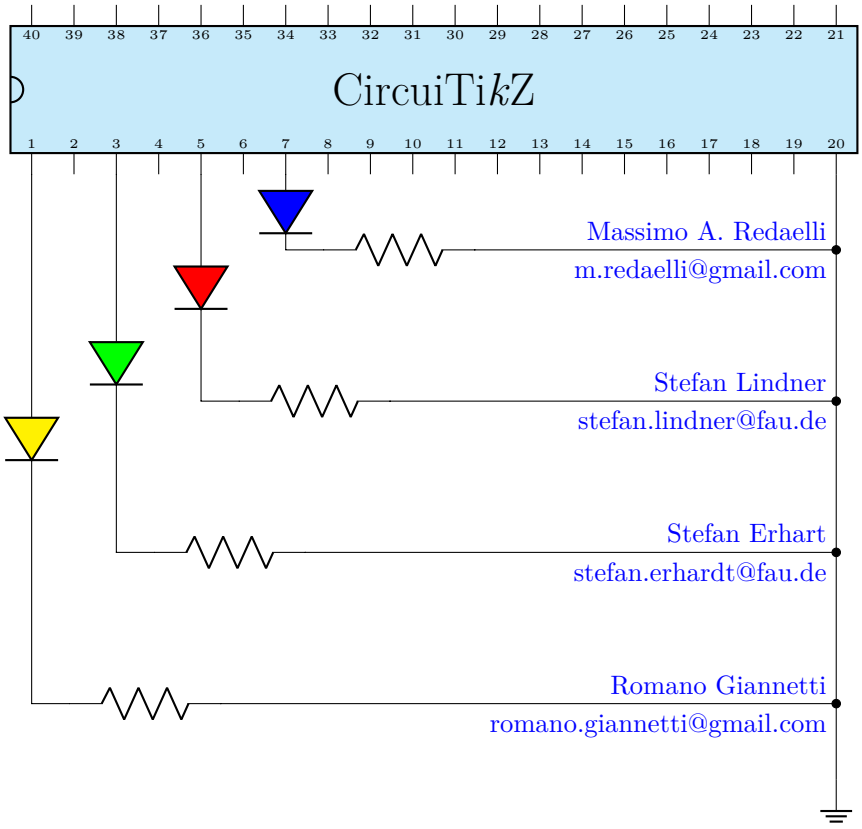


CircuiTi*k*Z
version git:f9472b0 (2019/09/01)

Massimo A. Redaelli (m.redaelli@gmail.com)
Stefan Lindner (stefan.lindner@fau.de)
Stefan Erhardt (stefan.erhardt@fau.de)
Romano Giannetti (romano.giannetti@gmail.com)

September 1, 2019

Contents

1 Introduction **5**

1.1 About . . . 5
1.2 License . . . 5
1.3 Loading the package . . . 5
1.4 Installing a new version of the package. . . . 5
1.5 Requirements . . . 6
1.6 Incompatible packages . . . 6
1.7 Known bugs and limitation . . . 6
1.8 Incompabilities between version . . . 6
1.9 Feedback . . . 7
1.10 Package options . . . 7

2 Tutorials **10**

2.1 Getting started with CircuiTi*k*Z: a current shunt . . . 10
2.2 A more complex tutorial: circuits, Romano style. . . . 12

3 The components **17**

3.1 Path-style components . . . 17
3.1.1 Anchors . . . 17
3.1.2 Customization . . . 18
3.1.2.1 Components size . . . 18
3.1.2.2 Thickness of the lines . . . 19
3.1.2.3 Shape of the components . . . 20
3.1.3 Descriptions . . . 20
3.2 Node-style components . . . 21
3.2.1 Mirroring and flipping . . . 21
3.2.2 Anchors . . . 21
3.2.3 Descriptions . . . 22
3.3 Styling circuits and components . . . 22
3.3.1 Relative size . . . 23
3.3.2 Fill color . . . 25
3.3.3 Line thickness . . . 26
3.3.4 Style files . . . 26
3.3.5 Style files: how to write them . . . 27
3.4 Grounds and supply voltages . . . 28
3.4.1 Grounds . . . 28
3.4.1.1 Grounds anchors . . . 29
3.4.1.2 Grounds customization . . . 29
3.4.2 Power supplies . . . 29

3.4.2.1 Power supply anchors . . . 29
3.4.2.2 Power supplies customization . . . 29
3.5 Resistive bipoles . . . 30
3.5.1 Generic sensors anchors . . . 32
3.5.2 Resistive components customization . . . 32
3.6 Capacitors and inductors: dynamical bipoles . . . 33
3.6.1 Capacitors . . . 33
3.6.2 Capacitive sensors anchors . . . 33
3.6.3 Capacitors customizations . . . 33
3.6.4 Inductors . . . 34
3.6.5 Inductive sensors anchors . . . 34
3.6.6 Inductors customizations . . . 35
3.7 Diodes and such . . . 35
3.7.1 Tripole-like diodes . . . 38
3.7.2 Triacs anchors . . . 38
3.7.3 Diode customizations . . . 39
3.8 Sources and generators . . . 39
3.8.1 Batteries . . . 39
3.8.2 Stationary sources . . . 39
3.8.3 Sinusoidal sources . . . 40
3.8.4 Controlled sources . . . 41
3.8.5 Noise sources . . . 42
3.8.6 Special sources . . . 43
3.8.7 DC sources . . . 43
3.8.8 Sources customizations . . . 44
3.9 Instruments . . . 44
3.9.1 Instruments customizations . . . 45
3.9.2 Rotation-invariant elements . . . 45
3.9.3 Instruments as node elements . . . 46
3.9.4 Measuring voltage and currents, multiple ways . . . 46
3.10 Mechanical Analogy . . . 49
3.10.1 Mechanical elements customizations . . . 49
3.11 Miscellaneous bipoles . . . 49
3.11.1 Miscellanous element customization . . . 50
3.12 Crossings . . . 51
3.13 Arrows . . . 52
3.13.1 Arrows size . . . 52
3.14 Terminal shapes . . . 53
3.14.1 BNC connector/terminal . . . 53
3.15 Block diagram components . . . 54
3.15.1 Blocks anchors . . . 56

3.15.2 Blocks customization . . . 57
3.15.2.1 Multi ports . . . 57
3.15.2.2 Labels and custom two-port boxes . . . 58
3.15.2.3 Box option . . . 58
3.15.2.4 Dash optional parts . . . 58
3.16 Transistors . . . 58
3.16.1 Transistors anchors . . . 62
3.16.2 Transistor paths . . . 64
3.16.3 Transistors customization . . . 65
3.17 Electronic Tubes . . . 65
3.17.1 Tubes customization . . . 67
3.18 RF components . . . 69
3.18.1 RF elements customization . . . 70
3.18.2 Microstrip customization . . . 70
3.19 Electro-Mechanical Devices . . . 71
3.19.1 Electro-Mechanical Devices anchors . . . 71
3.20 Double bipoles (transformers) . . . 72
3.20.1 Double dipoles anchors . . . 73
3.20.2 Double dipoles customization . . . 74
3.21 Amplifiers . . . 75
3.21.1 Amplifiers anchors . . . 76
3.21.2 Amplifiers customization . . . 78
3.21.2.1 European-style amplifier customization . . . 78
3.22 Switches and buttons . . . 80
3.22.1 Traditional switches . . . 80
3.22.2 Cute switches . . . 81
3.22.2.1 Cute switches anchors . . . 82
3.22.2.2 Cute switches customization . . . 82
3.22.3 Rotary switches . . . 83
3.22.3.1 Rotary switch anchors . . . 84
3.22.3.2 Rotary switch customization . . . 85
3.23 Logic gates . . . 85
3.23.1 American Logic gates . . . 86
3.23.2 European Logic gates . . . 86
3.23.3 Special components . . . 87
3.23.4 Logic port customization . . . 87
3.23.5 Logic port anchors . . . 88
3.24 Chips (integrated circuits) . . . 89
3.24.1 DIP and QFP chips customization . . . 90
3.24.2 Chips anchors . . . 91
3.24.3 Chips rotation . . . 91
3.24.4 Chip special usage . . . 91
3.25 Seven segment displays . . . 92
3.25.1 Seven segments anchors . . . 93
3.25.2 Seven segments customization . . . 93

4 Labels and similar annotations **94**
4.1 Labels and Annotations . . . 94
4.2 Currents and voltages . . . 96
4.3 Currents . . . 100
4.4 Flows . . . 101
4.5 Voltages . . . 102
4.5.1 European style . . . 102
4.5.2 American style . . . 103
4.5.3 Voltage position . . . 104
4.5.4 American voltages customization . . . 104
4.5.5 Global properties of voltages and currents . . . 105
4.6 Nodes (also called poles) . . . 106
4.7 Special components . . . 108
4.8 Integration with `siunitx` . . . 110
4.9 Mirroring and Inverting . . . 110
4.10 Putting them together . . . 111
4.11 Line joins between Path Components . . . 111

5 Colors **112**
5.1 Shape colors . . . 112
5.2 Fill colors . . . 113

6 FAQ **115**

7 Defining new components **116**
7.1 Suggested setup . . . 116
7.2 Path-style component . . . 117
7.3 Node-style component . . . 120
7.3.1 Finishing your work . . . 120

8 Examples **121**

9 Changelog and Release Notes **127**

Index of the components **133**

1 Introduction

1.1 About

CircuiTi*k*Z was initiated by Massimo Redaelli in 2007, who was working as a research assistant at the Polytechnic University of Milan, Italy, and needed a tool for creating exercises and exams. After he left University in 2010 the development of CircuiTi*k*Z slowed down, since LaTeX is mainly established in the academic world. In 2015 Stefan Lindner and Stefan Erhardt, both working as research assistants at the University of Erlangen-Nürnberg, Germany, joined the team and now maintain the project together with the initial author. In 2018 Romano Giannetti, full professor of Electronics at Comillas Pontifical University of Madrid, joined the team.

The use of CircuiTi*k*Z is, of course, not limited to academic teaching. The package gets widely used by engineers for typesetting electronic circuits for articles and publications all over the world.

1.2 License

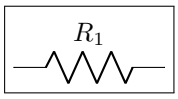
Copyright © 2007–2019 Massimo Redaelli. This package is author-maintained. Permission is granted to copy, distribute and/or modify this software under the terms of the LaTeX Project Public License, version 1.3.1, or the GNU Public License. This software is provided 'as is', without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

1.3 Loading the package

LaTeX	ConTeXt[1]
`\usepackage{circuitikz}`	`\usemodule[circuitikz]`

Ti*k*Z will be automatically loaded.

CircuiTi*k*Z commands are just Ti*k*Z commands, so a minimum usage example would be:




```
1 \tikz \draw (0,0) to[R=$R_1$] (2,0);
```

1.4 Installing a new version of the package.

The stable version of the package should come with your LaTeX distribution. Downloading the files from CTAN and installing them locally is, unfortunately, a distribution-dependent task and sometime not so trivial. If you search for `local texmf tree` and the name of your distribution on https://tex.stackexchange.com/ you will find a lot of hints.

Anyway, the easiest way of using whichever version of CircuiTi*k*Z is to point to the github page https://circuitikz.github.io/circuitikz/ of the project, and download the version you want. You will download a simple (biggish) file, called `circuitikz.sty`.

Now you can just put this file in your local `texmf` tree, if you have one, or simply adding it into the same directory where your main file resides, and then use

```
\usepackage[...options...]{circuitikzgit}
```

instead of `circuitikz`. This is also advantageous for "future resilience"; the authors try hard not to break backward compatibility with new versions, but sometime things happen.

[1] ConTeXt support was added mostly thanks to Mojca Miklavec and Aditya Mahajan.

1.5 Requirements

- `tikz`, version ≥ 3;
- `xstring`, not older than 2009/03/13;
- `siunitx`, if using `siunitx` option.

1.6 Incompatible packages

Ti*k*Z's own `circuit` library, which is based on CircuiTi*k*Z, (re?)defines several styles used by this library. In order to have them work together you can use the `compatibility` package option, which basically prefixes the names of all CircuiTi*k*Z `to[]` styles with an asterisk.

So, if loaded with said option, one must write `(0,0) to[*R] (2,0)` and, for transistors on a path, `(0,0) to[*Tnmos] (2,0)`, and so on (but `(0,0) node[nmos] {}`). See example at page 126.

1.7 Known bugs and limitation

CircuiTi*k*Z will **not work** correctly with global (in the main `circuitikz` environment, or in `scope` environments) *negative* scale parameters (`scale`, `xscale` or `yscale`), unless `transform shape` is also used, and even in this cases the behavior is not guaranteed. Neither it will work with angle-changing scaling (when `xscale` is different form `yscale`) and with the global `rotate` parameter.

Correcting this will need a big rewrite of the path routines, and although the authors are thinking about solving it, don't hold your breath; it will need changing a lot of interwoven code (labels, voltages, currents and so on). Contributions and help would be highly appreciated.

This same issue create a lot of problem of compatibility between CircuiTi*k*Z and the new `pic` Ti*k*Z feature, so basically don't put components into `pic`s.

1.8 Incompabilities between version

Here, we will provide a list of incompabilitys between different version of circuitikz. We will try to hold this list short, but sometimes it is easier to break with old syntax than including a lot of switches and compatibility layers. You can check the used version at your local installation using the macro `\pgfcircversion{}`.

- After v0.9.4: added the concept of styling of circuits. It should be backward compatible, but it's a big change, so be ready to use the `0.9.3` snapshot (see below for details).
- After v0.9.0: the parameters `tripoles/american or port/aaa`, `...bbb`, `...ccc` and `...ddd` are no longer used and are silently ignored; the same stands for `nor`, `xor`, and `xnor` ports.
- After v0.9.0: voltage and current directions/sign (plus and minus signs in case of `american voltages` and arrows in case of `european voltages` have been rationalized with a couple of new options (see details in section 4.2. The default case is still the same as v0.8.3.
- Since v0.8.2: voltage and current label directions(v<= / i<=) do NOT change the orientation of the drawn source shape anymore. Use the "invert" option to rotate the shape of the source. Furthermore, from this version on, the current label(i=) at current sources can be used independent of the regular label(l=).
- Since v0.7?: The label behaviour at mirrored bipoles has changes, this fixes the voltage drawing, but perhaps you have to adjust your label positions.

- Since v0.5.1: The parts pfet, pigfete, pigfetebulk and pigfetd are now mirrored by default. Please adjust your yscale-option to correct this.
- Since v0.5: New voltage counting direction, here exists an option to use the old behaviour

If you have older projects that show compatibility problems, you have two options:

- you can use an older version locally using the git-version and picking the correct commit from the repository (branch gh-pages) or the main GitHub site directly;
- if you are using LaTeX, the distribution has embedded several important old versions: `0.4`, `0.6`, `0.7`, `0.8.3` and `0.9.3`. To switch to use them, you simply change your `\usepackage` invocation like

```
\usepackage[]{circuitik-0.8.3} % or circuitikz-0.4, 0.6...
```

  You have to take care of the options that may have changed between versions;
- if you are using ConTeXt, only versions `0.8.3` and `0.9.3` are packaged for now; if can use it with

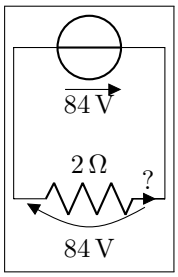
```
\usemodule[circuitik-0.8.3]
```

1.9 Feedback

The easiest way to contact the authors is via the official Github repository: https://github.com/circuitikz/circuitikz/issues. For general help question, a lot of nice people is quite active on https://tex.stackexchange.com/questions/tagged/circuitikz — be sure to read the help pages for the site and ask!

