# beam

draw optics experiment setups with CeTZ

v0.1.1 2026-01-28

https://github.com/bendix4620/beam

## Abstract

In a landscape dominated by copy pasting inkscape templates, **beam** aims to simplify the creation of schematics for experiment setups in the field of optics.

## Table of Contents

## 1 About

I built this package, because I was frustrated with the available tools to draw schematics for simple optical setups. The options available to me were full-fledged simulation software, blender, and a certain Inkscape template. None of these suit my preferences – or skills.

Amazed by the simplicity of zap⚡, I gathered inspiration from colleagues and friends and started drawing some symbols and extended the framework on the fly.

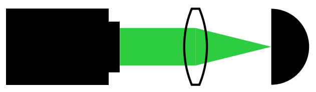
Please get in touch with me on github if you discover any bugs or have ideas for improvement!

## 2 Getting Started

beam is heavily inspired by zap⚡. The usage should feel very familiar to those accustomed to it. Just copy the example below and play around with the component parameters.

```
#import "@preview/beam:0.1.1"

#beam.setup({
    import beam: *
    // draw your setup, for example...
    laser("laser", (0, 0))
    lens("l1", (1, 0))
    detector("cam", (2, 0))
    beam("", "laser", "l1")
    focus("", "l1", "cam")
})
```



Be sure to check out the `examples` directory in the repository for more inspiration.

## 3 Styling

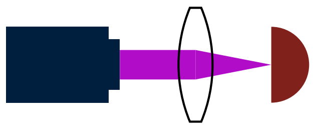
Styling works just like in CeTZ. However, beam uses a dedicated functions for styling to not interfere with other CeTZ-based libraries:

- `set-beam-style()` to change the current global style
- `get-beam-style()` to get the current global style from CeTZ's context

Style can also be configured locally on any given component.

```
#import "@preview/beam:0.1.1"

#beam.setup({
    import beam: *
    // change the global style
    set-beam-style(
        beam: (stroke: 11pt + purple),
        detector: (fill: red.darken(50%)),
    )
    laser("laser", (0, 0), fill: navy)
    lens("l1", (1, 0), scale: 1.5)
    detector("cam", (2, 0))
    beam("", "laser", "l1")
    focus("", "l1", "cam")
})
```



## 4 The Common Component Interface

All components (except `beam()`) accept the following parameters:

```
component(
  name: str,
  ..points-style: coordinate style,
  position: int float relative length ratio,
  rotate: angle,
  axis: auto bool style,
  debug: auto bool style,
  label: auto none content style
)
```

**name** `str`

Component identifyer. Used by cetz to reference the component

**..points-style** `coordinate` or `style`

Points (positional) and style (named) just like when using cetz's shapes

Components may support 1-3 points. These define how they are positioned and rotated

**1 point** places component at the given point. Rotate by passing `rotate: <angle>`

**2 points** place component between the given points and rotated to face in the direction of the latter point. Tune the position by passing `position: <...>` (for some components the position cannot be changed)

**3 points** place component at the 2nd given point and rotated to face the bisector of the angle spanned by the 3 points. This can emulate reflection/reflection, thus only refractive/reflective components support 3 point positioning.

Given style parameters are merged with the component's global style definition. You can check the default style of any given components with `#cetz.styles.resolve(beam.styles.default, root: "<component name>")`

The remaining arguments are referred to as `decoration`

**position** `int` or `float` or `relative` or `length` or `ratio` — default: `50%`

Position between start and end point. Only works when 2 points are given

**rotate** `angle` — default: `0deg`

Rotate the component. Only works when 1 point is given.

**axis** `auto` or `bool` or `style` — default: `auto`

optical axis decoration

- `auto` uses global style (equiv. to `(:)`)
- `bool` will turn axis on or off (equiv. to `(enabled: axis)`)
- `dictionary` will be merged with global style.

Valid dictionary keys are

```
enabled: bool
length: int float
stroke: stroke dictionary
```

Please refer to `sketch-axis` for the documentation of each field/parameter

**debug** `auto` or `bool` or `style` default: `auto`

debug info decoration

- `auto` uses global style
- `bool` will turn axis on or off (e.g. `(enabled: <value>)`)
- `dictionary` will be merged with global style.

