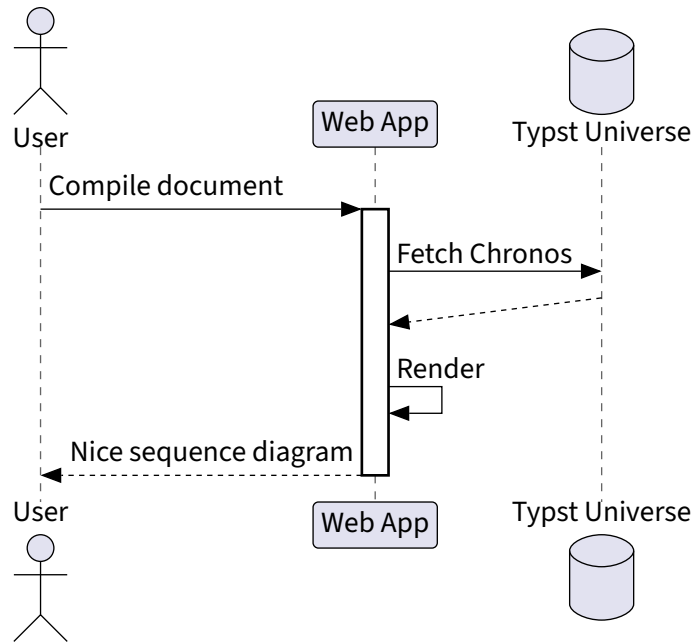


Chronos

v0.2.1



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	13
4.2.5	EVENTS	14
4.2.6	tips	14
4.3	Groups	15
4.3.1	_grp	15
4.3.2	_alt	16
4.3.3	_loop	17
4.3.4	_sync	18
4.3.5	_opt	18
4.3.6	_break	19
4.4	Gaps and separators	20
4.4.1	_sep	20
4.4.2	_delay	20
4.4.3	_gap	21
4.5	Notes	23
4.5.1	_note	23
4.5.2	SHAPES	24
4.5.3	SIDES	24

1 Introduction

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

2 Usage

Simply import `chronos` and call the diagram function:

```

1 #import "@preview/chronos:0.2.1"
2 #chronos.diagram({
3   import chronos: *
4   ...
5 })