1.10 Package options

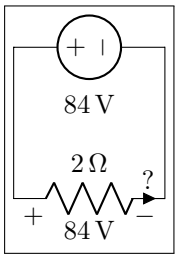
Circuit people are very opinionated about their symbols. In order to meet the individual gusto you can set a bunch of package options. The standard options are what the authors like, for example you get this:




```
\begin{circuitikz}
   \draw (0,0) to[R=2<\ohm>, i=?, v=84<\volt>] (2,0) --
   (2,2) to[V<=84<\volt>] (0,2)
   -- (0,0);
\end{circuitikz}
```

Feel free to load the package with your own cultural options:

LaTeX	ConTeXt
`\usepackage[american]{circuitikz}`	`\usemodule[circuitikz][american]`




```
\begin{circuitikz}
    \draw (0,0) to[R=2<\ohm>, i=?, v=84<\volt>] (2,0) --
    (2,2) to[V<=84<\volt>] (0,2)
    -- (0,0);
\end{circuitikz}
```

Here is the list of all the options:

- `europeanvoltages`: uses arrows to define voltages, and uses european-style voltage sources;
- `straightvoltages`: uses arrows to define voltages, and and uses straight voltage arrows;
- `americanvoltages`: uses − and + to define voltages, and uses american-style voltage sources;
- `europeancurrents`: uses european-style current sources;
- `americancurrents`: uses american-style current sources;
- `europeanresistors`: uses rectangular empty shape for resistors, as per european standards;
- `americanresistors`: uses zig-zag shape for resistors, as per american standards;
- `europeaninductors`: uses rectangular filled shape for inductors, as per european standards;
- `americaninductors`: uses "4-bumps" shape for inductors, as per american standards;
- `cuteinductors`: uses my personal favorite, "pig-tailed" shape for inductors;
- `americanports`: uses triangular logic ports, as per american standards;
- `europeanports`: uses rectangular logic ports, as per european standards;
- `americangfsurgearrester`: uses round gas filled surge arresters, as per american standards;
- `europeangfsurgearrester`: uses rectangular gas filled surge arresters, as per european standards;
- `european`: equivalent to `europeancurrents`, `europeanvoltages`, `europeanresistors`, `europeaninductors`, `europeanports`, `europeangfsurgearrester`;
- `american`: equivalent to `americancurrents`, `americanvoltages`, `americanresistors`, `americaninductors`, `americanports`, `americangfsurgearrester`;
- `siunitx`: integrates with `SIunitx` package. If labels, currents or voltages are of the form `#1<#2>` then what is shown is actually `\SI{#1}{#2}`;
- `nosiunitx`: labels are not interpreted as above;
- `fulldiode`: the various diodes are drawn *and* filled by default, i.e. when using styles such as `diode`, `D`, `sD`, …Other diode styles can always be forced with e.g. `Do`, `D-`, …
- `strokediode`: the various diodes are drawn *and* stroke by default, i.e. when using styles such as `diode`, `D`, `sD`, …Other diode styles can always be forced with e.g. `Do`, `D*`, …
- `emptydiode`: the various diodes are drawn *but not* filled by default, i.e. when using styles such as `D`, `sD`, …Other diode styles can always be forced with e.g. `Do`, `D-`, …
- `arrowmos`: pmos and nmos have arrows analogous to those of pnp and npn transistors;

- `noarrowmos`: pmos and nmos do not have arrows analogous to those of pnp and npn transistors;
- `fetbodydiode`: draw the body diode of a FET;
- `nofetbodydiode`: do not draw the body diode of a FET;
- `fetsolderdot`: draw solderdot at bulk-source junction of some transistors;
- `nofetsolderdot`: do not draw solderdot at bulk-source junction of some transistors;
- `emptypmoscircle`: the circle at the gate of a pmos transistor gets not filled;
- `lazymos`: draws lazy nmos and pmos transistors. Chip designers with huge circuits prefer this notation;
- `straightlabels`: labels on bipoles are always printed straight up, i.e. with horizontal baseline;
- `rotatelabels`: labels on bipoles are always printed aligned along the bipole;
- `smartlabels`: labels on bipoles are rotated along the bipoles, unless the rotation is very close to multiples of 90°;
- `compatibility`: makes it possibile to load CircuiTi*k*Z and Ti*k*Z circuit library together.
- Voltage directions: until v0.8.3, there was an error in the coherence between american and european voltages styles (see section 4.2) for the batteries. This has been fixed, but to guarantee backward compatibility and to avoid nasty surprises, the fix is available with new options:
  - `oldvoltagedirection`: Use old way of voltage direction having a difference between european and american direction, with wrong default labelling for batteries;
  - `nooldvoltagedirection`: The standard from 0.5 onward, utilize the (German?) standard of voltage arrows in the direction of electric fields (without fixing batteries);
  - `RPvoltages` (meaning Rising Potential voltages): the arrow is in direction of rising potential, like in `oldvoltagedirection`, but batteries and current sources are fixed to follow the passive/active standard;
  - `EFvoltages` (meaning Electric Field voltages): the arrow is in direction of the electric field, like in `nooldvoltagedirection`, but batteries are fixed;

  If none of these option are given, the package will default to `nooldvoltagedirection`, but will give a warning. The behavior is also selectable circuit by circuit with the `voltage dir` style.
- `betterproportions`[2]: nicer proportions of transistors in comparision to resistors;

The old options in the singular (like `american voltage`) are still available for compatibility, but are discouraged.

Loading the package with no options is equivalent to the following options: `[nofetsolderdot, europeancurrents, europeanvoltages, americanports, americanresistors, cuteinductors, europeangfsurgearrester, nosiunitx, noarrowmos, smartlabels, nocompatibility]`.

In ConTEXt the options are similarly specified: `current= european|american`, `voltage= european|american`, `resistor= american|european`, `inductor= cute|american|european`, `logic= american|european`, `siunitx= true|false`, `arrowmos= false|true`.

[2] May change in the future!

2 Tutorials

To draw a circuit, you have to load the `circuitikz` package; this can be done with

```
\usepackage[siunitx, RPvoltages]{circuitikz}
```

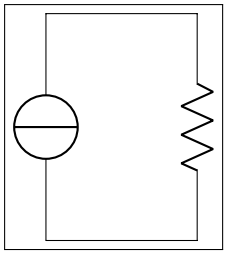
somewhere in your document preamble. It will load automatically the needed packages if not already done before.

2.1 Getting started with CircuiTi*k*Z: a current shunt

Let's say we want to prepare a circuit to teach how a current shunt works; the idea is to draw a current generator, a couple of resistors in parallel, and the indication of currents and voltages for the discussion.

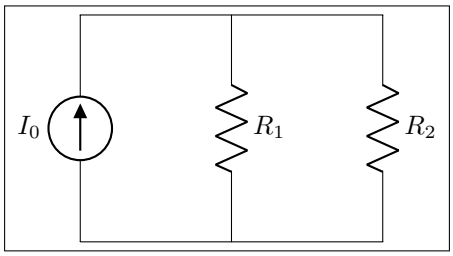
A circuit in CircuiTi*k*Zis drawn into a `circuitikz` environment (which is really an alias for `tikzpicture`). In this first example we will use absolute coordinates. The electrical components can be divided in two main categories: the one that are bipoles and are placed along a path (also known as `to`-style component, for their usage), and components that are nodes and can have any number of poles or connections.

Let's start with the first type of component, and build a basic mesh:



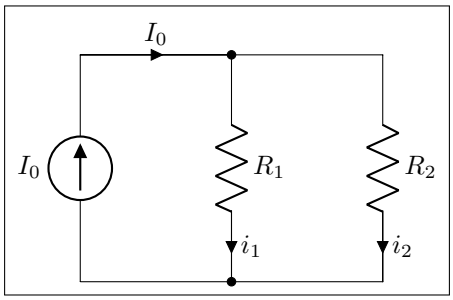
```
\begin{circuitikz}[]
    \draw (0,0) to[isource] (0,3) -- (2,3)
    to[R] (2,0) -- (0,0);
\end{circuitikz}
```

The symbol for the current source can surprise somebody; this is actually the european-style symbol, and the type of symbol chosen reflects the default options of the package (see section 1.10). Let's change the style for now (the author of the tutorial, Romano, is European - but he has always used American-style circuits, so …); and while we're at it, let's add the other branch and some labels.



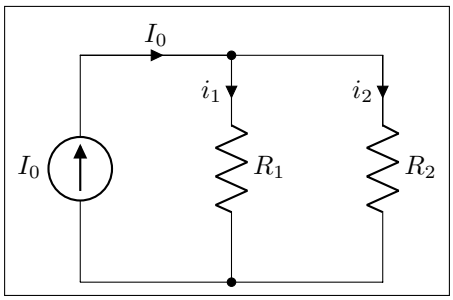

```
\begin{circuitikz}[american]
    \draw (0,0) to[isource, l=$I_0$] (0,3) --
        (2,3)
    to[R=$R_1$] (2,0) -- (0,0);
    \draw (2,3) -- (4,3) to[R=$R_2$]
        (4,0) -- (2,0);
\end{circuitikz}
```

You can use a single path or multiple path when drawing your circuit, it's just a question of style (but be aware that closing path could be non-trivial, see section 4.11), and you can use standard Ti*k*Z lines (`--`, `|-` or similar) for the wires. Nonetheless, sometime using the CircuiTi*k*Z specific `short` component for the wires can be useful, because then we can add labels and nodes at it, like for example in the following circuit.



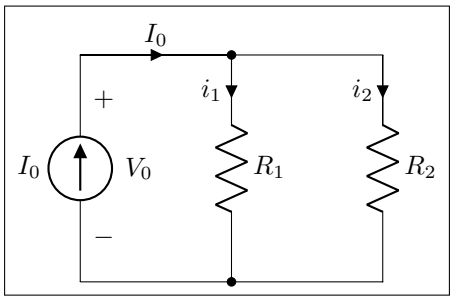

```
\begin{circuitikz}[american]
    \draw (0,0) to[isource, l=$I_0$] (0,3)
    to[short, -*, i=$I_0$] (2,3)
    to[R=$R_1$, i=$i_1$] (2,0) -- (0,0);
    \draw (2,3) -- (4,3)
    to[R=$R_2$, i=$i_2$]
    (4,0) to[short, -*] (2,0);
\end{circuitikz}
```

One of the problems with this circuit is that we would like to have the current in a different position, such as for example on the upper side of the resistors, so that Kirchoff's Current Law at the node is better shown to students. No problem; as you can see in section 4.2 you can use the position specifier `<>^_}` after the key `i`:



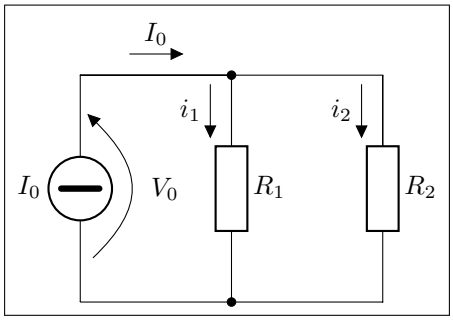

```
\begin{circuitikz}[american]
    \draw (0,0) to[isource, l=$I_0$] (0,3)
    to[short, -*, i=$I_0$] (2,3)
    to[R=$R_1$, i>_=$i_1$] (2,0) -- (0,0);
    \draw (2,3) -- (4,3)
    to[R=$R_2$, i>_=$i_2$]
    (4,0) to[short, -*] (2,0);
\end{circuitikz}
```

Finally, we would like to add voltages indication for carrying out the current formulas; as the default position of the voltage signs seems a bit cramped to me, I am adding the `voltage shift` parameter to make a bit more space for it…



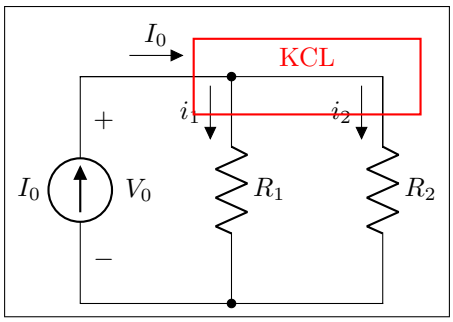

```
\begin{circuitikz}[american, voltage shift
    =0.5]
    \draw (0,0) to[isource, l=$I_0$, v=$V_0$]
        (0,3)
    to[short, -*, i=$I_0$] (2,3)
    to[R=$R_1$, i>_=$i_1$] (2,0) -- (0,0);
    \draw (2,3) -- (4,3)
    to[R=$R_2$, i>_=$i_2$]
    (4,0) to[short, -*] (2,0);
\end{circuitikz}
```

Et voilá!. Remember that this is still LaTeX, which means that you have done a description of your circuit, which is, in a lot of way, independent of the visualization of it. If you ever have to adapt the circuit to, say, a journal that force European style and flows instead of currents, you just change a couple of things and you have what seems a completely different diagram:




```
\begin{circuitikz}[european, voltage shift
    =0.5]
    \draw (0,0) to[isourceC, l=$I_0$, v=$V
        _0$] (0,3)
    to[short, -*, f=$I_0$] (2,3)
    to[R=$R_1$, f>_=$i_1$] (2,0) -- (0,0);
    \draw (2,3) -- (4,3)
    to[R=$R_2$, f>_=$i_2$]
    (4,0) to[short, -*] (2,0);
\end{circuitikz}
```

And finally, this is still Ti*k*Z, so that you can freely mix other graphics element to the circuit.




```
\begin{circuitikz}[american, voltage shift
    =0.5]
   \draw (0,0) to[isource, l=$I_0$, v=$V_0$]
        (0,3)
   to[short, -*, f=$I_0$] (2,3)
   to[R=$R_1$, f>_=$i_1$] (2,0) -- (0,0);
   \draw (2,3) -- (4,3)
   to[R=$R_2$, f>_=$i_2$]
   (4,0) to[short, -*] (2,0);
   \draw[red, thick] (1.5,2.5) rectangle
       (4.5,3.5)
   node[pos=0.5, above]{KCL};
\end{circuitikz}
```

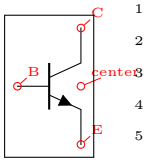
2.2 A more complex tutorial: circuits, Romano style.

The idea is to draw a two-stage amplifier for a lesson, or exercise, on the different qualities of BJT and MOSFET transistors. Notice that this is a more "personal" tutorial, showing a way to draw circuits that is, in the author's opinion, highly reusable and easy to do. The idea is using relative coordinates and named nodes as much as possible, so that changes in the circuit are easily done by changing keys numbers of position, and crucially, each block is reusable in other diagrams.