Valid dictionary keys are

```
enabled: bool
stroke: stroke dictionary
radius: length
angle: angle
shift: length
inset: length dictionary
fsize: length
fill: color
```

Please refer to `sketch-debug` for the documentation of each field/parameter

**label** `auto` or `none` or `content` or `style` default: `auto`

label decoration

- `auto` uses global style
- `none` and `content` will overwrite the displayed content, e.g. `(content: <value>)`
- `dictionary` will be merged with global style.

Valid dictionary keys are

```
scope: str
pos: str angle
content: none content
anchor: auto str
angle: auto angle
padding: int float length dictionary
..style: style
```

Please refer to `sketch-label` for the documentation of each field/parameter

## 4.2 Anchors

Components come with a rotating bounding box and many anchors. Anchors `"east"`, `"north-east"`, `"north"`, `"north-west"`, `"west"`, `"south-west"`, `"south"`, `"south-east"` and `"center"` are placed at the corresponding bounding box positions and `"o"` at the component's position. `"in"` and `"out"` are placed at the first and last given point, in case 2 or 3 point positioning is used.

## 5 Components

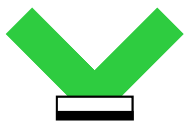
This section offers a comprehensive list of all the available components

- beam()
- beam-splitter()
- beam-splitter-plate()
- detector()
- fade()
- filter()
- filter-rot()
- focus()
- grating()
- laser()
- lens()
- mirror()
- objective()
- pinhole()
- prism()
- sample()

## 5.1 beam

draw a laser beam

```
#beam.setup({
    import beam: *
    mirror("m1", (0, 1), (1, 0), (2, 1))
    beam("", "m1.in", "m1", "m1.out")
})
```



### 5.1.a Points

Supports $\geq 2$ points

### 5.1.b Style

style id: `"beam"`

default values:

```
stroke: (
  thickness: 14pt,
  cap: "butt",
  join: "bevel",
  paint: rgb("#2ecc40"),
),
global-rotation: 0deg,
```

### 5.1.c Notes

Does not follow the component interface:

- no support for decorations ( `axis` , `label` , `debug` , `position` , `rotation` )
- no bounding box
- an anchor is placed at each given point (`"in"`, `"a"`, `"b"`, …, `"out"`)

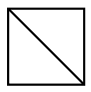
### 5.1.d Parameters

```
beam(
  name: str,
  ..points-style: coordinate style
)
```

## 5.2 beam-splitter

beam splitter cube

```
#beam.setup({
    import beam: *
    beam-splitter("", (0, 0))
})
```



### 5.2.a Points

Supports 1-3 points

### 5.2.b Style

style id: `"beam-splitter"`

default values:

```
scale: auto,
fill: none,
stroke: auto,
axis: auto,
label: auto,
debug: auto,
width: 1,
height: 1,
```

### 5.2.c Parameters

```
beam-splitter(
  name: name ,
  ..points-style-decoration: points  style  decoration ,
  flip: bool
)
```

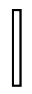
**flip** `bool` default: `false`

flip along local y-axis

### 5.3 beam-splitter-plate

beam splitter plate

```
#beam.setup({
    import beam: *
    beam-splitter-plate("", (0, 0))
})
```



#### 5.3.a Points

Supports 1-3 points

#### 5.3.b Style

style id: `"beam-splitter-plate"`

default values:

```
  scale: auto,
  fill: none,
  stroke: auto,
  axis: auto,
  label: (pos: 0deg),
  debug: auto,
  width: 0.2,
  height: 1,
```

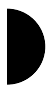
#### 5.3.c Parameters

```
beam-splitter-plate(
  name: str,
  ..points-style-decoration: coordinate style decoration
)
```

### 5.4 detector

detector / camera

```
#beam.setup({
    import beam: *
    detector("", (0, 0))
})
```



#### 5.4.a Points

Supports 1-2 points

#### 5.4.b Style

style id: `"detector"`

default values:

```
scale: auto,
fill: auto,
stroke: none,
axis: auto,
label: auto,
debug: auto,
radius: 0.5,
```

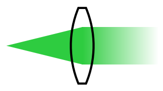
#### 5.4.c Parameters

```
detector(
  name: str,
  ..points-style-decoration: coordinate style decoration
)
```

## 5.5 fade

fading laser beam

```
#beam.setup({
    import beam: *
    lens("l1", (0, 0), (2, 0))
    focus("", "l1", "l1.in")
    fade("", "l1", "l1.out")
})
```



