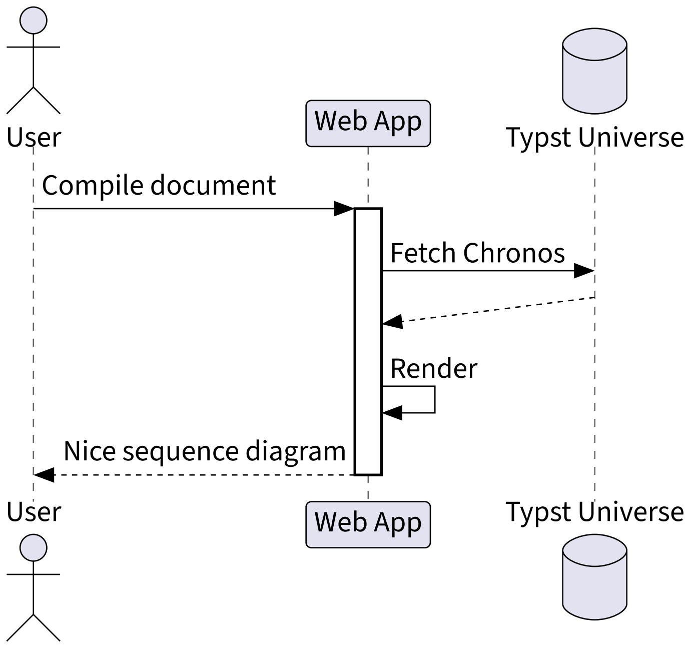
Chronos

v0.3.0



Contents

1 Introduction . . . 3
2 Usage . . . 3
3 Examples . . . 3
  3.1 Some groups and sequences . . . 3
  3.2 Lifelines . . . 4
  3.3 Found and lost messages . . . 5
  3.4 Custom images . . . 6
4 Reference . . . 7
  4.1 Participants . . . 7
    4.1.1 _par . . . 7
    4.1.2 _col . . . 8
    4.1.3 SHAPES . . . 9
  4.2 Sequences . . . 10
    4.2.1 _evt . . . 10
    4.2.2 _seq . . . 10
    4.2.3 _ret . . . 13
    4.2.4 comment-align . . . 14
    4.2.5 EVENTS . . . 14
    4.2.6 tips . . . 14
  4.3 Groups . . . 16
    4.3.1 _grp . . . 16
    4.3.2 _alt . . . 17
    4.3.3 _loop . . . 18
    4.3.4 _sync . . . 19
    4.3.5 _opt . . . 19
    4.3.6 _break . . . 20
  4.4 Gaps and separators . . . 21
    4.4.1 _sep . . . 21
    4.4.2 _delay . . . 21
    4.4.3 _gap . . . 22
  4.5 Notes . . . 24
    4.5.1 _note . . . 24
    4.5.2 SHAPES . . . 25
    4.5.3 SIDES . . . 25

1 Introduction

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

2 Usage

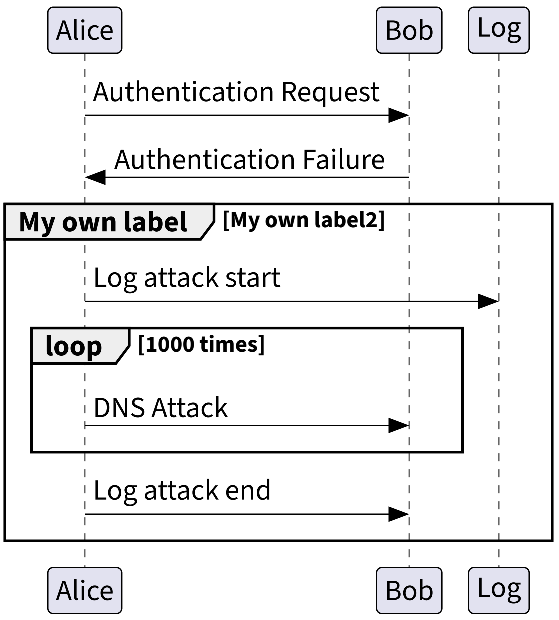
Simply import chronos and call the `diagram` function:

```typst
#import "@preview/chronos:0.3.0"
#chronos.diagram({
  import chronos: *
  ...
})
```