```

3 Examples

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

3.1 Some groups and sequences

```

sequenceDiagram
    participant Alice
    participant Bob
    participant Log

    Alice->>Bob: Authentication Request
    Bob-->>Alice: Authentication Failure

    group "My own label" [My own label2]
        Alice->>Log: Log attack start
        loop "loop" [1000 times]
            Alice->>Bob: DNS Attack
        end
        Alice->>Log: Log attack end
    end

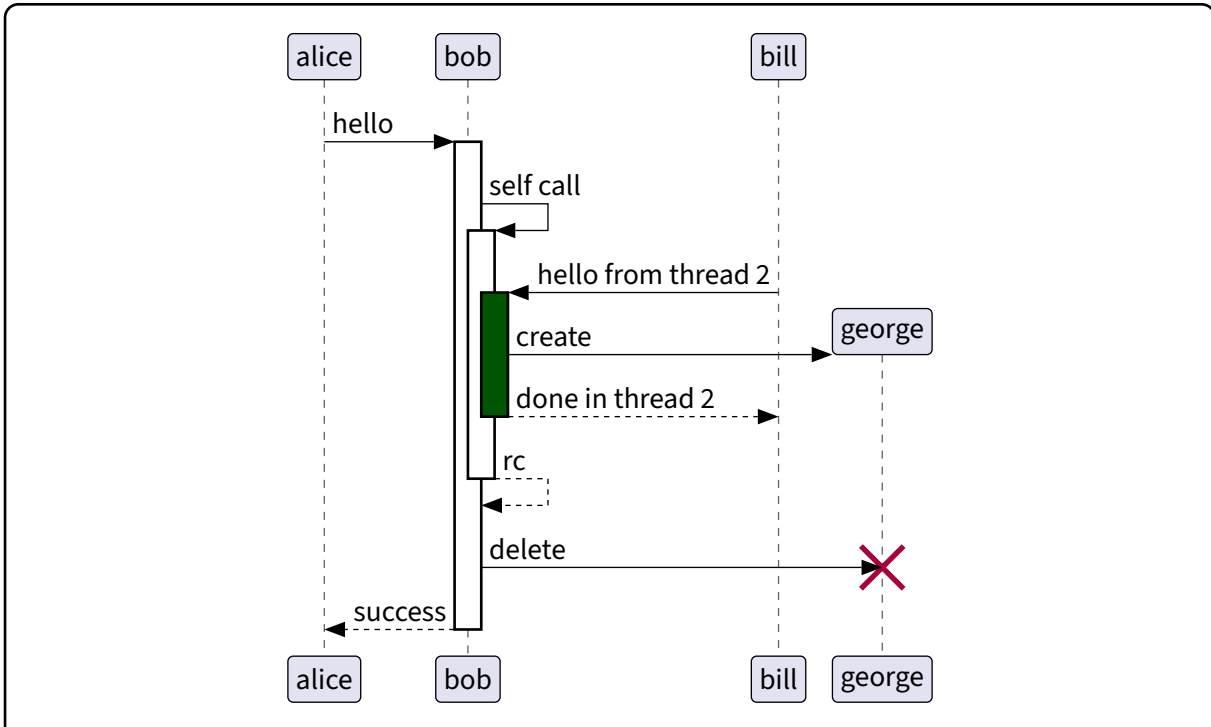
```

```

1 chronos.diagram({
2   import chronos: *
3   _seq("Alice", "Bob", comment: "Authentication Request")
4   _seq("Bob", "Alice", comment: "Authentication Failure")
5
6   _grp("My own label", desc: "My own label2", {
7     _seq("Alice", "Log", comment: "Log attack start")
8     _grp("loop", desc: "1000 times", {
9       _seq("Alice", "Bob", comment: "DNS Attack")
10    })
11    _seq("Alice", "Bob", comment: "Log attack end")
12  })
13 })