### 5.5.a Points

Supports 2 points

### 5.5.b Style

style id: `"beam"`

default values:

```
  stroke: (
    thickness: 14pt,
    cap: "butt",
    join: "bevel",
    paint: rgb("#2ecc40"),
  ),
  global-rotation: 0deg,
```

### 5.5.c Parameters

```
fade(
  name: str,
  ..points-style-decoration: coordinate style decoration,
  flip: bool
)
```

**flip** `bool` default: `false`

> flip the fade direction

### 5.6 filter

filter

```
#beam.setup({
    import beam: *
    filter("", (0, 0))
    flip-filter("", (2, 0))
})
```

#### 5.6.a Points

Supports 1-2 points

#### 5.6.b Style

style id: `"filter"`

default values:

```
  scale: auto,
  fill: luma(66.67%),
  stroke: auto,
  axis: auto,
  label: auto,
  debug: auto,
  width: 0.2,
  height: 1,
```

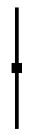
#### 5.6.c Parameters

```
filter(
  name: str,
  ..points-style-decoration: coordinate style decoration
)
```

## 5.7 filter-rot

rotational filter / filter wheel

```
#beam.setup({
    import beam: *
    filter-rot("", (0, 0))
})
```



### 5.7.a Points

Supports 1-2 points

### 5.7.b Style

style id: `"filter-rot"`

default values:

```
scale: auto,
fill: auto,
stroke: (thickness: 0.05),
axis: auto,
label: auto,
debug: auto,
diameter: 1.6,
```

### 5.7.c Parameters

```
filter-rot(
  name: str,
  ..points-style-decoration: coordinate style decoration,
  flip: bool
)
```

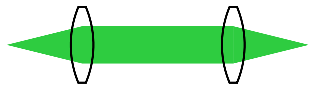
**flip** `bool` default: `false`

flip the filter

## 5.8 focus

focusing laser beam

```
#beam.setup({
    import beam: *
    lens("l1", (1, 0))
    lens("l2", (3, 0))
    beam("", "l1", "l2")
    focus("", (0, 0), "l1", flip: true)
    focus("", "l2", (4, 0))
})
```



### 5.8.a Points

Supports 2 points

### 5.8.b Style

style id: `"beam"`

default values:

```
  stroke: (
    thickness: 14pt,
    cap: "butt",
    join: "bevel",
    paint: rgb("#2ecc40"),
  ),
  global-rotation: 0deg,
```

### 5.8.c Parameters

```
focus(
  name: str,
  ..points-style-decoration: coordinate style decoration,
  flip: bool
)
```

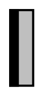
**flip** `bool` default: `false`

> flip the focus direction

## 5.9 grating

refraction grating

```
#beam.setup({
    import beam: *
    grating("", (0, 0))
})
```



### 5.9.a Points

Supports 1 or 3 points

### 5.9.b Style

style id: `"grating"`

default values:

```
  scale: auto,
  fill: luma(75%),
  stroke: auto,
  axis: auto,
  label: (pos: 0deg),
  debug: auto,
  width: 0.3,
  height: 1,
```

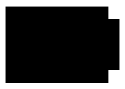
### 5.9.c Parameters

```
grating(
  name: str,
  ..points-style-decoration: points style decoration
)
```

## 5.10 laser

laser source

```
#beam.setup({
    import beam: *
    laser("", (0, 0))
})
```



### 5.10.a Points

Supports 1-2 points

### 5.10.b Style

style id: `"laser"`

default values:

```
scale: auto,
fill: auto,
stroke: none,
axis: auto,
label: auto,
debug: auto,
length: 1.5,
height: 1,
```

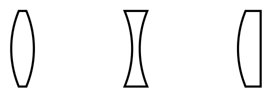
### 5.10.c Parameters

```
laser(
  name: str,
  ..points-style-decoration: coordinate style decoration,
  position: int float relative length ratio = 0%
)
```

## 5.11 lens

lens

```
#beam.setup({
    import beam: *
    lens("", (0, 0))
    lens("", (1.5, 0), kind: ")(")
    lens("", (3, 0), kind: "(|")
})
```



### 5.11.a Points

Supports 1-2 points

### 5.11.b Style

style id: `"lens"`

default values:

```
scale: auto,
fill: none,
stroke: auto,
axis: auto,
label: auto,
debug: auto,
width: 0.1,
height: 1,
extent: 0.1,
```