3 Examples

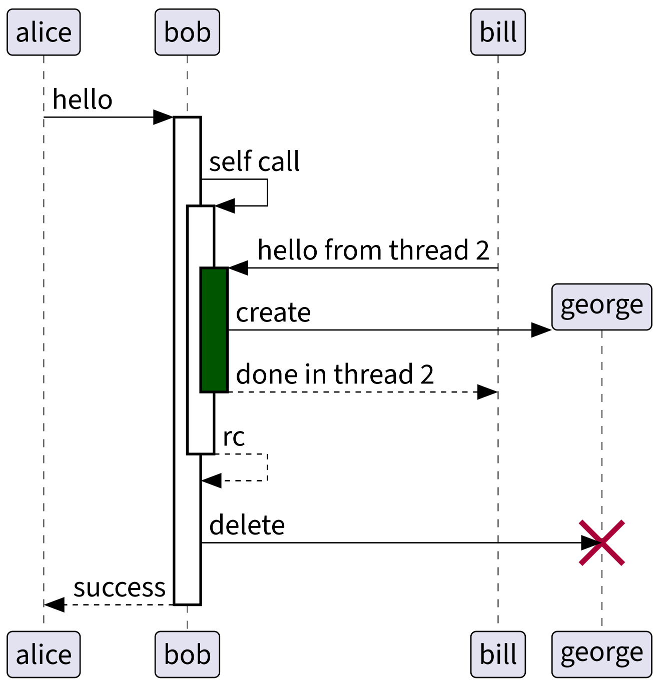
You can find the following examples and more in the gallery directory

3.1 Some groups and sequences



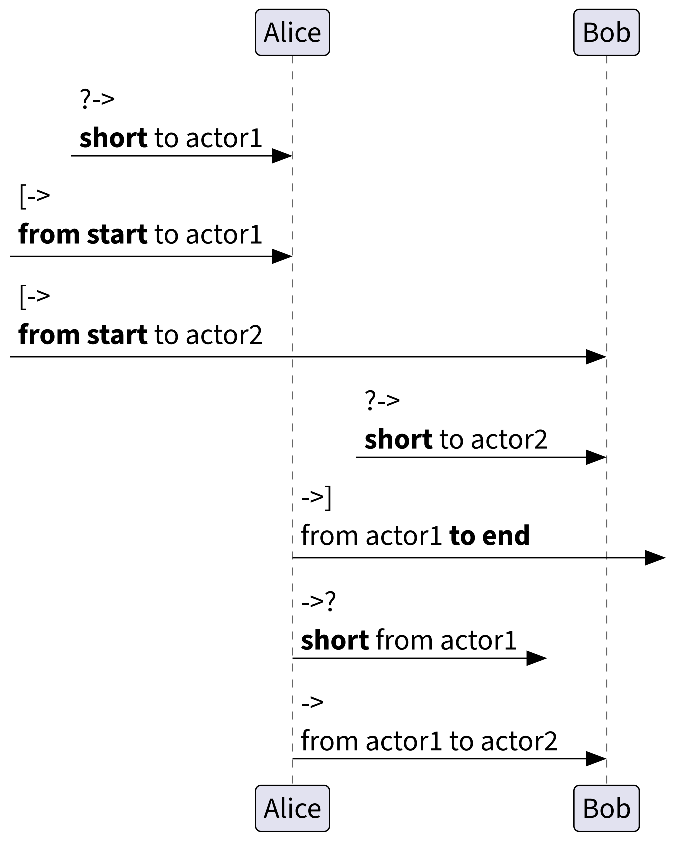
```typst
chronos.diagram({
  import chronos: *
  _seq("Alice", "Bob", comment: "Authentication Request")
  _seq("Bob", "Alice", comment: "Authentication Failure")

  _grp("My own label", desc: "My own label2", {
    _seq("Alice", "Log", comment: "Log attack start")
    _grp("loop", desc: "1000 times", {
      _seq("Alice", "Bob", comment: "DNS Attack")
    })
    _seq("Alice", "Bob", comment: "Log attack end")
  })
})
```