First of all, let's define a handy function to show the position of nodes:

```
\def\coord(#1){coordinate(#1)}
\def\coord(#1){node[circle, red, draw, inner sep=1pt,pin={[red, overlay, inner
    sep=0.5pt, font=\tiny, pin distance=0.1cm, pin edge={red, overlay
   ,}]45:#1}](#1){}}
```

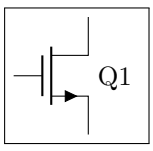
The idea is that you can use `\coord()` instead of `coordinate()` in paths, and that will draw sort of *markers* showing them. For example:




```
\begin{circuitikz}[american,]
   \draw (0,0) node[npn](Q){};
   \path (Q.center) \coord(center)
   (Q.B) \coord(B) (Q.C) \coord(C)
   (Q.E) \coord(E);
\end{circuitikz}
```

After the circuit is drawn, simply commenting out the second definition of `\coord` will hide all the markers.

So let's start with the first stage transistor; given that my preferred way of drawing a MOSFET is with arrows, I'll start with the command `\ctikzset{tripoles/mos style/arrows}`:

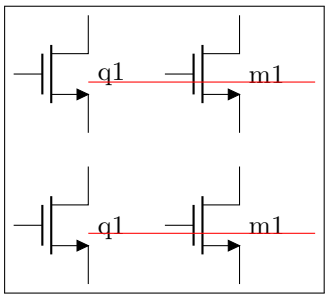



```
\begin{circuitikz}[american,]
\ctikzset{tripoles/mos style/arrows}
\def\killdepth#1{{\raisebox{0pt}[\height][0pt]{#1}}}
   \draw (0,0) node[nmos](Q1){};
   \draw (Q1.center) node[right]{\killdepth{Q1}};
\end{circuitikz}
```

Another thing I like to modify with respect to the standard is the position of the arrows in transistors, which are normally in the middle the symbol. Using the following setting (see section 3.16.3) will move the arrows to the start or end of the corresponding pin.

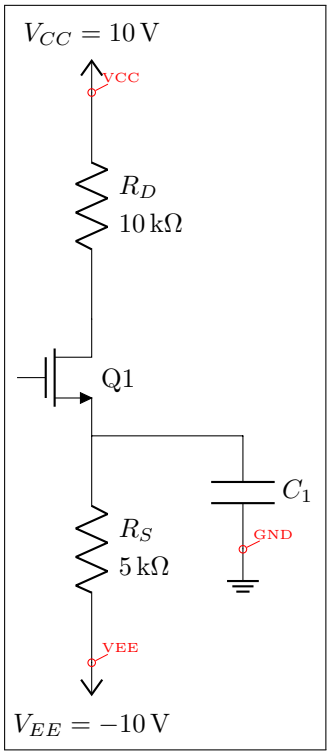
```
\ctikzset{transistors/arrow pos=end}
```

The tricky thing about `\killdepth{}` macro is finnicky details; I do not like the standard position of labels on transistors (which is near the collector/drain) so I plot the label at the right of the `center` anchor. Without the `\killdepth` macro, the labels of different transistor will be adjusted so that the center of the box is at the `center` anchor, and as an effect, labels with descenders (like Q) will have a different baseline than labels without. You can see this here (it's really subtle):



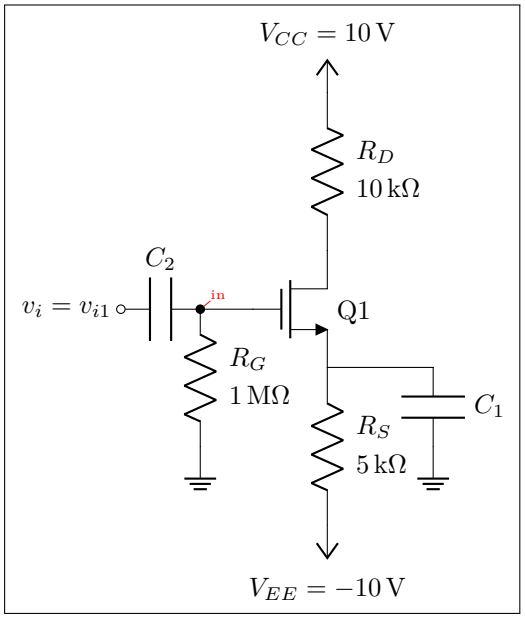

```
\begin{circuitikz}[american,]
\draw (0,0) node[nmos](Q1){} ++(2,0) node[nmos](M1){};
\draw (Q1.center) node[right]{q1};
\draw (M1.center) node[right]{m1};
\draw [red] (Q1.center) ++(0,-0.7ex) -- ++(3,0);
\draw (0,-2)node[nmos](Q1){} ++(2,0) node[nmos](M1){};
\draw (Q1.center) node[right]{\killdepth{q1}};
\draw (M1.center) node[right]{\killdepth{m1}};
\draw [red] (Q1.center) ++(0,-0.7ex) -- ++(3,0);
\end{circuitikz}
```

We will start connecting the first transistor with the power supply with a couple of resistors. Notice that I am naming the nodes `GND`, `VCC` and `VEE`, so that I can use the coordinates to have all the supply rails at the same vertical position (more on this later).




```
\begin{circuitikz}[american,]
    \draw (0,0) node[nmos,](Q1){};
    \draw (Q1.center) node[right]{\killdepth{Q1}};
    \draw (Q1.S) to[R, l2^=$R_S$ and \SI{5}{k\ohm}]
        ++(0,-3)
        node[vee](VEE){$V_{EE}=\SI{-10}{V}$};
    \draw (Q1.D) to[R, l2_=$R_D$ and \SI{10}{k\ohm}]
        ++(0,3)
        node[vcc](VCC){$V_{CC}=\SI{10}{V}$};
    \draw (Q1.S) to[short] ++(2,0) to[C=$C_1$]
        ++(0,-1.5) node[ground](GND){};
    \path (GND) \coord(GND) (VCC) \coord(VCC)
        (VEE) \coord(VEE);
\end{circuitikz}
```

After that, let's add the input part. I will use a named node here, to refer to it to add the input source. Notice how the ground node is positioned: the coordinate `(in |- GND)` is the point with the horizontal coordinate of `(in)` and the horizontal one of `(GND)`, lining it up with the ground of the capacitor C_1.



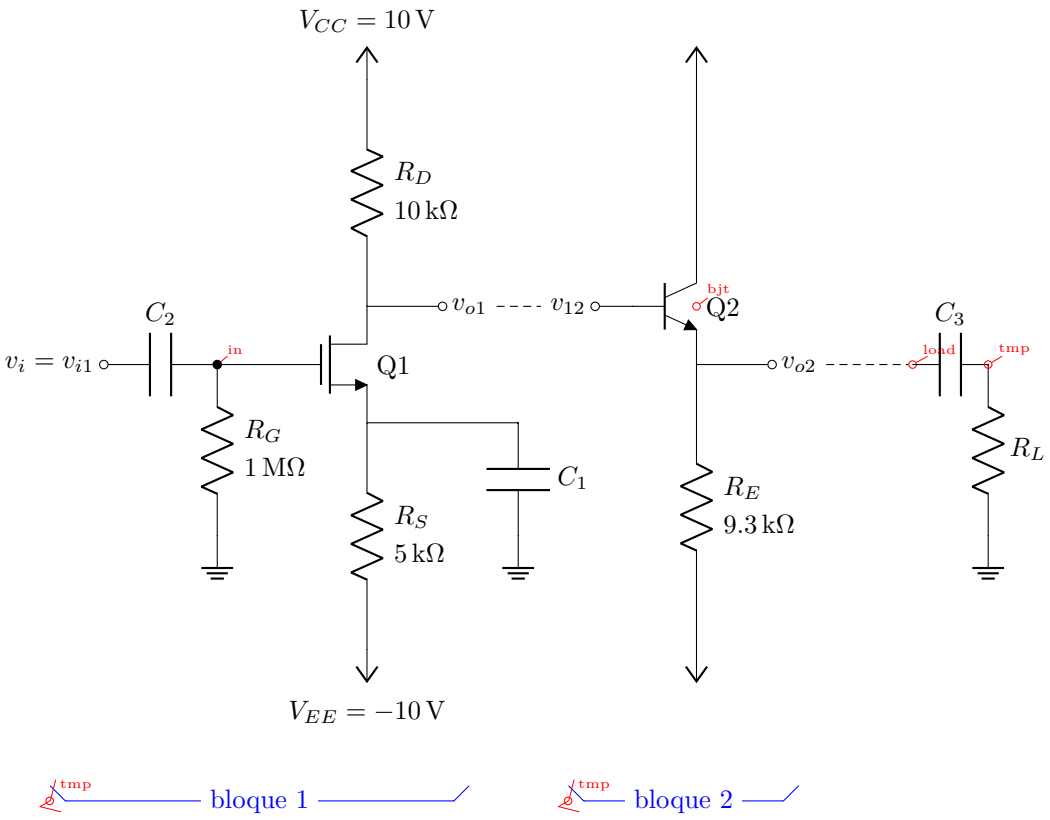

```
\begin{circuitikz}[american, scale=0.7]
    \draw (0,0) node[nmos,](Q1){};
    \draw (Q1.center) node[right]
        {\killdepth{Q1}};
    \draw (Q1.S) to[R, l2^=$R_S$ and \SI
        {5}{k\ohm}] ++(0,-3)
        node[vee](VEE){$V_{EE}=\SI{-10}{V
            }$};
    \draw (Q1.D) to[R, l2_=$R_D$ and \SI
        {10}{k\ohm}] ++(0,3)
        node[vcc](VCC){$V_{CC}=\SI{10}{V
            }$};
    \draw (Q1.S) to[short] ++(2,0) to[C
        =$C_1$] ++(0,-1.5) node[ground](
        GND){};
    \draw (Q1.G) to[short] ++(-1,0)
        \coord (in) to[R, l2^=$R_G$ and \
            SI{1}{M\ohm}]
        (in |- GND) node[ground]{};
    \draw (in) to[C, l_=$C_2$,*-o]
        ++(-1.5,0) node[left](vi1){$v_i=
        v_{i1}$};
\end{circuitikz}
```

Notice that the only absolute coordinate here is the first one, `(0,0)`; so the elements are connected with relative movements and can be moved by just changing one number (for example, changing the `to[C=$C_1$] ++(0,-1.5)` will move *all* the grounds down).

This is the final circuit, with the nodes still marked:

```
\tikzset{blockdef/.style={%
    {Straight Barb[harpoon, reversed, right, length=0.2cm]}-{Straight Barb[
        harpoon, reversed, left, length=0.2cm]},
    blue, %densely dotted,
}}
\def\killdepth#1{{\raisebox{0pt}[\height][0pt]{#1}}}
\def\coord(#1){coordinate(#1)}
\def\coord(#1){node[circle, red, draw, inner sep=1pt,pin={[red, overlay, inner
     sep=0.5pt, font=\tiny, pin distance=0.1cm, pin edge={red, overlay
    ,}]45:#1}](#1){}}
\begin{circuitikz}[american, ]
    \draw (0,0) node[nmos,](Q1){};
    \draw (Q1.center) node[right]{\killdepth{Q1}};
    \draw (Q1.S) to[R, l2^=$R_S$ and \SI{5}{k\ohm}] ++(0,-3) node[vee](VEE){$V
        _{EE}=\SI{-10}{V}$}; %define VEE level
    \draw (Q1.S) to[short] ++(2,0) to[C=$C_1$] ++(0,-1.5) node[ground](GND){};
    \draw (Q1.G) to[short] ++(-1,0) \coord (in) to[R, l2^=$R_G$ and \SI{1}{M\
        ohm}] (in |- GND) node[ground]{};
    \draw (in) to[C, l_=$C_2$,*-o] ++(-1.5,0) node[left](vi1){$v_i=v_{i1}$};
    \draw (Q1.D) to[R, l2_=$R_D$ and \SI{10}{k\ohm}] ++(0,3) node[vcc](VCC){$V
        _{CC}=\SI{10}{V}$};
    \draw (Q1.D) to[short, -o] ++(1,0) node[right](vo1){$v_{o1}$};
    %
    \path (vo1) -- ++(3,0) \coord(bjt);
    %
```

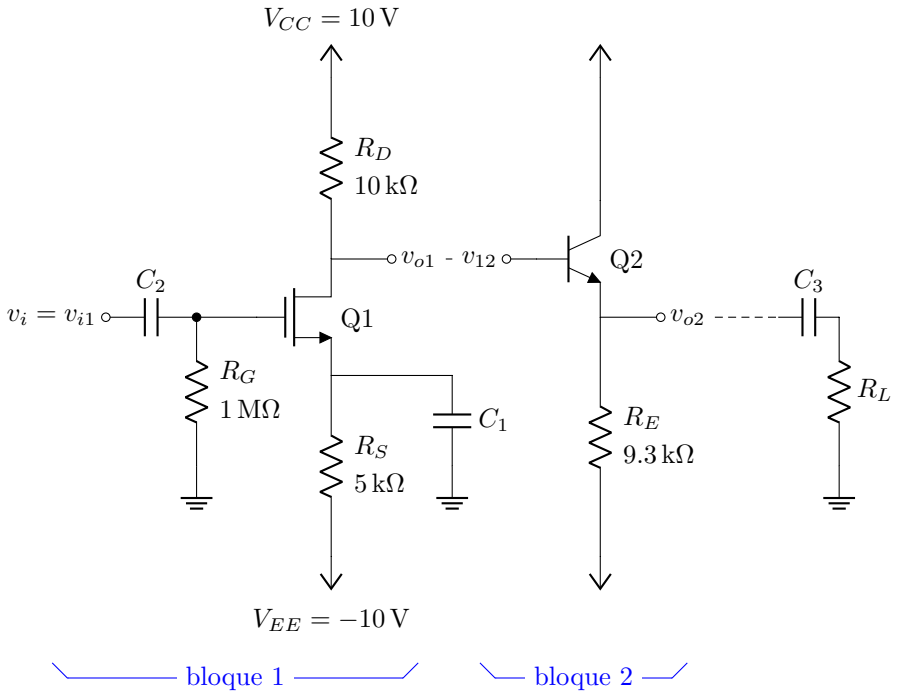
```
    \draw (bjt) node[npn, ](Q2){};
    \draw (Q2.center) node[right]{\killdepth{Q2}};
    \draw (Q2.B) to[short, -o] ++(-0.5,0) node[left](vi2){$v_{12}$};
    \draw (Q2.E) to[R,l2^=$R_E$ and \SI{9.3}{k\ohm}] (Q2.E |- VEE) node[vee]{};
    \draw (Q2.E) to[short, -o] ++(1,0) node[right](vo2){$v_{o2}$};
    \draw (Q2.C) to[short] (Q2.C |- VCC) node[vcc]{};
    %
    \path (vo2) ++(1.5,0) \coord(load);
    \draw (load) to[C=$C_3$] ++(1,0) \coord(tmp) to[R=$R_L$] (tmp |- GND) node[
        ground]{};
    \draw [densely dashed] (vo2) -- (load);
    %
    \draw [densely dashed] (vo1) -- (vi2);
    %
    \draw [blockdef](vi1|-VEE) ++(0,-2) \coord(tmp)
         -- node[midway, fill=white]{bloque 1} (vo1|- tmp);
         \draw [blockdef] (vi2|-VEE) ++(0,-2) \coord(tmp)
         -- node[midway, fill=white]{bloque 2} (vo2|- tmp);

\end{circuitikz}
```



Finally, if you like a more compact drawing, you can add the options (for example):

```
\begin{circuitikz}[american, scale=0.8] % this will scale only the coordinates
    \ctikzset{resistors/scale=0.7, capacitors/scale=0.6}
    ...
\end{circuitikz}
```

and you will obtain the following diagram with the exact same code (I just removed the second `\coord` definition to hide the coordinates markings).