### 5.11.c Parameters

```
lens(
  name: str,
  kind: str,
  ..points-style-decoration: coordinate style decoration
)
```

**kind** `str` default: `"()"`

> what kind of lens to draw. Supported lenses are `"()"`, `"(("`, `"))"`, `")("`, `"(|"`, `")|"`, `"|)"`, `"|("` and `"||"`

### 5.12 mirror

mirror

```
#beam.setup({
    import beam: *
    mirror("", (0, 0))
    flip-mirror("", (2, 0))
})
```

#### 5.12.a Points

Supports 1 or 3 points

#### 5.12.b Style

style id: `"mirror"`

default values:

```
  scale: auto,
  fill: luma(100%),
  stroke: auto,
  axis: auto,
  label: (pos: 0deg),
  debug: auto,
  width: 0.3,
  height: 1,
  extent: 0.1,
```

#### 5.12.c Parameters

```
mirror(
  name: str,
  kind: str,
  ..points-style-decoration: points style decoration
)
```

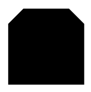
**kind** `str` default: `"|"`

> what kind of mirror to draw. Supported mirrors are `"("`, `")"` and `"|"`

## 5.13 objective

microscopy objective

```
#beam.setup({
    import beam: *
    objective("", (0, 0), rotate: 90deg)
})
```



### 5.13.a Points

Supports 1-2 points

### 5.13.b Style

style id: `"objective"`

default values:

```
scale: auto,
fill: auto,
stroke: none,
axis: (length: 2),
label: auto,
debug: auto,
width: 1,
height: 1,
```

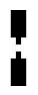
### 5.13.c Parameters

```
objective(
  name: str,
  ..points-style-decoration: coordinate style decoration
)
```

## 5.14 pinhole

pinhole / aperture

```
#beam.setup({
    import beam: *
    pinhole("", (0, 0))
})
```



### 5.14.a Points

Supports 1-2 points

### 5.14.b Style

style id: `"pinhole"`

default values:

```
scale: auto,
fill: auto,
stroke: none,
axis: auto,
label: auto,
debug: auto,
width: 0.2,
height: 1,
gap: 0.1,
```

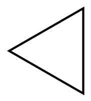
### 5.14.c Parameters

```
pinhole(
  name: str,
  ..points-style-decoration: coordinate style decoration
)
```

## 5.15 prism

dispersive prism

```
#beam.setup({
    import beam: *
    prism("", (0, 0))
})
```



### 5.15.a Points

Supports 1 or 3 points

### 5.15.b Style

style id: `"prism"`

default values:

```
scale: auto,
fill: none,
stroke: auto,
axis: auto,
label: (pos: 0deg),
debug: auto,
radius: 0.65,
```

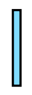
### 5.15.c Parameters

```
prism(
  name: str,
  ..points-style-decoration: coordinate  style  decoration
)
```

## 5.16 sample

sample

```
#beam.setup({
    import beam: *
    sample("", (0, 0))
})
```



### 5.16.a Points

Supports 1-2 points

### 5.16.b Style

style id: `"sample"`

default values:

```
scale: auto,
fill: rgb("#7fdbff"),
stroke: auto,
axis: auto,
label: (pos: 0deg),
debug: auto,
width: 0.1,
height: 1,
```

### 5.16.c Parameters

```
sample(
  name: str,
  ..points-style-decoration: points style decoration
)
```

## 6 Custom Components

Custom components can be easily created with the help of `component()` and `interface()`. The example below creates a simple rectangle as a component. You can use it as a starting point to draw all the components you need.

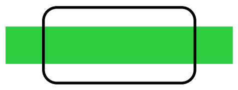
```
#import "@preview/beam:0.1.1"
#import beam: cetz, component, interface

// draw a simple rectangle
#let custom(name, ..params) = {
    let sketch(ctx, points, style) = {
        let w = style.at("width", default: 2)
        let h = style.at("height", default: 1)

        // create bounding box
        interface(
            (-w / 2, -h / 2),
            (w / 2, h / 2),
            io: points.len() < 2,
        )

        // draw the component
        cetz.draw.rect(
            "bounds.north-east",
            "bounds.south-west",
            ..style,
        )
    }
    component("my-custom-component", sketch: sketch,
name, ..params)
}

#beam.setup({
    import beam: *
    // Styling works out of the box!
    set-beam-style(my-custom-component: (radius: 5pt))
    custom("c", (0, 0), (3, 0))
    beam("", "c.in", "c.out")
})
```