3.2 Lifelines



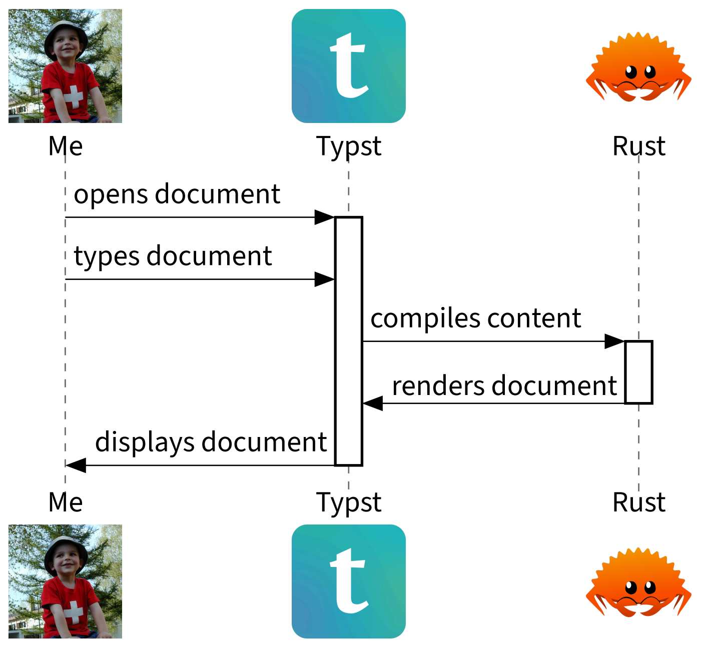
Typst
```typst
chronos.diagram({
  import chronos: *
  _seq("alice", "bob", comment: "hello", enable-dst: true)
  _seq("bob", "bob", comment: "self call", enable-dst: true)
  _seq(
    "bill", "bob",
    comment: "hello from thread 2",
    enable-dst: true,
    lifeline-style: (fill: rgb("#005500"))
  )
  _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



Typst
```
chronos.diagram({
  import chronos: *
  _seq("?", "Alice", comment: [?->\ *short* to actor1])
  _seq("[", "Alice", comment: [\[->\ *from start* to actor1])
  _seq("[", "Bob", comment: [\[->\ *from start* to actor2])
  _seq("?", "Bob", comment: [?->\ *short* to actor2])
  _seq("Alice", "]", comment: [->\]\ from actor1 *to end*])
  _seq("Alice", "?", comment: [->?\ *short* from actor1])
  _seq("Alice", "Bob", comment: [->\ from actor1 to actor2])
})
```

3.4 Custom images



```typst
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,
  line-stroke: stroke,
  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")`

line-stroke `stroke`

The participant’s line style (defaults to a light gray dashed line)

Default:
```
(
    dash: "dashed",
    paint: gray.darken(40%),
    thickness: .5pt
  )
```

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 _col

Sets some options for columns between participants

Parameters `p1` and `p2` MUST be consecutive participants (also counting found/lost messages), but they do not need to be in the left to right order

Parameters

```
_col(
  p1: str,
  p2: str,
  width: auto int float length,
  margin: int float length,
  min-width: int float length,
  max-width: int float length none
)
```

p1 `str`

The first neighbouring participant

p2 `str`

The second neighbouring participant

width `auto` or `int` or `float` or `length`

Optional fixed width of the column
If the column’s content (e.g. sequence comments) is larger, it will overflow

Default: `auto`

margin `int` or `float` or `length`

Additional margin to add to the column
This margin is not included in `width` and `min-width`, but rather added separately

Default: `0`

min-width `int` or `float` or `length`

Minimum width of the column
If set to a larger value than `width`, the latter will be overriden

Default: `0`

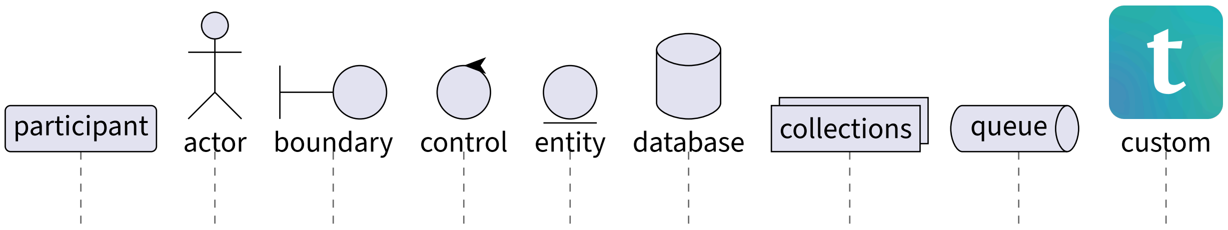
max-width `int` or `float` or `length` or `none`