3 The components

Components in CircuiTi*k*Z come in two forms: path-style, to be used in `to` path specifications, and node-style, which will be instantiated by a `node` specification.

3.1 Path-style components

The path-style components are used as shown below:

```
\begin{circuitikz}
\draw (0,0) to[#1=#2, #options] (2,0);
\end{circuitikz}
```

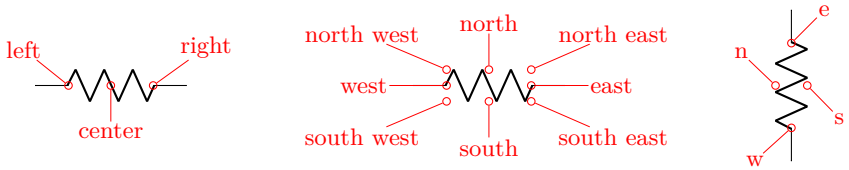
where `#1` is the name of the component, `#2` is an (optional) label, and `options` are optional labels, annotations, style specifier that will be explained in the rest of the manual.

Transistors and some other node-style components can also be placed using the syntax for bipoles. See section 3.16.2.

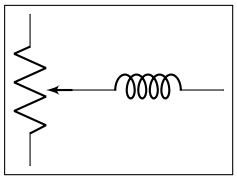
Most path-style components can be used as a node-style components; to access them, you add a `shape` to the main name of component (for example, `diodeshape`). Such a "node name" is specified in the description of each component.

3.1.1 Anchors

Normally, path-style components do not need anchors, although they have them just in case you need them. You have the basic "geographical" anchors (bipoles are defined horizontally and then rotated as needed):

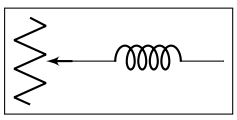


In the case of bipoles, also shortened geographical anchors exists. In the description, it will be shown when a bipole has additional anchors. To use the anchors, just give a name to the bipole element.



```
\begin{circuitikz}
    \draw (0,0) to[potentiometer, name=P, mirror] ++(0,2);
    \draw (P.wiper) to[L] ++(2,0);
\end{circuitikz}
```

Alternatively, that you can use the shape form, and then use the `left` and `right` anchors to do your connections.



```
\begin{circuitikz}
    \draw (0,0) node[potentiometershape, rotate=-90](P){};
    \draw (P.wiper) to[L] ++(2,0);
\end{circuitikz}
```

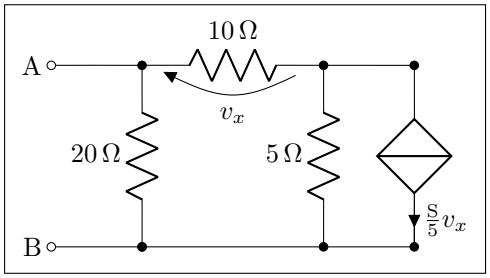
3.1.2 Customization

Pretty much all CircuiTi*k*Z relies heavily on `pgfkeys` for value handling and configuration. Indeed, at the beginning of `circuitikz.sty` and in the file `pfgcirc.define.tex` a series of key definitions can be found that modify all the graphical characteristics of the package.

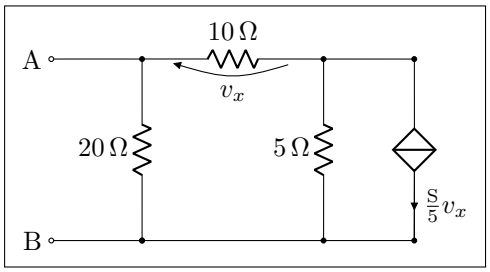
All can be varied using the `\ctikzset` command, anywhere in the code.

Note that the details of the parameters that are not described in the manual can change in the future, so be ready to use a fixed version of the package (the ones with the specific number, like `circuitikz-0.9.3`) if you dig into them.

3.1.2.1 Components size Perhaps the most important parameter is `bipoles/length` (default 1.4 cm), which can be interpreted as the length of a resistor (including reasonable connections): all other lengths are relative to this value. For instance:

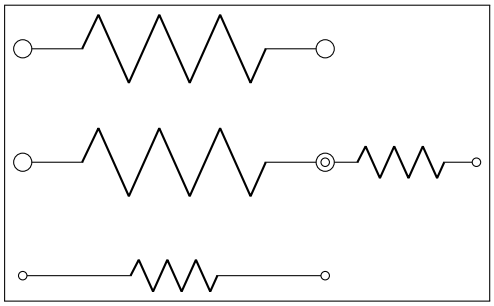



```
\ctikzset{bipoles/length=1.4cm}
\begin{circuitikz}[scale=1.2]\draw
  (0,0) node[anchor=east] {B}
        to[short, o-*] (1,0)
        to[R=20<\ohm>, *-*] (1,2)
        to[R=10<\ohm>, v=$v_x$] (3,2) -- (4,2)
        to[cI=$\frac{\si{\siemens}}{5} v_x$, *-*] (4,0) -- (3,0)
        to[R=5<\ohm>, *-*] (3,2)
  (3,0) -- (1,0)
  (1,2) to[short, -o] (0,2) node[anchor=east]{A}
;\end{circuitikz}
```



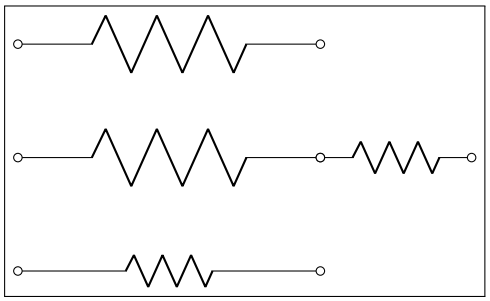

```
\ctikzset{bipoles/length=.8cm}
\begin{circuitikz}[scale=1.2]\draw
  (0,0) node[anchor=east] {B}
        to[short, o-*] (1,0)
        to[R=20<\ohm>, *-*] (1,2)
        to[R=10<\ohm>, v=$v_x$] (3,2) -- (4,2)
        to[cI=$\frac{\siemens}{5} v_x$, *-*] (4,0) -- (3,0)
        to[R=5<\ohm>, *-*] (3,2)
  (3,0) -- (1,0)
  (1,2) to[short, -o] (0,2) node[anchor=east]{A}
;\end{circuitikz}
```

The changes on `bipoles/length` should, however, be globally applied to every path, because they affect every element — including the poles. So you can have artifacts like these:



```
\begin{circuitikz}[
    bigR/.style={R, bipoles/length=3cm}
    ]
    \draw (0,3) to [bigR, o-o] ++(4,0);
    \draw (0,1.5) to [bigR, o-o] ++(4,0)
      to[R, o-o] ++(2,0); % will fail here
    \draw (0,0) to [R, o-o] ++(4,0);
\end{circuitikz}
```

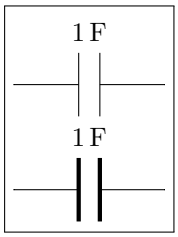
Several groups of components, on the other hand, have a special `scale` parameter that can be used safely in this case (starting with 0.9.4 — more groups of components will be added going forward); the key to use will be explained in the specific description of the components. For example, in the case of resistors you have `resistors/scale` available:



```
\begin{circuitikz}[
    bigR/.style={R, resistors/scale=1.8}
    ]
    \draw (0,3) to [bigR, o-o] ++(4,0);
    \draw (0,1.5) to [bigR, o-o] ++(4,0)
      to[R, o-o] ++(2,0); % ok now
    \draw (0,0) to [R, o-o] ++(4,0);
\end{circuitikz}
```

3.1.2.2 Thickness of the lines (globally)

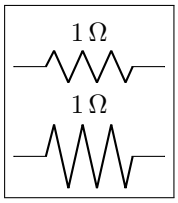
The best way to alter the thickness of components is using styling, see section 3.3.3. Alternatively, you can use "legacy" classes like `bipole`, `tripoles` and so on — for example changing the parameter `bipoles/thickness` (default 2). The number is relative to the thickness of the normal lines leading to the component.




```
\ctikzset{bipoles/thickness=1}
\tikz \draw (0,0) to[C=1<\farad>] (2,0); \par
\ctikzset{bipoles/thickness=4}
\tikz \draw (0,0) to[C=1<\farad>] (2,0);
```

3.1.2.3 Shape of the components (on a per-component-class basis)

The shape of the components are adjustable with a lot of parameters; in this manual we will comment the main ones, but you can look into the source files specified above to find more.

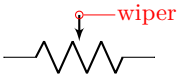



```
\tikz \draw (0,0) to[R=1<\ohm>] (2,0); \par
\ctikzset{bipoles/resistor/height=.6}
\tikz \draw (0,0) to[R=1<\ohm>] (2,0);
```

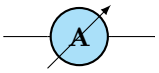
It is recommended to use the styling parameters to change the shapes; they are not so fine grained (for example, you can change the width of resistor, not the height at the moment), but they are more stable and coherent across your circuit.

3.1.3 Descriptions

The typical entry in the component list will be like this:

	resistor: resistor, american style, `type: path-style`, `nodename: resistorshape`. Aliases: `R`, `american resistor`. Class: `resistors`.
	**pR**: potentiometer, american style, `type: path-style`, `nodename: potentiometershape`. Aliases: `pR`, `american potentiometer`. Class: `resistors`.

where you have all the needed information about the bipole, with also no-standard anchors. If the component can be filled it will be specified in the description. In addition, as an example, the component shown will be filled with the option `fill=cyan!30!white`:

	**ammeter**: Ammeter, `type: path-style, fillable`, `nodename: ammetershape`. Class: `instruments`.

The *Class* of the component (see section 3.3) is printed at the end of the description.

3.2 Node-style components

Node-style components (monopoles, multipoles) can be drawn at a specified point with this syntax, where `#1` is the name of the component:

```
\begin{circuitikz}
   \draw (0,0) node[#1,#2] (#3) {#4};
\end{circuitikz}
```

Explanation of the parameters:
`#1`: component name[3] (mandatory)
`#2`: list of comma separated options (optional)
`#3`: name of an anchor (optional)
`#4`: text written to the text anchor of the component (optional)

Most path-style components can be used as a node-style components; to access them, you add a `shape` to the main name of component (for example, `diodeshape`). Such a "node name" is specified in the description of each component.

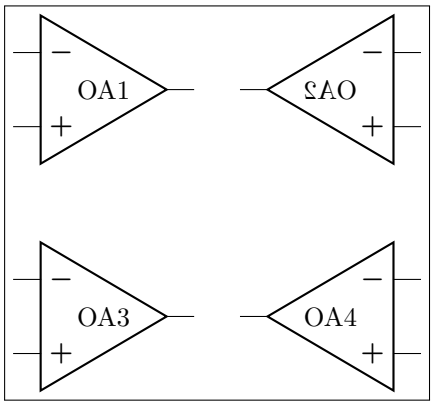
Notice: Nodes must have curly brackets at the end, even when empty. An optional anchor (`#3`) can be defined within round brackets to be addressed again later on. And please don't forget the semicolon to terminate the `\draw` command.

Also notice: If using the `\tikzexternalize` feature, as of Ti*k*z 2.1 all pictures must end with `\end{tikzpicture}`. Thus you *cannot* use the `circuitikz` environment.

Which is ok: just use the environment `tikzpicture`: everything will work there just fine.

3.2.1 Mirroring and flipping

Mirroring and flipping of node components is obtained by using the Ti*k*Z keys `xscale` and `yscale`. Notice that this parameters affect also text labels, so they need to be un-scaled by hand.




```
\begin{circuitikz}
   \draw (0,3) node[op amp]{OA1};
   \draw (3,3) node[op amp, xscale=-1]{OA2};
   \draw (0,0) node[op amp]{OA3};
   \draw (3,0) node[op amp, xscale=-1]{%
      \scalebox{-1}[1]{OA4}};
\end{circuitikz}
```

3.2.2 Anchors

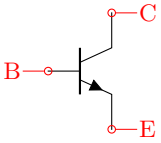
Node components anchors are variable across the various kind of components, so they will described better after each category is presented in the manual.

[3] For using bipoles as nodes, the name of the node is `#1shape`.

3.2.3 Descriptions

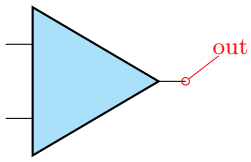
The typical entry in the component list will be like this:

Cute spdt down with arrow, type: node (`node[cute spdt down arrow]{}`). Class: `switches`.



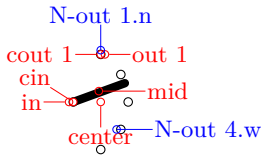
NPN, TYPE: NODE (`node[npn]{}`). Class: `transistors`.

All the shapes defined by CircuiTi*k*Z. These are all `pgf` nodes, so they are usable in both `pgf` and Ti*k*Z. If the component can be filled it will be specified in the description. In addition, as an example, the component shown will be filled with the option `fill=cyan!30!white`:



Plain amplifier, type: node, fillable (`node[plain amp]{}`). Class: `amplifiers`.

Sometime, components will expose internal (sub-)shapes that can be accessed with the syntax *<node name>-<internal node name>* (a dash is separating the node name and the internal node name); that will be shown in the description as a blue "anchor":



Rotary switch, type: node (`node[rotaryswitch](N){}`). Class: `switches`.

The *Class* of the component (see section 3.3) is printed at the end of the description.

3.3 Styling circuits and components

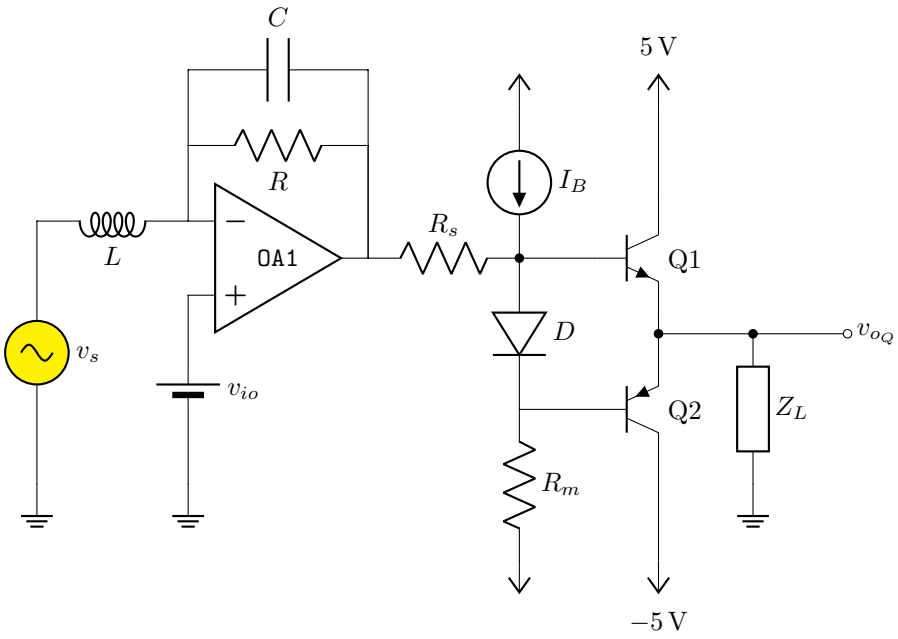
You can change the visual appearance of a circuit by using a circuit style different from the default. For styling the circuit, the concept of *class* of a component is key: almost every component has a class, and a style change will affect all the components of that class.