## 7 Internals

### 7.1 anchor-to-angle

Get the angle corresponding to an anchor or the anchor itself if it cannot be associated with an angle.

```
#("east", "north", "center").map(anchor-to-angle)
```

```
(0deg, 90deg, "center")
```

#### 7.1.a Parameters

```
anchor-to-angle(anchor: any ) -> any angle
```

### 7.2 angle-to-anchor

Get the anchor closest to an angle.

```
#(0deg, 35deg).map(angle-to-anchor)
```

```
("east", "north-east")
```

#### 7.2.a Parameters

```
angle-to-anchor(angle: angle ) -> str
```

### 7.3 component

Handle component creation

This function automates the positioning and rotation of components, resolves the style parameters, and adds in available decorations.

## 7.3.a Parameters

```
component(
  root: str,
  sketch: function,
  num-points: array,
  name: str,
  ..points-style: coordinate style,
  position: int float relative length ratio,
  rotate: angle,
  axis: auto bool style,
  debug: auto bool style,
  label: auto none content style
)
```

**root** `str`

> Component type identifyer. Used to find the correct style

**sketch** `function` default: `(ctx, points, style) => {}`

> Function that draws the component. Takes `context`, `array` of `vector` and `style`

**num-points** `array` default: `(1, 2)`

> Number of points supported by the component

**name** `str`

> Component identifyer. Used by cetz to reference the component

**..points-style** `coordinate` or `style`

> Points (positional) and style (named) just like when using cetz's shapes
>
> Components may support 1-3 points. These define how they are positioned and rotated
> **1 point** places component at the given point. Rotate by passing `rotate: <angle>`
> **2 points** place component between the given points and rotated to face in the direction of the latter point. Tune the position by passing `position: <...>` (for some components the position cannot be changed)
> **3 points** place component at the 2nd given point and rotated to face the bisector of the angle spanned by the 3 points. This can emulate reflection/reflection, thus only refractive/reflective components support 3 point positioning.
>
> Given style parameters are merged with the component's global style definition. You can check the default style of any given components with `#cetz.styles.resolve(beam.styles.default, root: "<component name>")`
>
> The remaining arguments are referred to as `decoration`

**position** `int` or `float` or `relative` or `length` or `ratio` default: `50%`

> Position between start and end point. Only works when 2 points are given

**rotate** `angle` default: `0deg`

> Rotate the component. Only works when 1 point is given.

**axis** `auto` or `bool` or `style` default: `auto`

> optical axis decoration
>
> - `auto` uses global style (equiv. to `(:)`)
> - `bool` will turn axis on or off (equiv. to `(enabled: axis)`)
> - `dictionary` will be merged with global style.
>
> Valid dictionary keys are
>
> ```
> enabled: bool
> length: int float
> stroke: stroke dictionary
> ```
>
> Please refer to `sketch-axis()` for the documentation of each field/parameter

**debug** `auto` or `bool` or `style` default: `auto`

> debug info decoration
>
> - `auto` uses global style
> - `bool` will turn axis on or off (e.g. `(enabled: <value>)`)
> - `dictionary` will be merged with global style.
>
> Valid dictionary keys are
>
> ```
> enabled: bool
> stroke: stroke dictionary
> radius: length
> angle: angle
> shift: length
> inset: length dictionary
> fsize: length
> fill: color
> ```
>
> Please refer to `sketch-debug()` for the documentation of each field/parameter

**label** `auto` or `none` or `content` or `style` default: `auto`

> label decoration
>
> - `auto` uses global style
> - `none` and `content` will overwrite the displayed content, e.g. `(content: <value>)`
> - `dictionary` will be merged with global style.
>
> Valid dictionary keys are
>
> ```
> scope: str
> pos: str angle
> content: none content
> anchor: auto str
> angle: auto angle
> padding: int float length dictionary
> ..style: style
> ```
>
> Please refer to `sketch-label()` for the documentation of each field/parameter

### 7.4 get-beam-style

get currently active style

#### 7.4.a Parameters

```
get-beam-style(ctx: style)
```

### 7.5 init-beam

initialize beam

Useful when working with other cetz extensions (for example zap) that bring their own canvas

```
#cetz.canvas({
  import cetz.draw: *
  init-beam()
  // draw setup here
})
```

