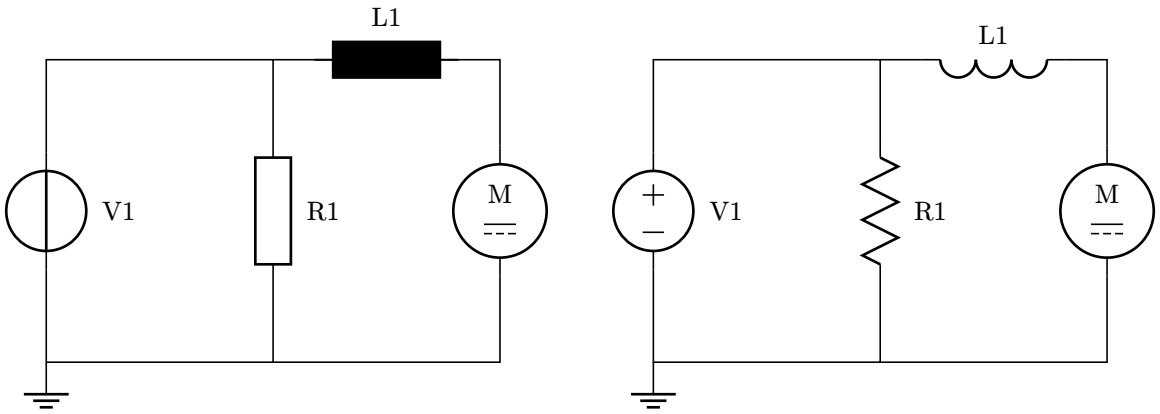


Circuitor

The Typst package to draw standard circuits

v0.1.0 (15/05/2025)

Louis Grange

May 15, 2025

Contents

1 Introduction

I'm Louis Grange, an engineering student at EPFL (École Polytechnique Fédérale de Lausanne), and I created Circuitor in April 2025 to address a missing piece in the Typst ecosystem: a library for drawing electrical circuits that adhere to IEC and IEEE standards.

At the time, no such solution existed, so I decided to build one 🚧. Circuitor started as a personal project but quickly grew into a structured, reusable library for creating clear, standards-compliant circuit diagrams directly in Typst.

My goal with Circuitor is to make it easier for students, educators, and professionals to integrate high-quality circuit drawings into their documents without leaving the Typst environment. Whether you're preparing lecture notes, technical reports, or academic publications, Circuitor aims to deliver both precision and simplicity 🤺.

2 Getting started

2.1 Installation

Circuitor is easy to set up and works out of the box 📦 with Typst Universe. There are two main ways to start using the library, depending on your workflow.

2.1.1 Typst Universe package

If you're working in the online Typst app, you can import Circuitor directly with a single line:

```
#import "@preview/circuitor:0.1.0"
```

2.1.2 Local package

If you prefer to work offline or want to explore the source code, you can also use Circuitor by downloading it locally from the GitHub repository.

1. Clone or download the repository to your computer.
2. In your Typst project, import the `exports.typ` file located in the `src` folder:

```
#import "PATH_TO_CIRCUITOR/src/exports.typ"
```

Make sure the path reflects the actual location of the downloaded folder relative to your `.typ` file.

2.2 Mimium requirements

This table shows the minimum required versions of Typst and CeTZ for each release of the Circuitor library.

Circuitor	Typst	CeTZ
`0.1.0`	`0.13.1`	`0.3.4`

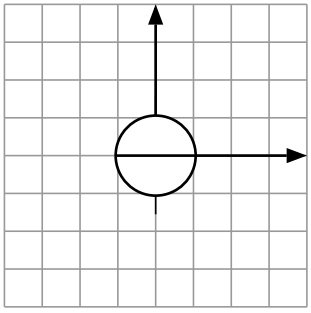
2.3 Report bugs or suggest features and components

I actively maintain Circuitor and greatly value community feedback 💬.

If you encounter a bug 🐞, spot an incorrect symbol, or have an idea for a new feature or electrical component, feel free to open an issue or a discussion on the GitHub repository.

3 Core concepts

3.1 CeTZ coordinate system



The `CeTZ` library uses a coordinate system with magic anchors.

Every component is by default drawn from it's center `(0,0)` in the vertical direction.

You can use any anchor from every component, such as `center`, `south`, `north`, etc. using the following syntax. The first parameter `uid` is mandatory and used for automatically creating anchors around the component.

```
#canvas({
    resistor("r1", "myOtherComponent.south")
})
```

3.2 Components

At the heart of Circuitor lies the `component` function. Every electrical symbol in the library is built on top of this function. It provides a unified structure to place, draw, label, and connect components in a flexible and composable way.

```
#let component(
    uid,
    position,
    draw,
    label: none,
    variant: "iec",
    wires: true,
    scale: 1.0,
    rotate: 0deg,
    label-anchor: none,
    ..params
) = { ... }
```

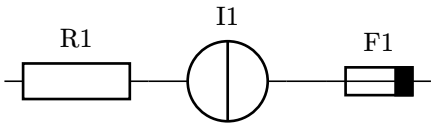
This low-level function is not meant to be used directly in most cases 😅, but understanding it helps if you want to define your own components or customize existing ones.