Let's see the effect over a simple circuit[4].

[4] This is a just an example, the circuit is not intended to be functional.

```
\def\killdepth#1{{\raisebox{0pt}[\height][0pt]{#1}}}
\newcommand\bjtname[1]{($(#1.C)!0.5!(#1.E)$) node[anchor=west]{\killdepth{#1}} }
\begin{circuitikz}[american, cute inductors]
    \node [op amp](A1){\texttt{OA1}};
    \draw (A1.-) to[short] ++(0,1) coordinate(tmp) to[R, l_=$R$] (tmp -| A1.out) to[short] (A1.out);
    \draw (tmp) to[short] ++(0,1) coordinate(tmp) to[C=$C$] (tmp -| A1.out) to[short] (A1.out);
    \draw (A1.+) to [battery2, invert] ++(0,-2.5) node[ground](GND){};
    \draw (A1.-) to [L=$L$] ++(-2,0) coordinate(tmp) to[sV, l=$v_s$, fill=yellow] (tmp |-GND) node[ground]{};
    \draw (A1.out) to[R=$R_s$] ++(2,0) coordinate(bb) to[I, l_=$I_B$, invert] ++(0,2) node[vcc](VCC){};
    \draw (bb) to[D, l=$D$, *-] ++(0,-2) coordinate(bb1) to[R=$R_m$] ++(0,-2) node[vee](VEE){};
    \draw (bb) --++(1,0) node[npn, anchor=B](Q1){} \bjtname{Q1};
    \draw (bb1) --++(1,0) node[pnp, anchor=B](Q2){} \bjtname{Q2};
    \draw (Q1.E) -- (Q2.E) ($(Q1.E)!0.5!(Q2.E)$) to [short, *-o, name=S] ++(2.5,0)
    node[right]{$v_{o_Q}$};
    \draw (S.s) to[european resistor, l=$Z_L$, *-] (S.s|-GND) node[ground]{};
    \draw (Q1.C) -- (Q1.C|-VCC) node[vcc]{\SI{5}{V}};
    \draw (Q2.C) -- (Q2.C|-VEE) node[vee]{\SI{-5}{V}};
\end{circuitikz}
```

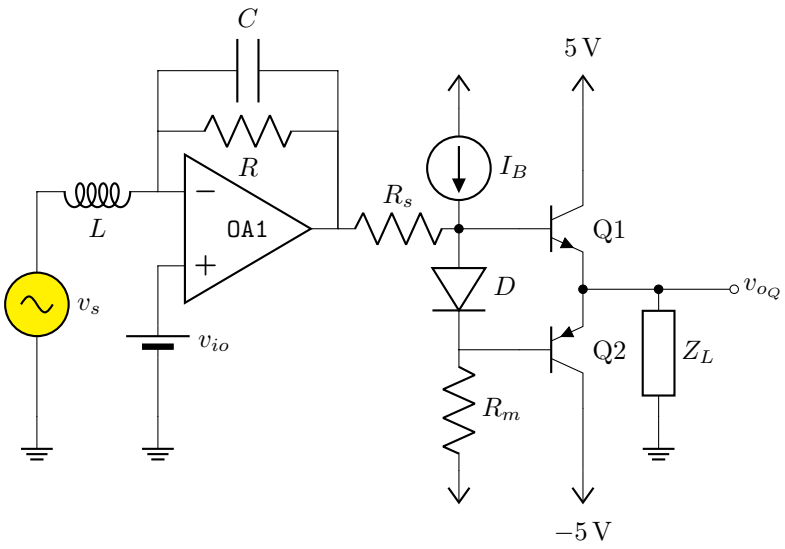
This code, with the default parameters, will render like the following image.



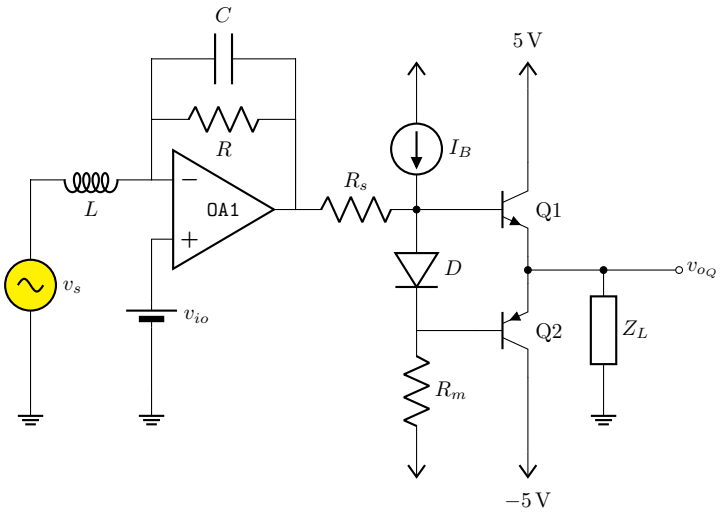
3.3.1 Relative size

Component size can be changed globally (see section 3.1.2.1), or you can change their relative size by scaling a family of components by setting the key *class*`/scale`; for example, you can change the size of all the diodes in your circuit by setting `diodes/scale` to something different from the default `1.0`.

Remember that if you use a global scale (be sure to read section 1.7!) you change the coordinate only, so using `scale=0.8` in the environment options you have:



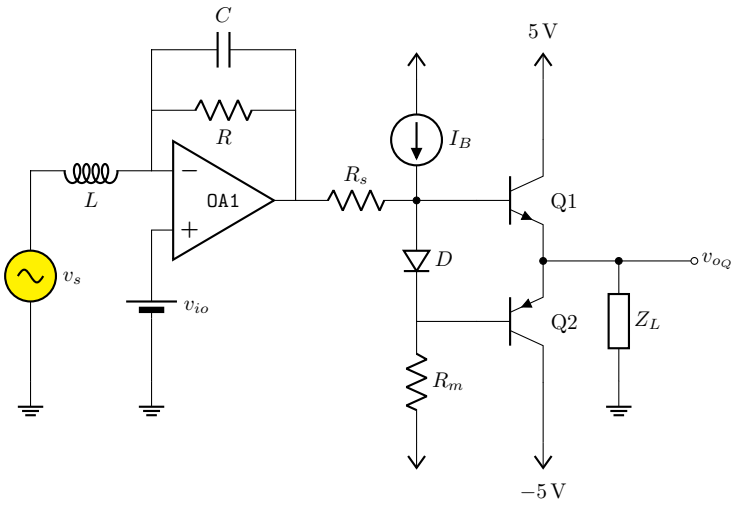
If you want to scale all the circuit, you have to use also `transform shape`:



Using relative sizes as described in section 3.1.2.1 enables your style for the circuit. For example, setting:

```
\ctikzset{resistors/scale=0.8, % smaller R
    capacitors/scale=0.7,      % even smaller C
    diodes/scale=0.6,          % small diodes
    transistors/scale=1.3}     % bigger BJTs
```

Will result in a (much more readable in Romano's opinion) circuit:



Warning: relative scaling is meant to work for a reasonable range of stretching and shortening, so try to keep your scale parameter in the `0.5` to `2.0` range (more or less). Bigger or smaller value can result in awkward shapes.

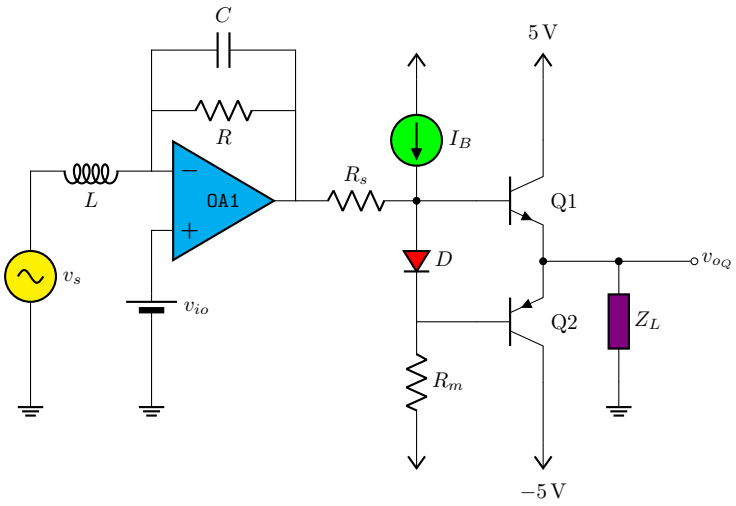
3.3.2 Fill color

You can also set a default fill color for the components. You can use the keys *`class`*`/fill` (which defaults to `none`, no fill, i.e. transparent component) for all fillable components in the library.

If you add to the previous styles the following commands:

```
\ctikzset{
    amplifiers/fill=cyan,
    sources/fill=green,
    diodes/fill=red,
    resistors/fill=violet,
}
```

you will have the following circuit (note that the first generator is *explicitly* set to be yellow, so if will not be colored green!):



Please use this option with caution. Although two-color circuits can be nice, using more than that can become rapidly unbearable. Old textbooks used the two-color style quite extensively, filling

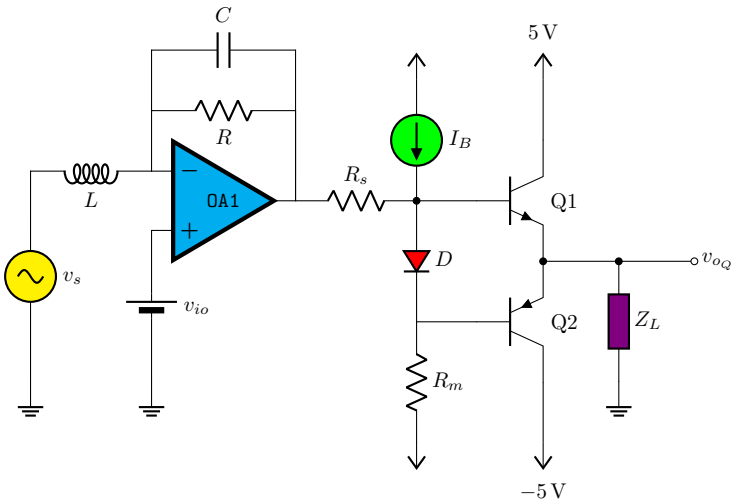
with a kind of light blue like `blue!30!white` "closed" components, but that was largely to hinder black-and-white photocopying…

3.3.3 Line thickness

You can change the line thickness for any class of component in an independent way. The default standard thickness of components is defined on a loose "legacy" category (like `bipoles`, `tripoles` and so on, see section 3.1.2.2); to override that you set the key *`class`*`/thickness` to any number. The default is `none`, which means that the old way of selecting thickness is used.

For example, *amplifiers* have the legacy class of `tripoles`, as well as transistors and tubes.By default they are drawn with thickness 2 (relative to the base linewidth). To change them to be thicker, you can for example add to the previous style

```
\ctikzset{amplifier/thickness=4}
```



Caveat: not every component has a "class", so you have to play with the available ones (it's specified in the component description) and with the absolute values to have the circuit following your taste. A bit of experimentation will create a kind of *style options* that you could use in all your documents.

3.3.4 Style files

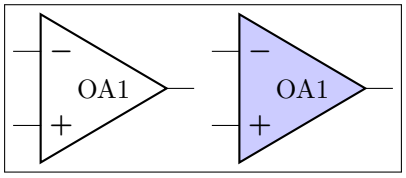
When using styles, it is possible to use *style files* (see section 3.3.5), that then you can load with the command `\ctikzloadstyle`. For example, in the distribution you have a number of style files: `legacy`, `romano`, `example`.When you load a style name *`name`*, you will have available a style called *`name`* `circuit style` that you can apply to your circuits. The last style loaded is not enacted — you have to explicitly do it if you want the style used by default, by putting for example in the preamble:

```
\ctikzloadstyle{romano}
\tikzset{romano circuit style}
```

Please notice that the style is at Ti*k*Z level, not CircuiTi*k*Z— that let's you use it in the top option of the circuit, like:

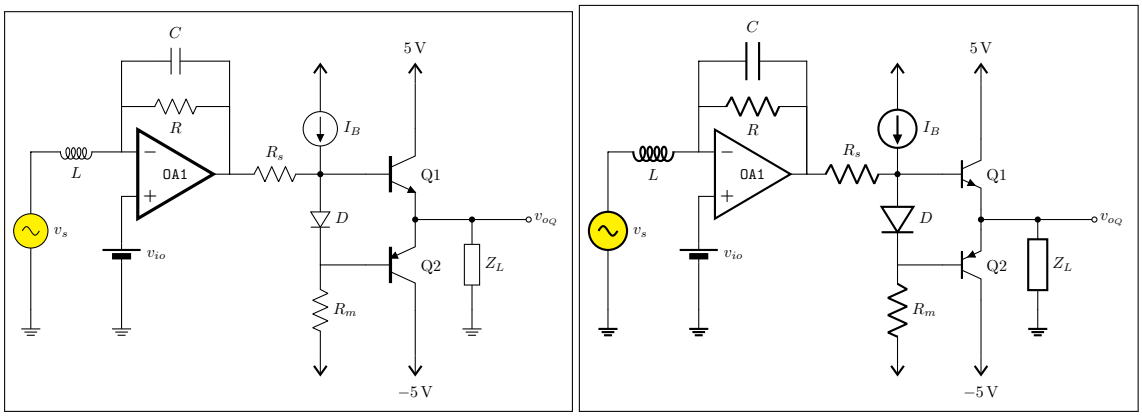
```
\begin{circuitikz}[legacy circuit style,
..., ]
...
\end{circuitikz}
```

The `example` style file will simply make the amplifiers filled with light blue:




```
\begin{circuitikz}
    \draw (0,0) node[op amp]{OA1};
\end{circuitikz}
\ctikzloadstyle{example}
\begin{circuitikz}[example circuit style]
    \draw (0,0) node[op amp]{OA1};
\end{circuitikz}
```

The styles `legacy` is a style that set (most) of the style parameters to the default, and `romano` is a style used by one of the authors; you can use these styles as is or you can use them to learn to how to write new file style following the instructions in section 3.3.5. In the next diagrams, the left hand one is using the `romano circuit style` and the rigth hand one the legacy style.



3.3.5 Style files: how to write them

The best option is to start from `ctikzstyle-legacy.tex` and edit your style file from it. Then you just put it in your input path and that's all. If you want, you can contribute your style file to the project.