```

3.2 Lifelines

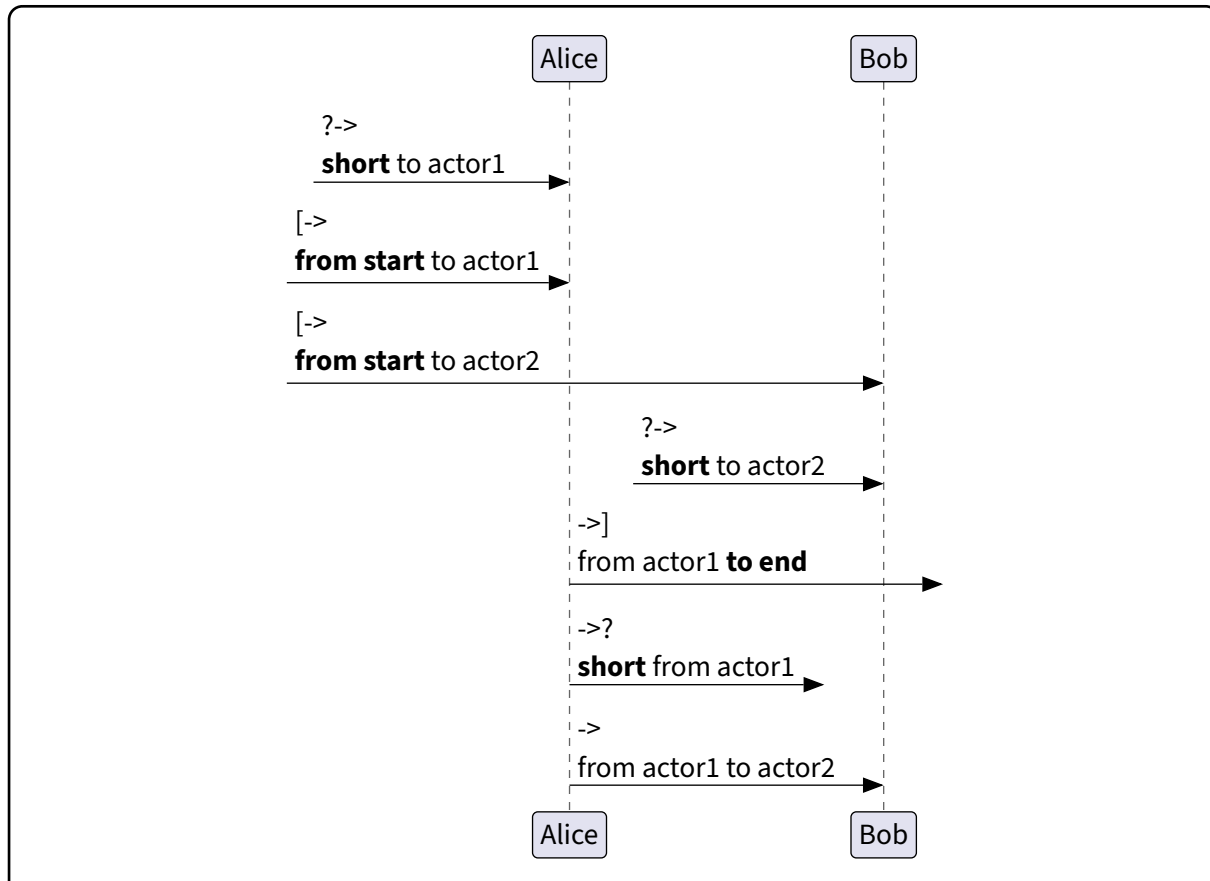


```

1  chronos.diagram({
2    import chronos: *
3    _seq("alice", "bob", comment: "hello", enable-dst: true)
4    _seq("bob", "bob", comment: "self call", enable-dst: true)
5    _seq(
6      "bill", "bob",
7      comment: "hello from thread 2",
8      enable-dst: true,
9      lifeline-style: (fill: rgb("#005500"))
10   )
11   _seq("bob", "george", comment: "create", create-dst: true)
12   _seq(
13     "bob", "bill",
14     comment: "done in thread 2",
15     disable-src: true,
16     dashed: true
17   )
18   _seq("bob", "bob", comment: "rc", disable-src: true, dashed: true)
19   _seq("bob", "george", comment: "delete", destroy-dst: true)
20   _seq("bob", "alice", comment: "success", disable-src: true, dashed: true)
21 })