Maximum width of the column
If set to a lower value than `width`, the latter will be overriden
If set to `none`, no restriction is applied

Default: `none`

4.1.3 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,
  lifeline-style: auto dict
)
```

participant `str`

The participant concerned by the event

event `str`

The event type (see EVENTS for ccepted values)

lifeline-style `auto` or `dict`

See `_seq()`

Default: `auto`

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,
  outer-lifeline-connect: bool
) -> 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`

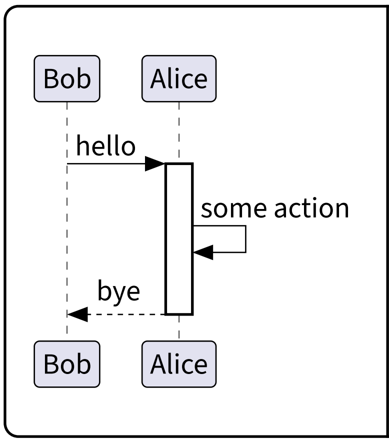
outer-lifeline-connect `bool`

If true, enables legacy anchoring, making sequences connect to the leftmost lifeline when arriving from the left side. If false, all connections are made with the latest/rightmost lifeline

Default: `false`

4.2.3 _ret

Creates a return sequence



```typst
_seq(
  "Bob", "Alice",
  comment: [hello],
  enable-dst: true
)
_seq(
  "Alice", "Alice",
  comment: [some action]
)
_ret(comment: [bye])
```

Parameters

`_ret(comment: none content)`

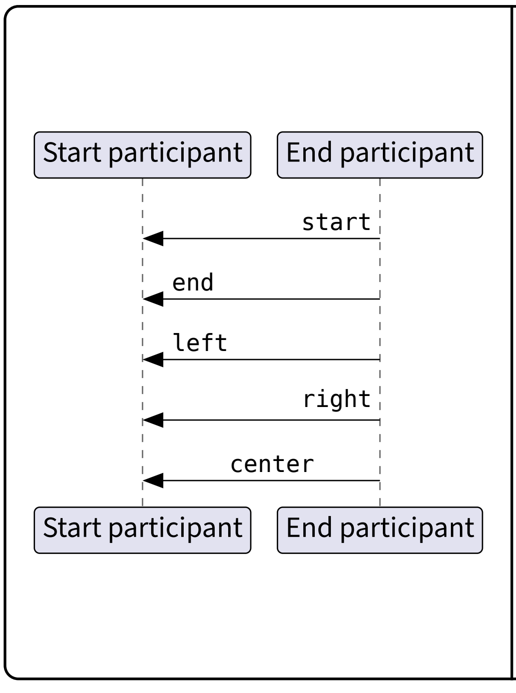
comment `none` or `content`

Optional comment to display along the arrow

Default: `none`

4.2.4 comment-align

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



```
_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
  )
}
```

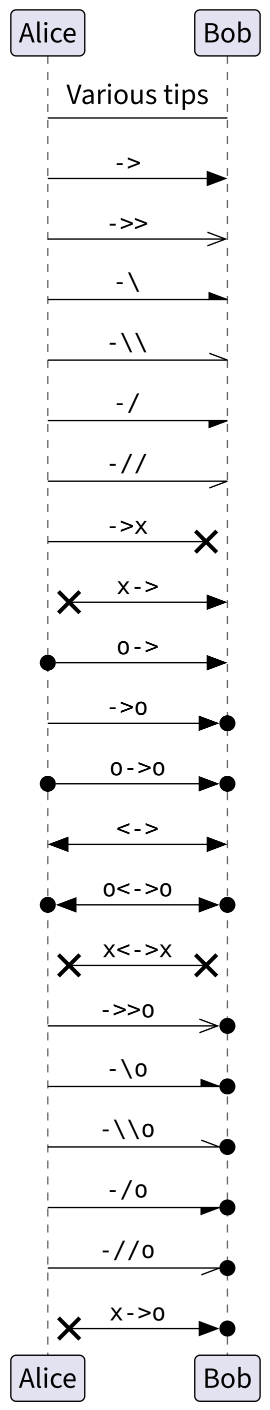
4.2.5 EVENTS

Accepted values for `event` argument of `_evt()`

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

4.2.6 tips

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

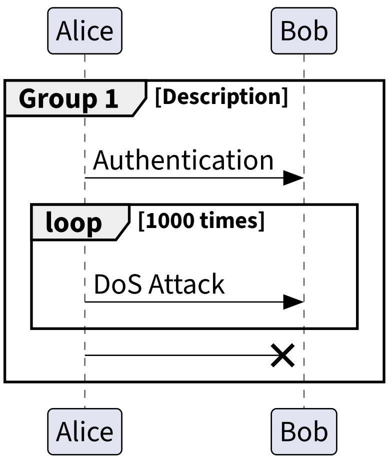