Basically, to write the style `example`, you edit a file named `ctikzstyle-romano.tex` with will define and enact Ti*k*Z style with name `example circuit style`; basically it has to be something along this:

```
% example style for circuits
% Do not use LaTeX commands if you want it to be compatible with ConTeXt
% Do not add spurious spaces
\tikzset{example circuit style/.style={%
    \circuitikzbasekey/.cd,%
    amplifiers/fill=blue!20!white,
},% end .style
}% end \tikzset
%
\endinput
```

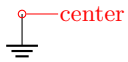
This kind of style will *add* to the existing style. If you want to have a style that *substitute* the current style, you should do like this:

```
\ctikzloadstyle{legacy}% start from a know state
\tikzset{romano circuit style/.style={%
legacy circuit style, % load the legacy style
\circuitikzbasekey/.cd,%
% Resistors
resistors/scale=0.8,
[...]
```

3.4 Grounds and supply voltages

3.4.1 Grounds

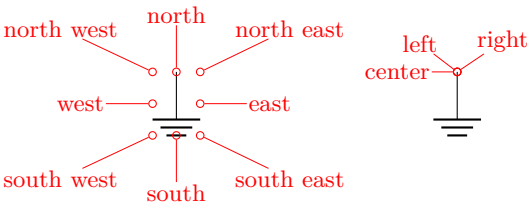
For the grounds, the `center` anchor is put on the connecting point of the symbol, so that you can use them directly in a `path` specification.

	Ground, type: node (`node[ground]{}`). Class: `grounds`.
	Tailless ground, type: node (`node[tlground]{}`). Class: `grounds`.
	Reference ground, type: node (`node[rground]{}`). Class: `grounds`.
	Signal ground, type: node, fillable (`node[sground]{}`). Class: `grounds`.
	Thicker tailless reference ground, type: node (`node[tground]{}`). Class: `grounds`.
	Noiseless ground, type: node (`node[nground]{}`). Class: `grounds`.
	Protective ground, type: node, fillable (`node[pground]{}`). Class: `grounds`.
	Chassis ground[5], type: node (`node[cground]{}`). Class: `grounds`.
	European style ground, type: node (`node[eground]{}`). Class: `grounds`.
	European style ground, version 2[6], type: node (`node[eground2]{}`). Class: `grounds`.

[5] These last three were contributed by Luigi «Liverpool»
[6] These last two were contributed by `@fotesan`

3.4.1.1 Grounds anchors

Anchors for grounds are a bit strange, given that they have the `center` spot at the same location than `north` and all the ground will develop "going down":



3.4.1.2 Grounds customization

You can change the scale of these components (all the ground symbols together) by setting the key `grounds/scale` (default `1.0`).

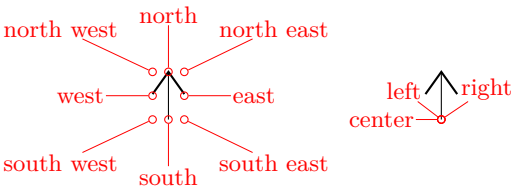
3.4.2 Power supplies

	VCC/VDD, type: node (`node[vcc]{}`). Class: `power supplies`.
	VEE/VSS, type: node (`node[vee]{}`). Class: `power supplies`.

The power supplies are normally drawn with the arrows shown in the list above.

3.4.2.1 Power supply anchors

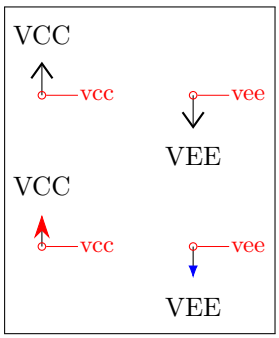
They are similar to grounds anchors, and the geographical anchors are correct only for the default arrow.



3.4.2.2 Power supplies customization

You can change the scale of the power supplies by setting the key `power supplies/scale` (default `1.0`).

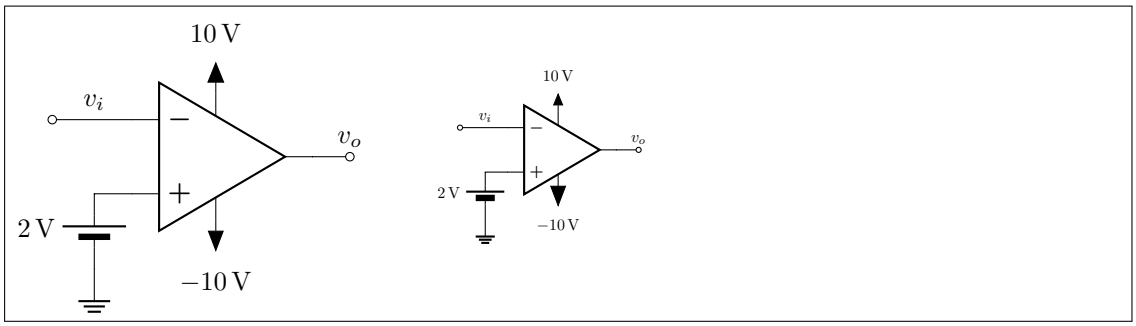
Given that the power supply symbols are basically arrows, you can change them using all the options of the `arrows.meta` package (see the Ti*k*Z manual for details) by changing the keys `monopoles/vcc/arrow` and `monopoles/vee/arrow` (the default for both is `legacy`, which will use the old code for drawing them). Note that the anchors are at the start of the connecting lines, and that geographical anchors are just approximation if you change the arrow symbol!




```
\begin{circuitikz}
  \def\coord(#1){\showcoord(#1)<0:0.3>}
  \draw (0,0)
  node[vcc](vcc){VCC} \coord(vcc) ++(2,0)
  node[vee](vee){VEE} \coord(vee);
  \ctikzset{monopoles/vcc/arrow={Stealth[red, width=6pt,
      length=9pt]}}
  \ctikzset{monopoles/vee/arrow={Latex[blue]}}
  \draw (0,-2)
  node[vcc](vcc){VCC} \coord(vcc) ++(2,0)
  node[vee](vee){VEE} \coord(vee);
\end{circuitikz}
```

However, arrows in Ti*k*Z are in the same class with the line thickness, so they do not scale with neither the class `power supplies` scale nor the global scale parameter (you should use `transform canvas=scale…` for this).

If you want that the arrows behave like the legacy symbols (which are shapes), *only in the arrow definitions*, you can use the special length parameter `\scaledwidth`[7] in the arrow definition, which correspond to the width of the legacy `vcc` or `vee`. Compare the effects on the following circuit.

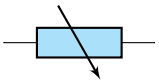



```
\ctikzset{%
   monopoles/vcc/arrow={Triangle[width=0.8*\scaledwidth, length=\scaledwidth]},
   monopoles/vee/arrow={Triangle[width=6pt, length=8pt]},
}
\begin{circuitikz}[baseline=(vo.center)]
   \node [ocirc](TW) at (0,0) {};
   \draw (TW.east) -- ++(1,0) node[midway, above]{$v_i$} node[op amp, anchor=-](A1){};
   \draw (A1.up) -- ++(0, 0.3) node[vcc]{\SI{+10}{V}};
   \draw (A1.down) -- ++(0,-0.3) node[vee]{\SI{-10}{V}};
   \draw (A1.+) -- ++(-0.5,0) to[battery2, invert, l_=\SI{2}{V}] ++(0,-1) node[ground
       ]{};
   \draw (A1.out) to[short, -o] ++(0.5,0) node[above](vo){$v_o$};
\end{circuitikz} \qquad
\begin{circuitikz}[baseline=(vo.center), scale=0.6, transform shape]
   \node [ocirc](TW) at (0,0) {};
   \draw (TW.east) -- ++(1,0) node[midway, above]{$v_i$} node[op amp, anchor=-](A1){};
   \draw (A1.up) -- ++(0, 0.3) node[vcc]{\SI{+10}{V}};
   \draw (A1.down) -- ++(0,-0.3) node[vee]{\SI{-10}{V}};
   \draw (A1.+) -- ++(-0.5,0) to[battery2, invert, l_=\SI{2}{V}] ++(0,-1) node[ground
       ]{};
   \draw (A1.out) to[short, -o] ++(0.5,0) node[above](vo){$v_o$};
\end{circuitikz}
```

3.5 Resistive bipoles

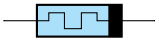
	short: Short circuit, `type: path-style, nodename: shortshape.` Class: `default`.
	open: Open circuit, `type: path-style, nodename: openshape.` Class: `default`.
	generic: Generic (symmetric) bipole, `type: path-style, fillable, nodename: genericshape.` Class: `resistors`.

[7] Thanks to @Schrödinger's cat on TEX stackexchange site



tgeneric: Tunable generic bipole, `type: path-style, fillable, nodename: tgenericshape.` Class: `resistors`.

ageneric: Generic asymmetric bipole, `type: path-style, fillable, nodename: agenericshape.` Class: `resistors`.

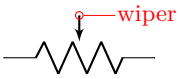


memristor: Memristor, `type: path-style, fillable, nodename: memristorshape.` Aliases: `Mr`. Class: `resistors`.

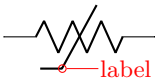
If `americanresistors` option is active (or the style `[american resistors]` is used; this is the default for the package), the resistors are displayed as follows:

R: Resistor, `type: path-style, nodename: resistorshape.` Aliases: `american resistor`. Class: `resistors`.

vR: Variable resistor, `type: path-style, nodename: vresistorshape.` Aliases: `variable american resistor`. Class: `resistors`.



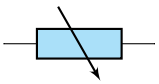
pR: Potentiometer, `type: path-style, nodename: potentiometershape.` Aliases: `american potentiometer`. Class: `resistors`.



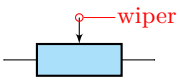
sR: Resisitive sensor, `type: path-style, nodename: resistivesensshape.` Aliases: `american resisitive sensor`. Class: `resistors`.

If instead `europeanresistors` option is active (or the style `[european resistors]` is used), the resistors, variable resistors and potentiometers are displayed as follows:

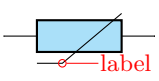
R: Resistor, `type: path-style, fillable, nodename: genericshape.` Aliases: `european resistor`. Class: `resistors`.



vR: Variable resistor, `type: path-style, fillable, nodename: tgenericshape.` Aliases: `variable european resistor`. Class: `resistors`.

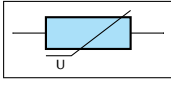


pR: Potentiometer, `type: path-style, fillable, nodename: genericpotentiometershape.` Aliases: `european potentiometer`. Class: `resistors`.



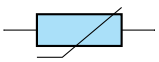
sR: Resistive sensor, `type: path-style, fillable, nodename: thermistorshape.` Aliases: `european resistive sensor`. Class: `resistors`.

Other miscellaneous resistor-like devices:

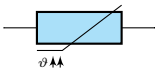


varistor: Varistor, `type: path-style, fillable, nodename: varistorshape.` Class: `resistors`.

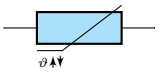
phR: Photoresistor, `type: path-style, fillable, nodename: photoresistorshape`. Aliases: `photoresistor`. Class: `resistors`.



thR: Thermistor, `type: path-style, fillable, nodename: thermistorshape`. Aliases: `thermistor`. Class: `resistors`.



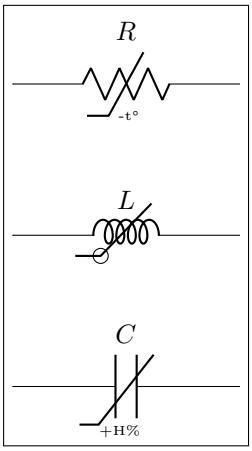
thRp: PTC thermistor, `type: path-style, fillable, nodename: thermistorptcshape`. Aliases: `thermistor ptc`. Class: `resistors`.



thRn: NTC thermistor, `type: path-style, fillable, nodename: thermistorntcshape`. Aliases: `thermistor ntc`. Class: `resistors`.

3.5.1 Generic sensors anchors

Generic sensors have an extra anchor named `label` to help position the type of dependence, if needed:




```
\begin{circuitikz}
  \draw (0,2) to[sR, l=$R$, name=mySR] ++(3,0);
  \node [font=\tiny, right] at(mySR.label) {-t\si{\degree
      }};
  \draw (0,0) to[sL, l=$L$, name=mySL] ++(3,0);
  \node [draw, circle, inner sep=2pt] at(mySL.label) {};
  \draw (0,-2) to[sC, l=$C$, name=mySC] ++(3,0);
  \node [font=\tiny, below right, inner sep=0pt] at(mySC.
      label) {+H\si{\%}};
\end{circuitikz}
```

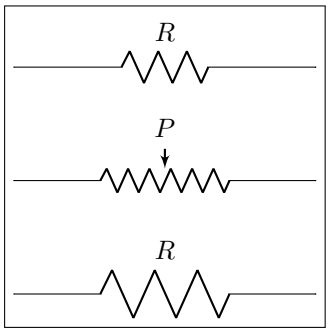
The anchor is positioned just on the corner of the segmented line crossing the component.

3.5.2 Resistive components customization

You can change the scale of these components (all the resistive bipoles together) by setting the key `resistors/scale` (default `1.0`). Similarly, you can change the widths by setting `resistors/width` (default `0.8`).

You can change the width of these components (all the resistive bipoles together) by setting the key `resistors/width` to something different from the default `0.8`.

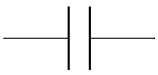
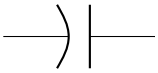
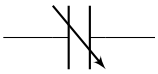
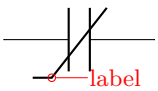
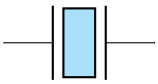
For the american style resistors, you can change the number of "zig-zags" by setting the key `resistors/zigs` (default value `3`).




```
\begin{circuitikz}[
      longpot/.style = {pR, resistors/scale=0.75,
      resistors/width=1.6, resistors/zigs=6}]
  \draw (0,1.5) to[R, l=$R$] ++(4,0);
  \draw (0,0) to[longpot, l=$P$] ++(4,0);
  \ctikzset{resistors/scale=1.5}
  \draw (0,-1.5) to[R, l=$R$] ++(4,0);
\end{circuitikz}
```

3.6 Capacitors and inductors: dynamical bipoles

3.6.1 Capacitors

	**capacitor**: Capacitor, `type: path-style, nodename: capacitorshape`. Aliases: `C`. Class: `capacitors`.
	**polar capacitor**: Polar capacitor, `type: path-style, nodename: pcapacitorshape`. Aliases: `pC`. Class: `capacitors`.
	ecapacitor: Electrolytic capacitor, `type: path-style, fillable, nodename: ecapacitorshape`. Aliases: `eC,elko`. Class: `capacitors`.
	**variable capacitor**: Variable capacitor, `type: path-style, nodename: vcapacitorshape`. Aliases: `vC`. Class: `capacitors`.
	**capacitive sensor**: Capacitive sensor, `type: path-style, nodename: capacitivesensshape`. Aliases: `sC`. Class: `capacitors`.
	**piezoelectric**: Piezoelectric Element, `type: path-style, fillable, nodename: piezoelectricshape`. Aliases: `PZ`. Class: `capacitors`.

3.6.2 Capacitive sensors anchors

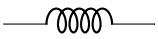
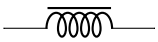
For capacitive sensors, see section 3.5.1.

3.6.3 Capacitors customizations

You can change the scale of the capacitors by setting the key `capacitors/scale` to something different from the default `1.0`.