```

3.3 Found and lost messages



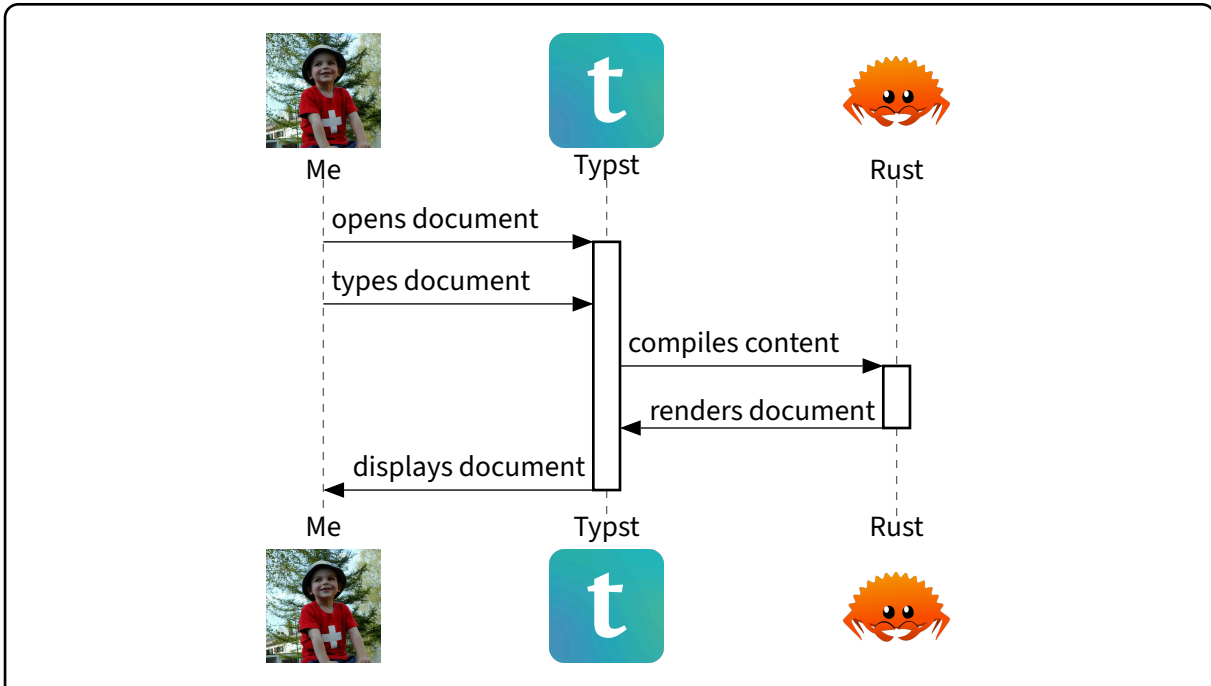
```

1 chronos.diagram({
2   import chronos: *
3   _seq("?", "Alice", comment: [?->\ *short* to actor1])
4   _seq("[", "Alice", comment: [\[->\ *from start* to actor1])
5   _seq("[", "Bob", comment: [\[->\ *from start* to actor2])
6   _seq("?", "Bob", comment: [?->\ *short* to actor2])
7   _seq("Alice", "]", comment: [->\]\ from actor1 *to end*])
8   _seq("Alice", "?", comment: [->?\ *short* from actor1])
9   _seq("Alice", "Bob", comment: [->\ from actor1 to actor2])
10  })

```

Typst

3.4 Custom images



```

1  let load-img(path) = image(
2    path,
3    width: 1.5cm, height: 1.5cm,
4    fit:"contain"
5  )
6  let TYPST = load-img("../gallery/typst.png")
7  let FERRIS = load-img("../gallery/ferris.png")
8  let ME = load-img("../gallery/me.jpg")
9
10 chronos.diagram({
11   import chronos: *
12   _par("me", display-name: "Me", shape: "custom", custom-image: ME)
13   _par("typst", display-name: "Typst", shape: "custom", custom-image: TYPST)
14   _par("rust", display-name: "Rust", shape: "custom", custom-image: FERRIS)
15
16   _seq("me", "typst", comment: "opens document", enable-dst: true)
17   _seq("me", "typst", comment: "types document")
18   _seq("typst", "rust", comment: "compiles content", enable-dst: true)
19   _seq("rust", "typst", comment: "renders document", disable-src: true)
20   _seq("typst", "me", comment: "displays document", disable-src: true)
21 })

```

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 `_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 overridden

Default: `0`

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

Maximum width of the column

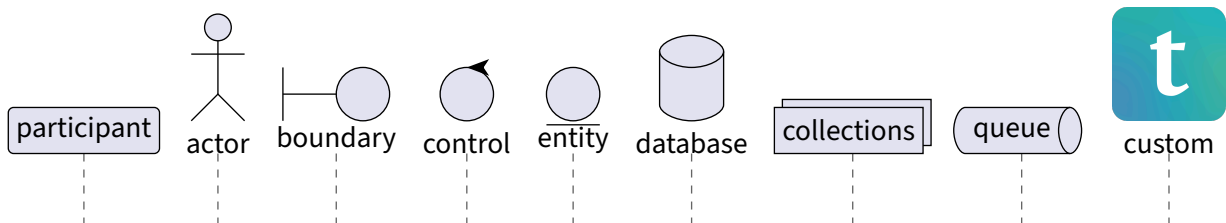
If set to a lower value than `width`, the latter will be overridden