#### 7.5.a Parameters

```
init-beam()
```

### 7.6 interface

Create a bounding box for a component

cetz's `rect-around()` does not work properly on groups with rotation, so a manual bounding box is necessary

#### 7.6.a Parameters

```
interface(
  ll: coordinate,
  ur: coordinate,
  io: bool
)
```

**ll** `coordinate`

> lower left point of the bbox

**ur** `coordinate`

> upper right point of the bbox

**io** `bool` `default: false`

> wether to automatically create input and output anchors

### 7.7 normalize-angle

force an angle to the range [0, 360°]

### 7.7.a Parameters

```
normalize-angle(a: angle) -> angle
```

## 7.8 opposite-anchor

Get the opposite anchor

| | |
|---|---|
| `#("east", "north", "center").map(opposite-anchor)` | `("west", "south", "center")` |

### 7.8.a Parameters

```
opposite-anchor(anchor: str) -> str
```

## 7.9 set-beam-style

change component style for the entire scope

### 7.9.a Parameters

```
set-beam-style(..style: style)
```

## 7.10 setup

beam’s canvas wrapper. Takes care of proper initialization

```
#beam.setup({
    import beam: *
    // draw setup here
})
```

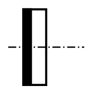
### 7.10.a Parameters

```
setup(
  body,
  preamble = none,
  ..params
)
```

## 7.11 sketch-axis

draw the optical axis

```
#beam.setup({
  import beam: *
  mirror("m1", (), axis: true)
})
```



### 7.11.a Parameters

```
sketch-axis(
  enabled: bool ,
  length: int  float ,
  stroke: stroke  dictionary
)
```

**enabled** `bool` default: `false`

> Wether to draw the axis

**length** `int` or `float` default: `1`

> axis length

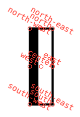
**stroke** `stroke` or `dictionary` default: `(paint: black, thickness: .5pt, dash: "densely-dash-dotted")`

> axis stroke

## 7.12 sketch-debug

draw debug information

```
#beam.setup({
  import beam: *
  mirror("m1", (), debug: true)
})
```



### 7.12.a Parameters

```
sketch-debug(
  name: str ,
  enabled: bool ,
  stroke: stroke  dictionary ,
  radius: length ,
  angle: angle ,
  shift: length ,
  inset: length  dictionary ,
  fsize: length ,
  fill: color
)
```

**enabled** `bool` default: `false`

> wether to draw debug info

**stroke** `stroke` or `dictionary` default: `.2pt + red`

> anchor marker stroke

**radius** `length` default: `.7pt`

> anchor marker radius

**angle** `angle` default: `-30deg`

> anchor name rotation

**shift** `length` default: `3pt`

> anchor name shift

**inset** `length` or `dictionary` default: `1pt`

> anchor name inset

**fsize** `length` default: `3pt`

> anchor name font size

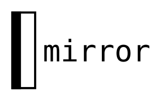
**fill** `color` default: `red`

> anchor name text color

## 7.13 sketch-label

draw the component label

```
#beam.setup({
  import beam: *
  mirror("m1", (), label: [mirror])
})
```



### 7.13.a Parameters

```
sketch-label(
  name: str,
  rotation: angle,
  scope: str,
  pos: str angle,
  content: none content,
  anchor: auto str,
  angle: auto angle,
  padding: int float length dictionary,
  ..style: style
)
```

**name** `str`

> component's name (identifier)

**rotation** `angle`

> component's rotation

**scope** `str` default: `"local"`

> ```
>            north/90deg
> 
>               ______
>              |      |
> west/180deg  |      | east/0deg
>              |______|
> 
>           south/270deg
> ```
>
> Imagine a component with anchors like above. The `scope` argument defines how the given angle is interpreted in terms of the components rotation. This does not account for external rotations on the canvas.
>
> - `"local"` the position is resolve relative to the component bbox after rotation
> - `"parent"` the position is resolved relative to the component bbox before rotation

**pos** `str` or `angle` default: `90deg`

> where to position the label

**content** `none` or `content` default: `none`

> the label content

**anchor** `auto` or `str` default: `auto`

> label anchor. `auto` will try to pick anchor so that label and component do not overlap

**angle** `auto` or `angle` default: `0deg`

> rotate the label. `auto` will rotate the label with its position

**padding** `int` or `float` or `length` or `dictionary` default: `.1`

> label content padding

**..style** `style`

> additional styling passed to cetz's `content()`