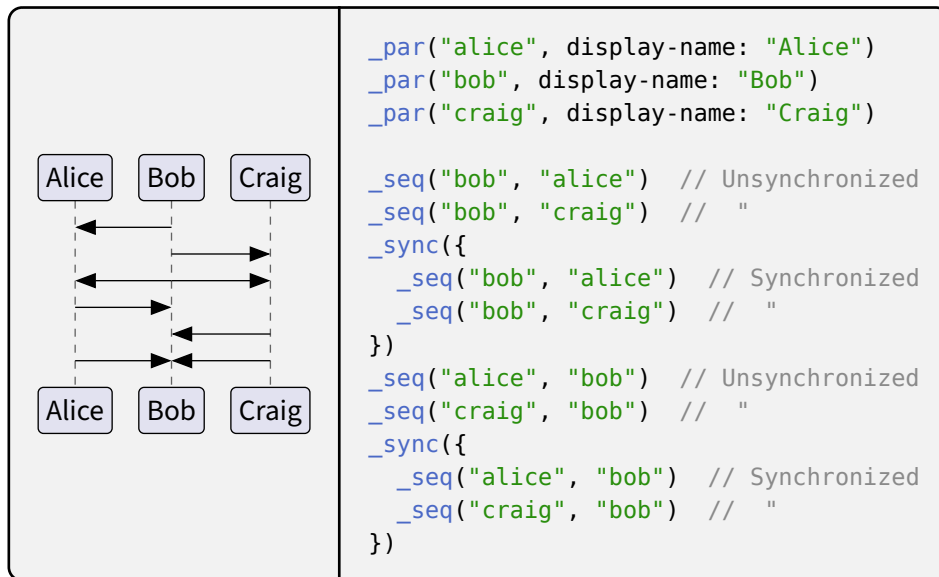
Default: `"default"`

elmts `array`

Elements inside the group (can be sequences, other groups, notes, etc.)

4.3.2 `_sync`

Synchronizes multiple sequences



Parameters

`_sync`(`elmts`: array)

elmts array

Synchronized elements (generally sequences or notes)

4.4 Gaps and separators

4.4.1 `_gap`

Creates a gap before the next element

Parameters

`_gap(size: int)`

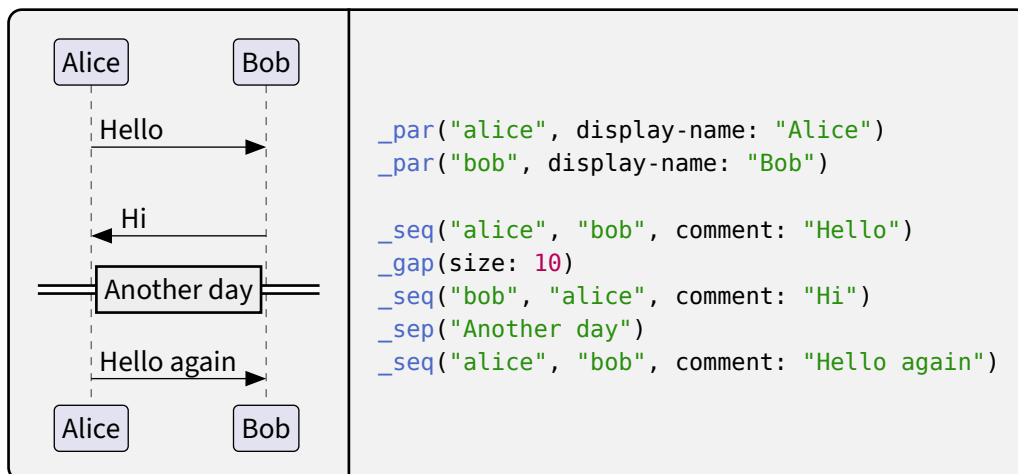
size `int`

Size of the gap

Default: `20`

4.4.2 `_sep`

Creates a separator before the next element



Parameters

`_sep(name: content)`

name `content`

Name to display in the middle of the separator

4.5 Notes

4.5.1 `_note`

Creates a note

Parameters

```
_note(  
    side: str,  
    content: content,  
    pos: none | str | array,  
    color: color,  
    shape: str,  
    aligned: bool  
)
```

side str

The side on which to place the note (see [SIDES](#) for accepted values)

content content

The note's content

pos none or str or array

Optional participant(s) on which to draw next to / over. If side is "left" or "right", sets next to which participant the note is placed. If side is "over", sets over which participant(s) it is placed

Default: none

color color

The note's color

Default: `rgb("#FEFFDD")`

shape str

The note's shape (see [SHAPES](#) for accepted values)

Default: "default"

aligned bool

True if the note is aligned with another note, in which case side must be "over", false otherwise

Default: false

4.5.2 SHAPES

Accepted values for shape argument of `_note()`



4.5.3 SIDES

Accepted values for side argument of `_note()`