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  
)
```

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 _ret

Creates a return sequence

<pre> sequenceDiagram participant Bob participant Alice Bob->>Alice: hello activate Alice Alice->>Alice: some action deactivate Alice Alice-->>Bob: bye </pre>	<pre> 1 _seq(2 "Bob", "Alice", 3 comment: [hello], 4 enable-dst: true 5) 6 _seq(7 "Alice", "Alice", 8 comment: [some action] 9) 10 _ret(comment: [bye]) </pre>
--	---

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()`

	<pre> 1 _par("p1", 2 display-name: "Start participant") 3 _par("p2", 4 display-name: "End participant") 5 let alignments = (6 "start", "end", 7 "left", "right", 8 "center" 9) 10 for a in alignments { 11 _seq(12 "p2", "p1", 13 comment: raw(a), 14 comment-align: a 15) 16 } </pre>
--	--

4.2.5 EVENTS

Accepted values for event argument of `_evt()`

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

4.2.6 tips

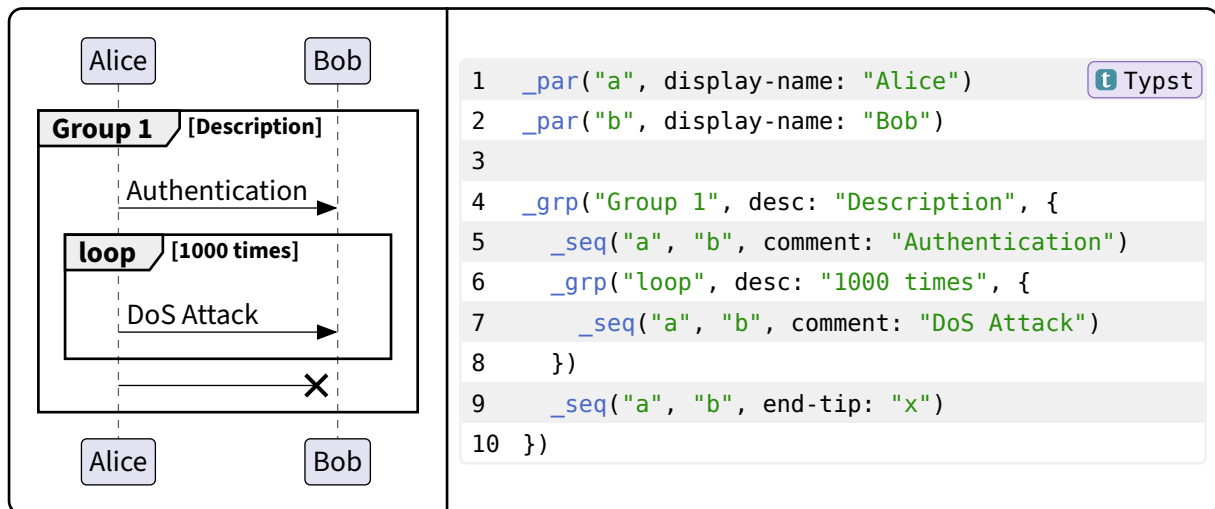
Accepted values for start-tip and end-tip arguments of `_seq()`

	<pre> 1 let _seq = _seq.with(comment-align: "center") 2 _par("a", display-name: "Alice") 3 _par("b", display-name: "Bob") 4 5 _seq("a", "b", comment: "Various tips", end-tip: "") 6 _seq("a", "b", end-tip: ">", comment: `->`) 7 _seq("a", "b", end-tip: ">>", comment: `->>`) 8 _seq("a", "b", end-tip: "\\", comment: `-\`) 9 _seq("a", "b", end-tip: "\\\", comment: `-\`) 10 _seq("a", "b", end-tip: "/", comment: `-/`) 11 _seq("a", "b", end-tip: "//", comment: `-//`) 12 _seq("a", "b", end-tip: "x", comment: `->x`) 13 _seq("a", "b", start-tip: "x", comment: `x->`) 14 _seq("a", "b", start-tip: "o", comment: `o->`) 15 _seq("a", "b", end-tip: ("o", ">"), comment: `->o`) 16 _seq("a", "b", start-tip: "o", 17 end-tip: ("o", ">"), comment: `o->o`) 18 _seq("a", "b", start-tip: ">", 19 end-tip: ">", comment: `<->`) 20 _seq("a", "b", start-tip: ("o", ">"), 21 end-tip: ("o", ">"), comment: `o<->o`) 22 _seq("a", "b", start-tip: "x", 23 end-tip: "x", comment: `x<->x`) 24 _seq("a", "b", end-tip: ("o", ">>"), comment: `->>o`) 25 _seq("a", "b", end-tip: ("o", "\\\"), comment: `-\o`) 26 _seq("a", "b", end-tip: ("o", "\\\"), comment: `-\o`) 27 _seq("a", "b", end-tip: ("o", "/\"), comment: `-/o`) 28 _seq("a", "b", end-tip: ("o", "//\"), comment: `-//o`) 29 _seq("a", "b", start-tip: "x", 30 end-tip: ("o", ">"), comment: `x->o`) </pre>
--	--

4.3 Groups

4.3.1 _grp

Creates a group of sequences



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