```typst
let _seq = _seq.with(comment-align: "center")
_par("a", display-name: "Alice")
_par("b", display-name: "Bob")

_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`)
```

4.3 Groups

4.3.1 _grp

Creates a group of sequences



```typst
_par("a", display-name: "Alice")
_par("b", display-name: "Bob")

_grp("Group 1", desc: "Description", {
  _seq("a", "b", comment: "Authentication")
  _grp("loop", desc: "1000 times", {
    _seq("a", "b", comment: "DoS Attack")
  })
  _seq("a", "b", end-tip: "x")
})
```

Parameters

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

name `content`

The group's name

elmts `array`

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

desc `none` or `content`

Optional description

Default: none

type `str`

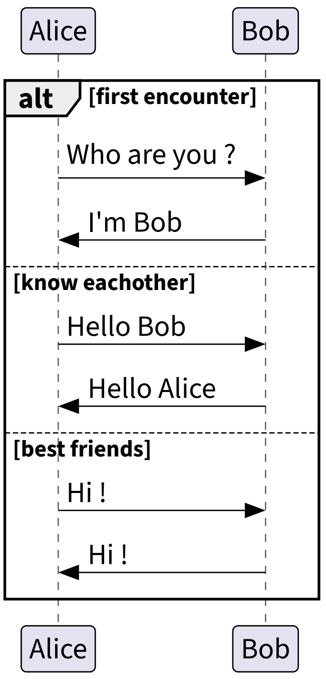
The groups's type (should only be set through other functions like `_alt()` or `_loop()`)

Default: `"default"`

4.3.2 _alt

Creates an alt-else group of sequences

It contains at least one section but can have as many as needed



```typst
_par("a", display-name: "Alice")
_par("b", display-name: "Bob")

_alt(
  "first encounter", {
    _seq("a", "b", comment: "Who are you ?")
    _seq("b", "a", comment: "I'm Bob")
  },

  "know eachother", {
    _seq("a", "b", comment: "Hello Bob")
    _seq("b", "a", comment: "Hello Alice")
  },

  "best friends", {
    _seq("a", "b", comment: "Hi !")
    _seq("b", "a", comment: "Hi !")
  }
)
```

Parameters

```
_alt(
  desc: content,
  elmts: array,
  ..args: content array
)
```

desc `content`

The alt's label

elmts `array`

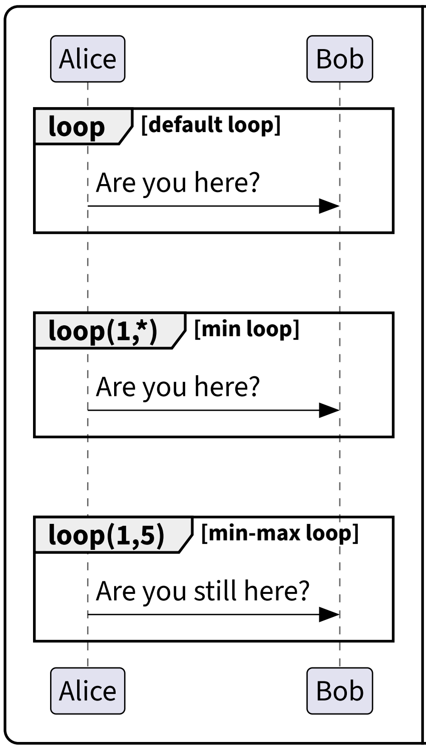
Elements inside the alt's first section

..args `content` or `array`

Complementary "else" sections.
You can add as many else sections as you need by passing a content (else section label) followed by an array of elements (see example)

4.3.3 _loop

Creates a looped group of sequences



```typst
_par("a", display-name: "Alice")
_par("b", display-name: "Bob")