3.3 Simple example

To get started with Circuitor, here's a minimal working example showing how to draw a simple circuit with just a few lines of code.

```
#import "@preview/circuitor:0.1.0"

#canvas({
    resistor("r1", (0,0), label: "I1", rotate: 90deg)
    isource("i1", (1.7,0), rotate: -135deg, label: "I1", rotate: 90deg)
    afuse("f2", (3.5,0), label: "F1", rotate: 90deg)
    wire("r1.out", "i1.minus")
    wire("i1.plus", "f2.in")
})
```

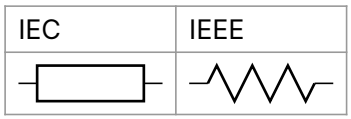


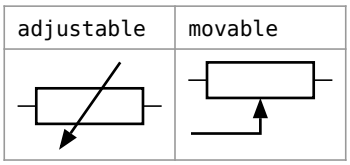
4 Components

Circuitor provides a wide set of predefined components based on IEC and IEEE standards 📎. These include resistors, capacitors, voltage sources, switches, diodes, transistors and more. Each component is built on top of the core `component` function and is ready to use with sensible defaults ⚙️.

4.1 Resistors

```
#let resistor(
    uid,
    position,
    adjustable: false,
    movable: false,
    ..params
) = { ... }
```

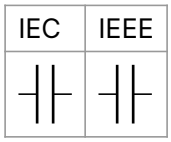




Available shortcuts: `potentiometer` for `adjustable: true`

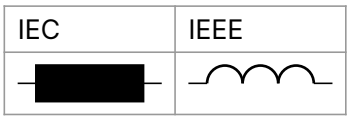
4.2 Capacitors

```
#let capacitor(
    uid,
    position,
    adjustable: false,
    ..params
) = { ... }
```



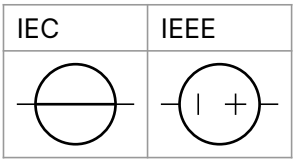
4.3 Inductors

```
#let inductor(
    uid,
    position,
    ..params
) = { ... }
```



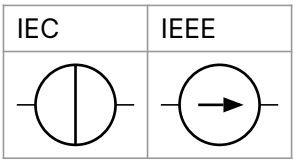
4.4 Voltage sources

```
#let vsource(
    uid,
    position,
    ..params
) = { ... }
```



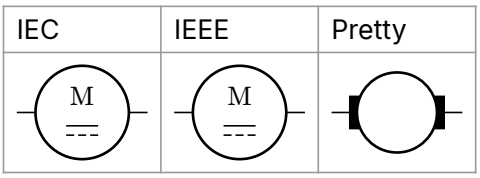
4.5 Current sources

```
#let isource(
    uid,
    position,
    ..params
) = { ... }
```



4.6 Motors

```
#let dcmotor(
    uid,
    position,
    ..params
) = { ... }
```



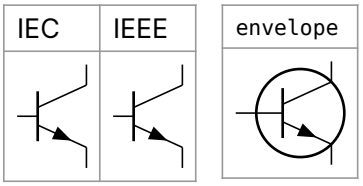
4.7 Fuses

```
#let fuse(
    uid,
    position,
    asymmetric: false,
    ..params
) = { ... }
```

Available shortcuts: `afuse` for `asymmetric: true`

4.8 Bipolar junction transistors (BJTs)

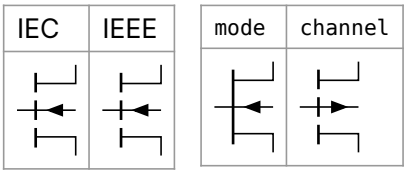
```
#let bjt(
    uid,
    position,
    polarisation: "npn",
    envelope: false
    ..params
) = { ... }
```



Available shortcuts: `npn` for `polarisation: npn`, `pnp` for `polarisation: pnp`

4.9 MOSFETs

```
#let mosfet(
    uid,
    position,
    channel: "n",
    envelope: false,
    gates: 1,
    mode: "enhancement",
    substrate: "internal"
    ..params
) = { ... }
```



Available shortcuts: `pmos` for `channel: p`, `nmos` for `channel: n`

4.10 Custom components

Circuitor is not limited to the built-in components, you can also create your own ✏️.

At the core of every element is the `component` function, which gives you full control over the drawing logic, positioning, orientation, labeling, and wiring behavior. To define a custom component, simply write your own drawing function and pass it to `component`, along with metadata like position and UID.

```
#let mycustomcomponent(uid, position, ..params) = {
    // Drawing function
    let draw(variant, scale, rotate, wires, ..styling) = {
        rect((-1,-1), (0,0))
    }
    // Low-level component call
    component(uid, position, draw, ..params)
}
```

If you've designed a component that you're proud of 😎, feel free to open an issue on the GitHub repository and propose it for inclusion in a future release!