```

sequenceDiagram
    participant Alice
    participant Bob
    alt [first encounter]
        Alice->>Bob: Who are you ?
        Bob-->>Alice: I'm Bob
    and [know eachother]
        Alice->>Bob: Hello Bob
        Bob-->>Alice: Hello Alice
    and [best friends]
        Alice->>Bob: Hi !
        Bob-->>Alice: Hi !
    end
    
```

```

1  _par("a", display-name: "Alice")
2  _par("b", display-name: "Bob")
3
4  _alt(
5    "first encounter", {
6      _seq("a", "b", comment: "Who are you ?")
7      _seq("b", "a", comment: "I'm Bob")
8    },
9
10 "know eachother", {
11  _seq("a", "b", comment: "Hello Bob")
12  _seq("b", "a", comment: "Hello Alice")
13 },
14
15 "best friends", {
16  _seq("a", "b", comment: "Hi !")
17  _seq("b", "a", comment: "Hi !")
18 }
19 )
    
```

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

The diagram illustrates three different loop behaviors between Alice and Bob:

- loop [default loop]:** A single message "Are you here?" is sent from Alice to Bob.
- loop(1,*) [min loop]:** A single message "Are you here?" is sent from Alice to Bob, indicating a minimum of 1 iteration.
- loop(1,5) [min-max loop]:** A single message "Are you still here?" is sent from Alice to Bob, indicating a minimum of 1 and a maximum of 5 iterations.

```

1  _par("a", display-name: "Alice")
2  _par("b", display-name: "Bob")
3
4  _loop("default loop", {
5    _seq("a", "b", comment: "Are you here?")
6  })
7  _gap()
8  _loop("min loop", min: 1, {
9    _seq("a", "b", comment: "Are you here?")
10 })
11 _gap()
12 _loop("min-max loop", min: 1, max: 5, {
13   _seq("a", "b", comment: "Are you still here?")
14 })

```

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

```

1  _par("alice", display-name: "Alice")
2  _par("bob", display-name: "Bob")
3  _par("craig", display-name: "Craig")
4
5  _seq("bob", "alice") // Unsynchronized
6  _seq("bob", "craig") // "
7  _sync({
8    _seq("bob", "alice") // Synchronized
9    _seq("bob", "craig") // "
10 })
11 _seq("alice", "bob") // Unsynchronized
12 _seq("craig", "bob") // "
13 _sync({
14   _seq("alice", "bob") // Synchronized
15   _seq("craig", "bob") // "
16 })

```

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

The diagram illustrates a sequence of interactions between Alice and Bob. It is divided into two main phases: 'Initialization' and 'Repetition'. In the 'Initialization' phase, Alice sends a 'Request 1' and Bob returns a 'Response 1'. In the 'Repetition' phase, Alice sends a 'Request 2' and Bob returns a 'Response 2'. The diagram uses solid lines for requests and dashed lines for responses.