3.6.4 Inductors

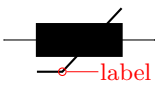
If the `cuteinductors` option is active (default behaviour), or the style `[cute inductors]` is used, the inductors are displayed as follows:

	`L`: Inductor, `type: path-style, nodename: cuteinductorshape`. Aliases: `cute inductor`. Class: `inductors`.
	`cute choke`: Choke, `type: path-style, nodename: cutechokeshape`. Class: `inductors`.
	`vL`: Variable inductor, `type: path-style, nodename: vcuteinductorshape`. Aliases: `variable cute inductor`. Class: `inductors`.
	`sL`: Inductive sensor, `type: path-style, nodename: scuteinductorshape`. Aliases: `cute inductive sensor`. Class: `inductors`.

If the `americaninductors` option is active (or the style `[american inductors]` is used), the inductors are displayed as follows:

	`L`: Inductor, `type: path-style, nodename: americaninductorshape`. Aliases: `american inductor`. Class: `inductors`.
	`vL`: Variable inductor, `type: path-style, nodename: vamericaninductorshape`. Aliases: `variable american inductor`. Class: `inductors`.
	`sL`: Inductive sensor, `type: path-style, nodename: samericaninductorshape`. Aliases: `american inductive sensor`. Class: `inductors`.

Finally, if the `europeaninductors` option is active (or the style `[european inductors]` is used), the inductors are displayed as follows:

	`L`: Inductor, `type: path-style, nodename: fullgenericshape`. Aliases: `european inductor`. Class: `inductors`.
	`vL`: Variable inductor, `type: path-style, nodename: tfullgenericshape`. Aliases: `variable european inductor`. Class: `inductors`.
	`sL`: Inductive sensor, `type: path-style, nodename: sfullgenericshape`. Aliases: `european inductive sensor`. Class: `inductors`.

3.6.5 Inductive sensors anchors

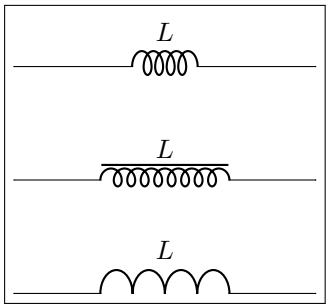
For inductive sensors, see section 3.5.1.

3.6.6 Inductors customizations

You can change the scale of the inductors by setting the key `inductors/scale` to something different from the default `1.0`.

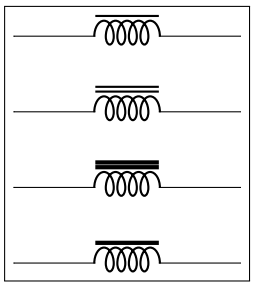
You can change the width of these components (all the inductors together, unless you use style or scoping) by setting the key `inductors/width` to something different from the default, which is `0.8` for american and european inductors, and `0.6` for cute inductors.

Moreover, you can change the number of "coils" drawn by setting the key `inductors/coils` (default value `5` for cute inductors and `4` for american ones).




```
\begin{circuitikz}[
      longL/.style = {cute choke, inductors/scale
          =0.75,
      inductors/width=1.6, inductors/coils=9}]
  \draw (0,1.5) to[L, l=$L$] ++(4,0);
  \draw (0,0) to[longL, l=$L$] ++(4,0);
  \ctikzset{inductors/scale=1.5, inductor=american}
  \draw (0,-1.5) to[L, l=$L$] ++(4,0);
\end{circuitikz}
```

Chokes (which comes only in the `cute` style) can have single and double lines, and can have the line thickness adjust (the value is relative to the thickness of the inductor).



```
\begin{circuitikz}[american]
   \draw (0,0) to[cute choke] ++(3,0);
   \draw (0,-1) to[cute choke, twolineschoke] ++(3,0);

   \ctikzset{bipoles/cutechoke/cthick=2, twolineschoke}

   \draw (0,-2) to[cute choke] ++(3,0);
   \draw (0,-3) to[cute choke, onelinechoke] ++(3,0);
\end{circuitikz}
```

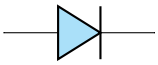
3.7 Diodes and such

There are three basic styles for diodes: `empty` (fillable in color), `full` (completely filled with the draw color) and `stroke` (empty, but with a line across them).

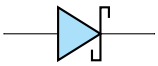
You can switch between the styles setting the key `diode` (for example `\ctikzset{diode=full}` or `empty` or `stroke`, or with the styles `full diodes`, `empty diodes` and `stroke diodes`.

To use the default element, simply use the name shown for the empty diodes without the final "o" — that is `D`, `sD`, and so on. The names shown in the following tables will draw the specified diode independently on the style chosen (that is, `leD*` is always a full LED diode).

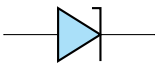
The package options `fulldiode`, `strokediode`, and `emptydiode` (and the styles `[full diodes]`, `[stroke diodes]`, and `[empty diodes]`) define which shape will be used by abbreviated commands such that `D`, `sD`, `zD`, `zzD`, `tD`, `pD`, `leD`, `VC`, `Ty`,`Tr` (no stroke symbol available!).



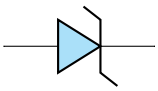
empty diode: Empty diode, type: path-style, fillable, nodename: emptydiodeshape. Aliases: Do. Class: diodes.



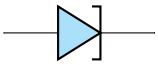
empty Schottky diode: Empty Schottky diode, type: path-style, fillable, nodename: emptysdiodeshape. Aliases: sDo. Class: diodes.



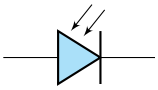
empty Zener diode: Empty Zener diode, type: path-style, fillable, nodename: emptyzdiodeshape. Aliases: zDo. Class: diodes.



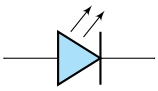
empty ZZener diode: Empty ZZener diode, type: path-style, fillable, nodename: emptyzzdiodeshape. Aliases: zzDo. Class: diodes.



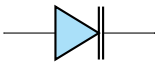
empty tunnel diode: Empty tunnel diode, type: path-style, fillable, nodename: emptytdiodeshape. Aliases: tDo. Class: diodes.



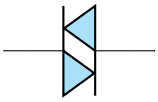
empty photodiode: Empty photodiode, type: path-style, fillable, nodename: emptypdiodeshape. Aliases: pDo. Class: diodes.



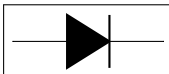
empty led: Empty led, type: path-style, fillable, nodename: emptylediodeshape. Aliases: leDo. Class: diodes.



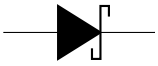
empty varcap: Empty varcap, type: path-style, fillable, nodename: emptyvarcapshape. Aliases: VCo. Class: diodes.



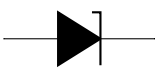
empty bidirectionaldiode: Empty bidirectionaldiode, type: path-style, fillable, nodename: emptybidirectionaldiodeshape. Aliases: biDo. Class: diodes.



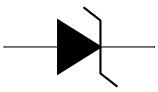
full diode: Full diode, type: path-style, nodename: fulldiodeshape. Aliases: D*. Class: diodes.



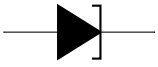
full Schottky diode: Full Schottky diode, type: path-style, nodename: fullsdiodeshape. Aliases: sD*. Class: diodes.



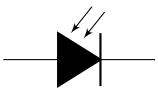
full Zener diode: Full Zener diode, type: path-style, nodename: fullzdiodeshape. Aliases: zD*. Class: diodes.



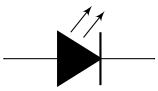
full ZZener diode: Full ZZener diode, type: path-style, nodename: fullzzdiodeshape. Aliases: zzD*. Class: diodes.



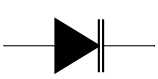
full tunnel diode: Full tunnel diode, type: path-style, nodename: fulltdiodeshape. Aliases: tD*. Class: diodes.



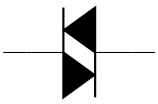
full photodiode: Full photodiode, `type: path-style, nodename: fullpdiodeshape`. Aliases: `pD*`. Class: `diodes`.



full led: Full led, `type: path-style, nodename: fullledиodeshape`. Aliases: `leD*`. Class: `diodes`.

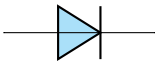


full varcap: Full varcap, `type: path-style, nodename: fullvarcapshape`. Aliases: `VC*`. Class: `diodes`.

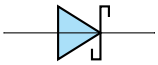


full bidirectionaldiode: Full bidirectionaldiode, `type: path-style, nodename: fullbidirectionaldiodeshape`. Aliases: `biD*`. Class: `diodes`.

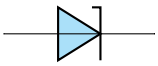
These shapes have no exact node-style counterpart, because the stroke line is built upon the empty variants:



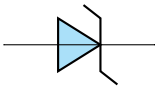
stroke diode: Stroke diode, `type: path-style, fillable, nodename: emptydiodeshape`. Aliases: `D-`. Class: `diodes`.



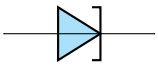
stroke Schottky diode: Stroke Schottky diode, `type: path-style, fillable, nodename: emptysdiodeshape`. Aliases: `sD-`. Class: `diodes`.



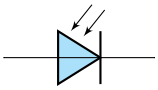
stroke Zener diode: Stroke Zener diode, `type: path-style, fillable, nodename: emptyzdiodeshape`. Aliases: `zD-`. Class: `diodes`.



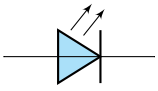
stroke ZZener diode: Stroke ZZener diode, `type: path-style, fillable, nodename: emptyzzdiodeshape`. Aliases: `zzD-`. Class: `diodes`.



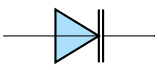
stroke tunnel diode: Stroke tunnel diode, `type: path-style, fillable, nodename: emptytdiodeshape`. Aliases: `tD-`. Class: `diodes`.



stroke photodiode: Stroke photodiode, `type: path-style, fillable, nodename: emptypdiodeshape`. Aliases: `pD-`. Class: `diodes`.



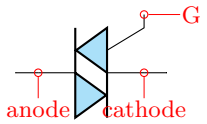
stroke led: Stroke led, `type: path-style, fillable, nodename: emptylediodeshape`. Aliases: `leD-`. Class: `diodes`.



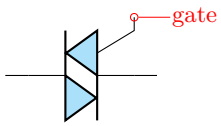
stroke varcap: Stroke varcap, `type: path-style, fillable, nodename: emptyvarcapshape`. Aliases: `VC-`. Class: `diodes`.

3.7.1 Tripole-like diodes

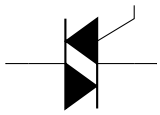
The following tripoles are entered with the usual command, of the form `to[Tr, …]`.



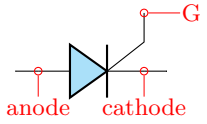
triac: Standard triac (shape depends on package option), `type: path-style, fillable, nodename: emptytriacshape`. Aliases: `Tr`. Class: `diodes`.



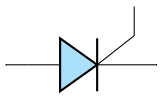
empty triac: Empty triac, `type: path-style, fillable, nodename: emptytriacshape`. Aliases: `Tro`. Class: `diodes`.



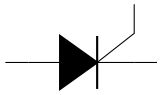
full triac: Full triac, `type: path-style, nodename: fulltriacshape`. Aliases: `Tr*`. Class: `diodes`.



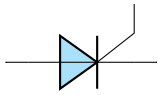
thyristor: Standard thyristor (shape depends on package option), `type: path-style, fillable, nodename: emptythyristorshape`. Aliases: `Ty`. Class: `diodes`.



empty thyristor: Empty thyristor, `type: path-style, fillable, nodename: emptythyristorshape`. Aliases: `Tyo`. Class: `diodes`.



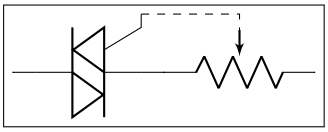
full thyristor: Full thyristor, `type: path-style, nodename: fullthyristorshape`. Aliases: `Ty*`. Class: `diodes`.



stroke thyristor: Stroke thyristor, `type: path-style, fillable, nodename: emptythyristorshape`. Aliases: `Ty-`. Class: `diodes`.

3.7.2 Triacs anchors

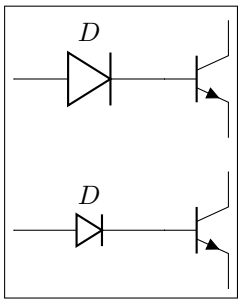
When inserting a thrystor, a triac or a potentiometer, one needs to refer to the third node-gate (`gate` or `G`) for the former two; wiper (`wiper` or `W`) for the latter one. This is done by giving a name to the bipole:



```
\begin{circuitikz} \draw
  (0,0) to[Tr, n=TRI] (2,0)
        to[pR, n=POT] (4,0);
  \draw[dashed] (TRI.G) -| (POT.wiper)
;\end{circuitikz}
```

3.7.3 Diode customizations

You can change the scale of the diodes by setting the key `diodes/scale` to something different from the default `1.0`. In Romano's opinion, diodes are somewhat big with the default style of the package, so a setting like `\ctikzset{diode/scale=0.6}` is recommended.




```
\begin{circuitikz}
  \draw (0,1) to[D, l=$D$] ++(2,0)
    node[npn, anchor=B]{};
  \ctikzset{diodes/scale=0.6}
  \draw (0,-1) to[D, l=$D$] ++(2,0)
    node[npn, anchor=B]{};
\end{circuitikz}
```

3.8 Sources and generators

Notice that source and generators are divided in three classes that can be styled independently: traditional battery symbols (class `batteries`), independent generators (class `sources`) and dependent generators (class `csources`). This is because they are often treated differently, and so you can choose to, for example, fill the dependent sources but not the independent ones.

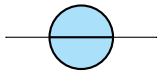
3.8.1 Batteries

battery: Battery, `type: path-style, nodename: batteryshape.` Class: `batteries`.

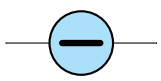
battery1: Single battery cell, `type: path-style, nodename: battery1shape.` Class: `batteries`.

battery2: Single battery cell, `type: path-style, nodename: battery2shape.` Class: `batteries`.

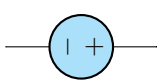
3.8.2 Stationary sources



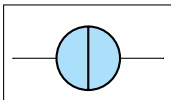
european voltage source: Voltage source (european style), `type: path-style, fillable, nodename: vsourceshape.` Class: `sources`.



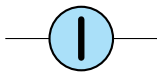
cute european voltage source: Voltage source (cute european style), `type: path-style, fillable, nodename: vsourceCshape.` Aliases: `vsourceC, ceV`. Class: `sources`.



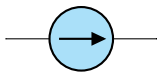
american voltage source: Voltage source (american style), `type: path-style, fillable, nodename: vsourceAMshape.` Class: `sources`.



european current source: Current source (european style), `type: path-style, fillable, nodename: isourceshape.` Class: `sources`.



cute european current source: Current source (cute european style), `type: path-style, fillable, nodename: isourceCshape.` Aliases: `isourceC, ceI`. Class: `sources`.



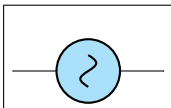
american current source: Current source (american style), `type: path-style, fillable, nodename: isourceAMshape.` Class: `sources`.

If (default behaviour) `europeancurrents` option is active (or the style `[european currents]` is used), the shorthands `current source`, `isource`, and `I` are equivalent to `european current source`. Otherwise, if `americancurrents` option is active (or the style `[american currents]` is used) they are equivalent to `american current source`.