_loop("default loop", {
  _seq("a", "b", comment: "Are you here?")
})
_gap()
_loop("min loop", min: 1, {
  _seq("a", "b", comment: "Are you here?")
})
_gap()
_loop("min-max loop", min: 1, max: 5, {
  _seq("a", "b", comment: "Are you still here?")
})
```

Parameters

```
_loop(
  desc: content,
  elmts: array,
  min: none number,
  max: auto number
)
```

desc `content`

Loop description

elmts `array`

Elements inside the group

min `none` or `number`

Optional lower bound of the loop

Default: `none`

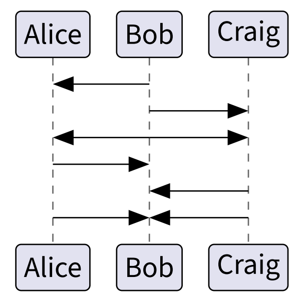
max `auto` or `number`

Upper bound of the loop. If left as `auto` and `min` is set, it will be infinity (`'*'`)

Default: `auto`

4.3.4 _sync

Synchronizes multiple sequences
All elements inside a synchronized group will start at the same time



```typst
_par("alice", display-name: "Alice")
_par("bob", display-name: "Bob")
_par("craig", display-name: "Craig")

_seq("bob", "alice")  // Unsynchronized
_seq("bob", "craig")  //  "
_sync({
  _seq("bob", "alice")  // Synchronized
  _seq("bob", "craig")  //  "
})
_seq("alice", "bob")  // Unsynchronized
_seq("craig", "bob")  //  "
_sync({
  _seq("alice", "bob")  // Synchronized
  _seq("craig", "bob")  //  "
})
```

Parameters

`_sync(elmts: array)`

elmts `array`

Synchronized elements (generally sequences or notes)

4.3.5 _opt

Creates an optional group
This is a simple wrapper around `_grp()`

Parameters

```
_opt(
  desc: content,
  elmts: array
)
```

desc `content`

Group description

elmts `array`

Elements inside the group

4.3.6 _break

Creates a break group
This is a simple wrapper around `_grp()`

Parameters

```
_break(
  desc: content,
  elmts: array
)
```

desc `content`

Group description

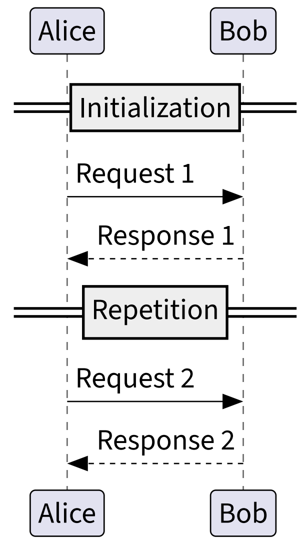
elmts `array`

Elements inside the group

4.4 Gaps and separators

4.4.1 _sep

Creates a separator before the next element



Typst
```
_par("a", display-name: "Alice")
_par("b", display-name: "Bob")

_sep[Initialization]
_seq("a", "b", comment: [Request 1])
_seq(
  "b", "a",
  comment: [Response 1],
  dashed: true
)

_sep[Repetition]
_seq("a", "b", comment: [Request 2])
_seq(
  "b", "a",
  comment: [Response 2],
  dashed: true
)
```

Parameters

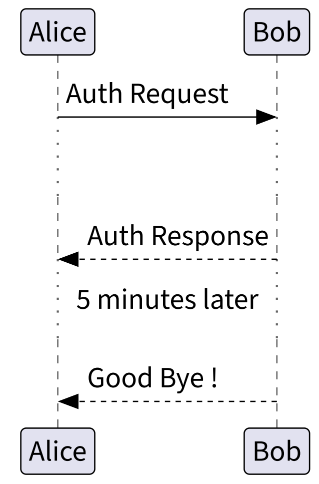
`_sep(name: content)`

name `content`

Name to display in the middle of the separator

4.4.2 _delay

Creates a delay before the next element



```typst
_par("a", display-name: "Alice")
_par("b", display-name: "Bob")

_seq("a", "b", comment: [Auth Request])
_delay()
_seq(
  "b", "a",
  comment: [Auth Response],
  dashed: true
)
_delay(name: [5 minutes later])
_seq(
  "b", "a",
  comment: [Good Bye !],
  dashed: true
)
```

Parameters