```

1  _par("a", display-name: "Alice")
2  _par("b", display-name: "Bob")
3
4  _sep[Initialization]
5  _seq("a", "b", comment: [Request 1])
6  _seq(
7    "b", "a",
8    comment: [Response 1],
9    dashed: true
10 )
11
12 _sep[Repetition]
13 _seq("a", "b", comment: [Request 2])
14 _seq(
15   "b", "a",
16   comment: [Response 2],
17   dashed: true
18 )

```

Parameters

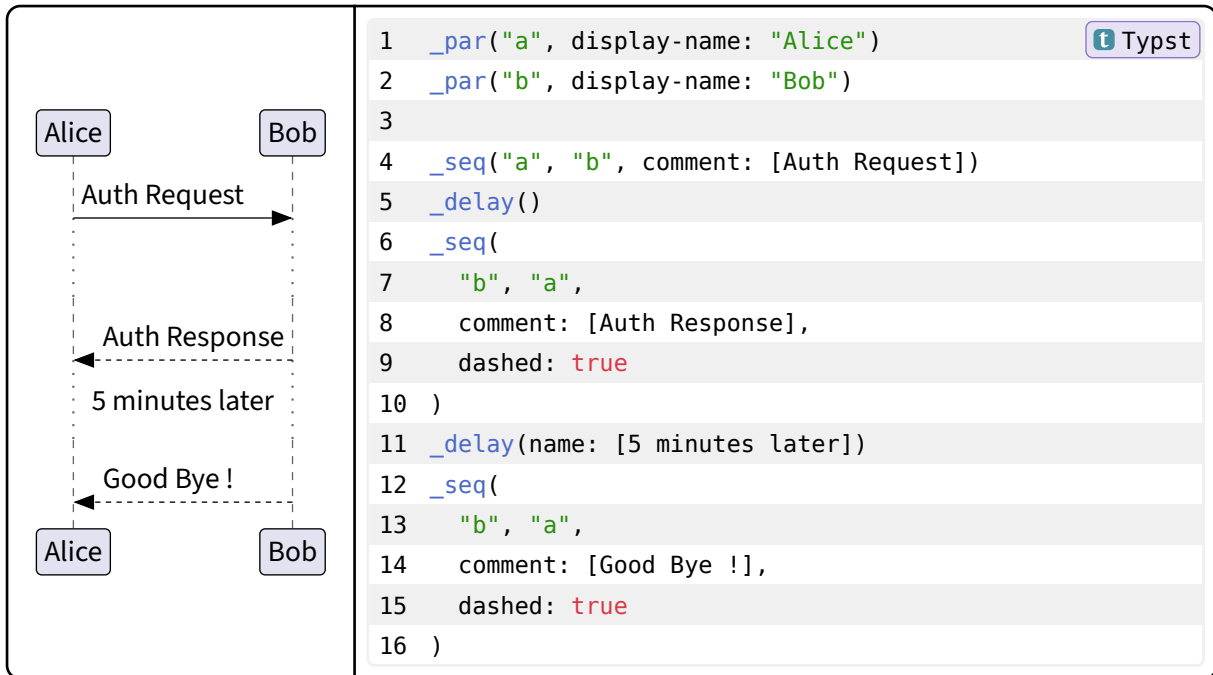
`_sep`(name: content)

name content

Name to display in the middle of the separator

4.4.2 `_delay`

Creates a delay before the next element



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



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
)
```

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`

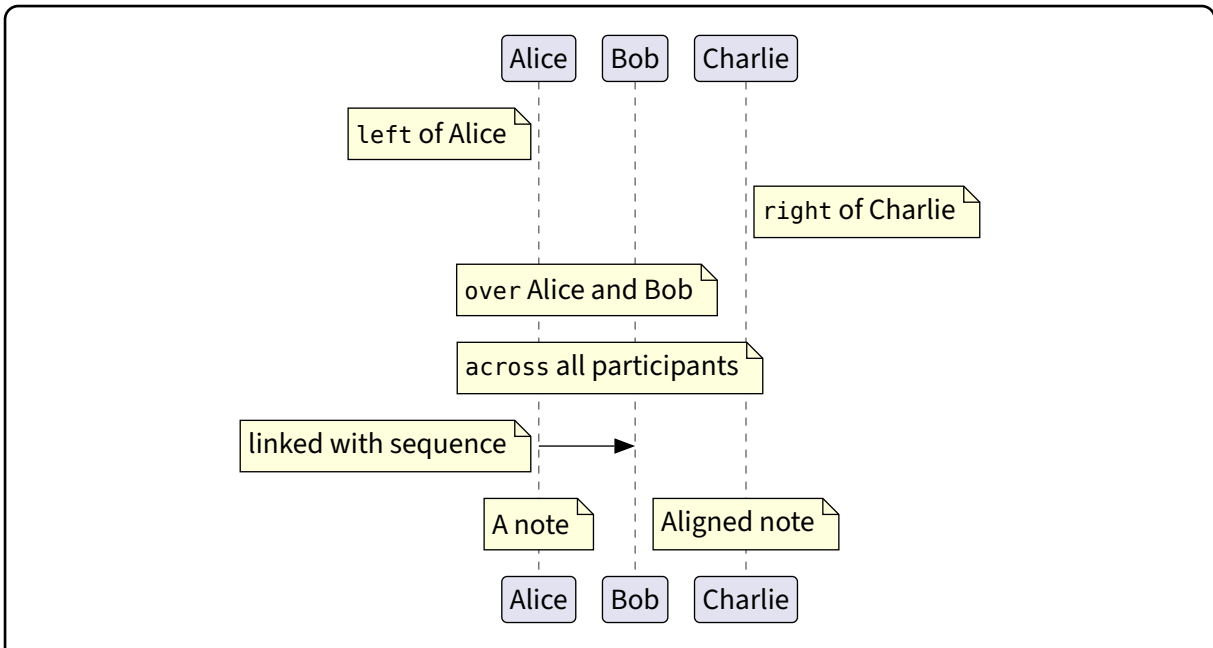
4.5.2 SHAPES

Accepted values for shape argument of `_note()`

<p>The diagram illustrates a flow of information or data. At the top, two boxes labeled 'Alice' and 'Bob' are connected by dashed lines to a yellow box labeled 'default'. From 'default', a dashed line leads to another yellow box labeled 'rect'. From 'rect', a dashed line leads to a yellow hexagon labeled 'hex'. Finally, dashed lines from 'hex' lead to two boxes labeled 'Alice' and 'Bob' at the bottom.</p>	<pre> 1 _par("alice", display-name: "Alice") 2 _par("bob", display-name: "Bob") 3 _note("over", `default`, pos: "alice") 4 _note("over", `rect`, pos: "bob", shape: "rect") 5 _note("over", `hex`, pos: ("alice", "bob"), shape: "hex") </pre> <p>Typst</p>
--	---

4.5.3 SIDES

Accepted values for side argument of `_note()`



```

1  _par("alice", display-name: "Alice")
2  _par("bob", display-name: "Bob")
3  _par("charlie", display-name: "Charlie")
4  _note("left", [ `left` of Alice], pos: "alice")
5  _note("right", [ `right` of Charlie], pos: "charlie")
6  _note("over", [ `over` Alice and Bob], pos: ("alice", "bob"))
7  _note("across", [ `across` all participants])
8  _seq("alice", "bob")
9  _note("left", [linked with sequence])
10 _note("over", [A note], pos: "alice")
11 _note("over", [Aligned note], pos: "charlie", aligned: true)

```