```
_delay(
  name: content none,
  size: int
)
```

name `content` or `none`

Name to display in the middle of the delay area

Default: `none`

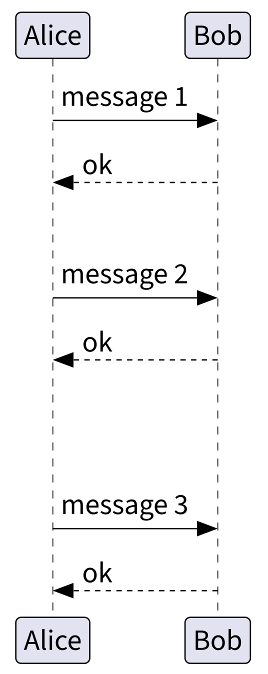
size `int`

Size of the delay

Default: `30`

4.4.3 _gap

Creates a gap before the next element



```typst
_par("a", display-name: "Alice")
_par("b", display-name: "Bob")

_seq("a", "b", comment: [message 1])
_seq("b", "a", comment: [ok], dashed: true)
_gap()
_seq("a", "b", comment: [message 2])
_seq("b", "a", comment: [ok], dashed: true)
_gap(size: 40)
_seq("a", "b", comment: [message 3])
_seq("b", "a", comment: [ok], dashed: true)
```

Parameters

`_gap(size: int)`

size `int`

Size of the gap

Default: `20`

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,
  allow-overlap: 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`

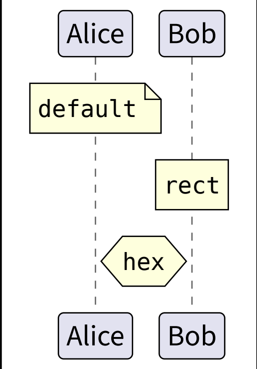
allow-overlap `bool`

If set to `false`, the note will try to reserve space in the column to avoid overlapping with neighboring participants. If set to `true`, the not will overlap other participants

Default: `true`

4.5.2 SHAPES

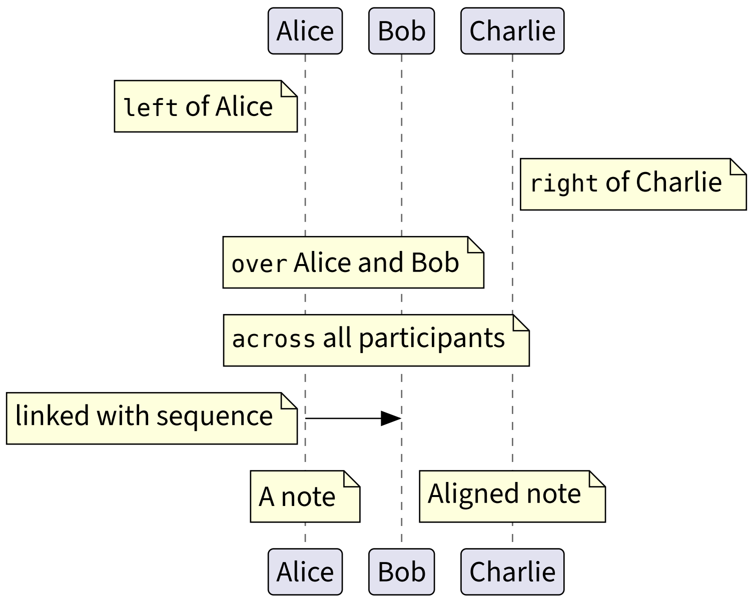
Accepted values for `shape` argument of `_note()`



```typst
_par("alice", display-name: "Alice")
_par("bob", display-name: "Bob")
_note("over", `default`, pos: "alice")
_note("over", `rect`, pos: "bob", shape: "rect")
_note("over", `hex`, pos: ("alice", "bob"), shape: "hex")
```

4.5.3 SIDES

Accepted values for `side` argument of `_note()`



```typst
_par("alice", display-name: "Alice")
_par("bob", display-name: "Bob")
_par("charlie", display-name: "Charlie")
_note("left", [`left` of Alice], pos: "alice")
_note("right", [`right` of Charlie], pos: "charlie")
_note("over", [`over` Alice and Bob], pos: ("alice", "bob"))
_note("across", [`across` all participants])
_seq("alice", "bob")
_note("left", [linked with sequence])
_note("over", [A note], pos: "alice")
_note("over", [Aligned note], pos: "charlie", aligned: true)
```