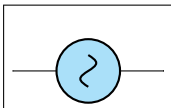
Similarly, if (default behaviour) `europeanvoltages` option is active (or the style `[european voltages]` is used), the shorthands `voltage source`, `vsource`, and `V` are equivalent to `european voltage source`. Otherwise, if `americanvoltages` option is active (or the style `[american voltages]` is used) they are equivalent to `american voltage source`.

3.8.3 Sinusoidal sources

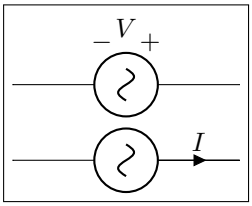
These two are basically the same symbol; to distinguish among them, you have to add a label, which will be a voltage or a current.



sinusoidal voltage source: Sinusoidal voltage source, `type: path-style, fillable, nodename: vsourcesinshape.` Aliases: `vsourcesin, sV`. Class: `sources`.

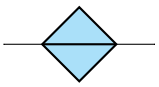
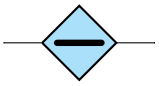
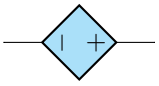
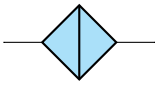
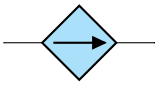
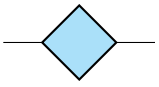


sinusoidal current source: Sinusoidal current source, `type: path-style, fillable, nodename: isourcesinshape.` Aliases: `isourcesin, sI`. Class: `sources`.




```
\begin{circuitikz}[american]
  \draw (0,1) to[sV=$V$] ++(3,0);
  \draw (0,0) to[sI=$I$] ++(3,0);
\end{circuitikz}
```

3.8.4 Controlled sources

	**european controlled voltage source**: Controlled voltage source (european style), `type: path-style, fillable, nodename: cvsourceshape.` Class: `csources`.
	**cute european controlled voltage source**: Voltage source (cute european style), `type: path-style, fillable, nodename: cvsourceCshape.` Aliases: `cvsourceC, cceV`. Class: `csources`.
	**american controlled voltage source**: Controlled voltage source (american style), `type: path-style, fillable, nodename: cvsourceAMshape.` Class: `csources`.
	**european controlled current source**: Controlled current source (european style), `type: path-style, fillable, nodename: cisourceshape.` Class: `csources`.
	cute european controlled current source: Current source (cute european style), `type: path-style, fillable, nodename: cisourceCshape.` Aliases: `cisourceC, cceI`. Class: `csources`.
	**american controlled current source**: Controlled current source (american style), `type: path-style, fillable, nodename: cisourceAMshape.` Class: `csources`.
	**empty controlled source**: Empty controlled source, `type: path-style, fillable, nodename: ecsourceshape.` Aliases: `ecsource`. Class: `csources`.

If (default behaviour) `europeancurrents` option is active (or the style `[european currents]` is used), the shorthands `controlled current source`, `cisource`, and `cI` are equivalent to `european controlled current source`. Otherwise, if `americancurrents` option is active (or the style `[american currents]` is used) they are equivalent to `american controlled current source`.

Similarly, if (default behaviour) `europeanvoltages` option is active (or the style `[european voltages]` is used), the shorthands `controlled voltage source`, `cvsource`, and `cV` are equivalent to `european controlled voltage source`. Otherwise, if `americanvoltages` option is active (or the style `[american voltages]` is used) they are equivalent to `american controlled voltage source`.

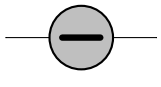
The following two behave like the corresponding independent sources, see section 3.8.3.

controlled sinusoidal voltage source: Controlled sinusoidal voltage source, `type: path-style, fillable, nodename: cvsourcesinshape`. Aliases: `controlled vsourcesin, cvsourcesin, csV`. Class: `csources`.

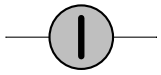
controlled sinusoidal current source: Controlled sinusoidal current source, `type: path-style, fillable, nodename: cisourcesinshape`. Aliases: `controlled isourcesin, cisourcesin, csI`. Class: `csources`.

3.8.5 Noise sources

In this case, the "direction" of the source is undefined. Noise sources are filled in gray by default, but if you choose the dashed style, they become fillable.

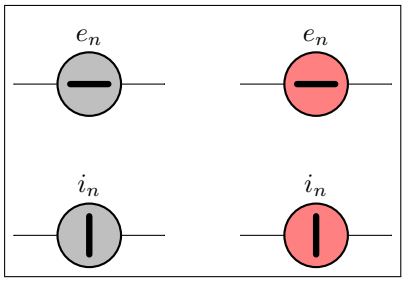


noise voltage source: Sinusoidal voltage source, `type: path-style, nodename: vsourceNshape`. Aliases: `vsourceN, nV`. Class: `sources`.



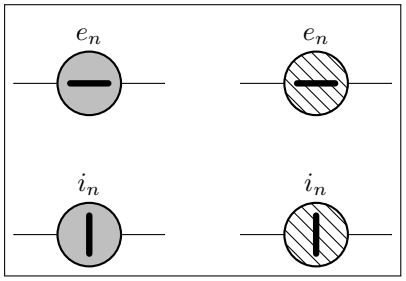
noise current source: Sinusoidal current source, `type: path-style, nodename: isourceNshape`. Aliases: `isourceN, nI`. Class: `sources`.

You can change the fill color with the key `circuitikz/bipoles/noise sources/fillcolor`:



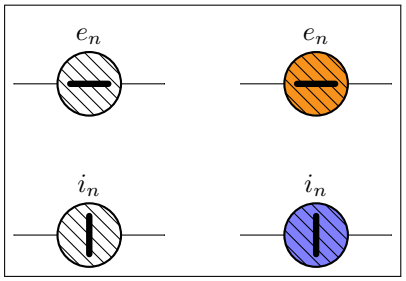

```
\begin{circuitikz}
    \draw(0,0) to [nV, l=$e_n$] ++(2,0);
    \draw(0,-2) to [nI, l=$i_n$] ++(2,0);
    \begin{scope}[circuitikz/bipoles/noise
        sources/fillcolor=red!50]
        \draw(3,0) to [nV, l=$e_n$] ++(2,0);
        \draw(3,-2) to [nI, l=$i_n$] ++(2,0);
    \end{scope}
\end{circuitikz}
```

If you prefer a patterned noise generator (similar to the one you draw by hand) you can use the fake color `dashed`:



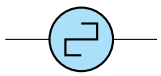
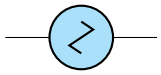
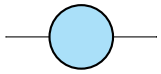
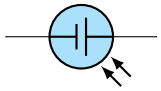
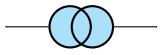
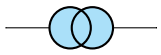

```
\begin{circuitikz}
    \draw(0,0) to [nV, l=$e_n$] ++(2,0);
    \draw(0,-2) to [nI, l=$i_n$] ++(2,0);
    \begin{scope}[circuitikz/bipoles/noise
        sources/fillcolor=dashed]
        \draw(3,0) to [nV, l=$e_n$] ++(2,0);
        \draw(3,-2) to [nI, l=$i_n$] ++(2,0);
    \end{scope}
\end{circuitikz}
```

Notice that if you choose the dashed style, the noise sources are fillable:

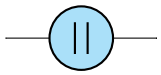
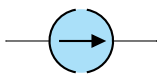



```
\begin{circuitikz}
    \ctikzset{bipoles/noise sources/fillcolor=
        dashed}
    \draw(0,0) to [nV, l=$e_n$] ++(2,0);
    \draw(0,-2) to [nI, l=$i_n$] ++(2,0);
    \begin{scope}
        \draw(3,0) to [nV, l=$e_n$, fill=yellow
            !50!red] ++(2,0);
        \draw(3,-2) to [nI, l=$i_n$, fill=blue
            !50!white] ++(2,0);
    \end{scope}
\end{circuitikz}
```

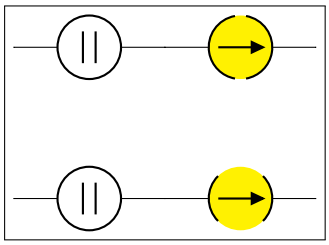
3.8.6 Special sources

	**square voltage source**: Square voltage source, `type: path-style, fillable, nodename: vsourcesquareshape`. Aliases: `vsourcesquare, sqV`. Class: `sources`.
	**vsourcetri**: Triangle voltage source, `type: path-style, fillable, nodename: vsourcetrishape`. Aliases: `tV`. Class: `sources`.
	**esource**: Empty voltage source, `type: path-style, fillable, nodename: esourceshape`. Class: `sources`.
	**pvsource**: Photovoltaic-voltage source, `type: path-style, fillable, nodename: pvsourceshape`. Class: `sources`.
	**ioosource**: Double Zero style current source, `type: path-style, fillable, nodename: oosourceshape`. Class: `sources`.
	**voosource**: Double Zero style voltage source, `type: path-style, fillable, nodename: oosourceshape`. Class: `sources`.

3.8.7 DC sources

	**dcvsource**: DC voltage source, `type: path-style, fillable, nodename: dcvsourceshape`. Class: `sources`.
	**dcisource**: DC current source, `type: path-style, fillable, nodename: dcisourceshape`. Class: `sources`.

The size of the broken part of the DC current source is configurable by changing the value of `bipoles/dcisource/angle` (default `80`); values must be between 0 (no circle at all, probably not useful) and 90 (full circle, again not useful).

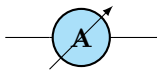
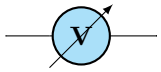
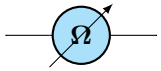
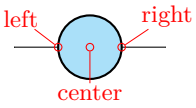
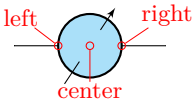
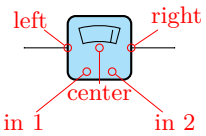
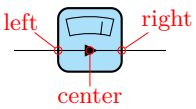
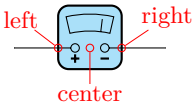
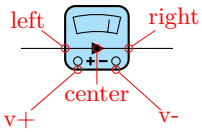


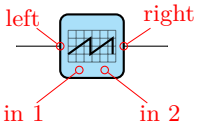
```
\begin{circuitikz}
    \draw (0,0) to[dcvsource] ++(2,0)
    to [dcisource, fill=yellow] ++(2,0) ;
    \ctikzset{bipoles/dcisource/angle=45}
    \draw (0,-2) to[dcvsource] ++(2,0)
    to [dcisource, fill=yellow] ++(2,0) ;
\end{circuitikz}
```

3.8.8 Sources customizations

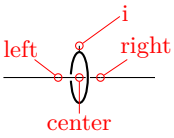
You can change the scale of the batteries by setting the key `batteries/scale`, for the controlled (dependent) sources with `csources/scale`, and for all the other independent sources and generators with `sources/scale`, to something different from the default `1.0`.

3.9 Instruments

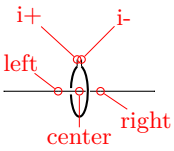
	**ammeter**: Ammeter, `type: path-style, fillable, nodename: ammetershape.` Class: `instruments`.
	**voltmeter**: Voltmeter, `type: path-style, fillable, nodename: voltmetershape.` Class: `instruments`.
	**ohmmeter**: Ohmmeter, `type: path-style, fillable, nodename: ohmmetershape.` Class: `instruments`.
	**rmeter**: Round meter (use `t=...` for the symbol), `type: path-style, fillable, nodename: rmetershape.` Class: `instruments`.
	**rmeterwa**: Round meter with arrow (use `t=...` for the symbol), `type: path-style, fillable, nodename: rmeterwashape.` Class: `instruments`.
	**smeter**: Square meter (use `t=...` for the symbol), `type: path-style, fillable, nodename: smetershape.` Class: `instruments`.
	**qiprobe**: QUCS-style current probe, `type: path-style, fillable, nodename: qiprobeshape.` Class: `instruments`.
	**qvprobe**: QUCS-style voltage probe, `type: path-style, fillable, nodename: qvprobeshape.` Class: `instruments`.
	**qpprobe**: QUCS-style power probe, `type: path-style, fillable, nodename: qpprobeshape.` Class: `instruments`.



`oscope`: Oscilloscope[8], `type: path-style, fillable, nodename: oscopeshape.` Class: `instruments`.



`iloop`: Current loop (symbolic), `type: path-style, nodename: iloopshape.` Class: `instruments`.



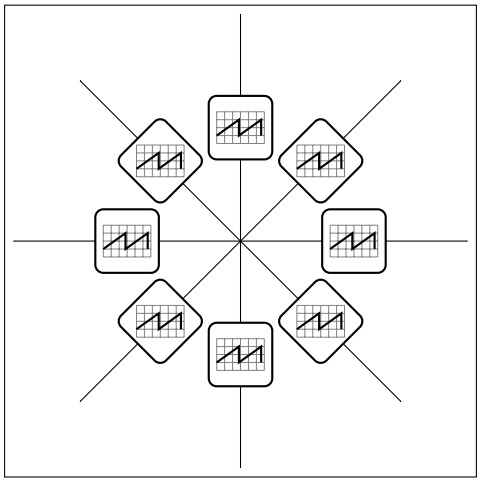
`iloop2`: Current loop (real), `type: path-style, nodename: iloop2shape.` Class: `instruments`.

3.9.1 Instruments customizations

You can change the scale of all the instruments (including the current loops) by setting the key `instruments/scale` to something different from the default `1.0`.

3.9.2 Rotation-invariant elements

The `oscope` element will not rotate the "graph" shown with the component:



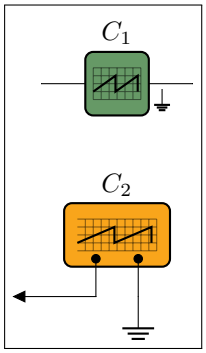
```
\begin{circuitikz}
    \foreach \a in {0,45,...,350} {
        \draw (0,0) to[oscope] (\a:3);
    }
\end{circuitikz}
```

The `rmeter`, `rmaterwa`, and `smeter` have the same behavior.

[8] Suggested by `@nobrl` on GitHub

3.9.3 Instruments as node elements

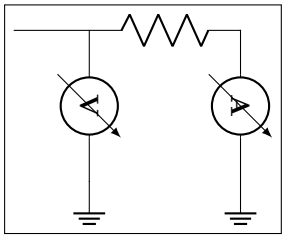
The node-style usage of the `oscope` is also interesting, using the additional `in 1` and `in 2` anchors; notice that in this case you can use the text content of the node to put labels above it. Moreover, you can change the size of the oscilloscope by changing `bipoles/oscope/width` and `bipoles/oscope/height` keys (which both default to 0.6).




```
\begin{circuitikz}
    \draw (0,1)
        to[oscope=$C_1$, fill=green!20!gray, name=O1] ++(2,0);
    \path (O1.right)
        node[ground, scale=0.5, below right=4pt]{};
    \ctikzset{bipoles/oscope/width=1.0}
    \draw (1,-1)
        node[oscopeshape, fill=yellow!20!orange](O2){$C_2$};
    \draw (O2.in 2) to[short, *-] ++(0,-0.5) node[ground]{};
    \draw (O2.in 1) to[short, *-] ++(0,-0.5)
          -- ++(-1,0) node[currarrow, xscale=-1]{};
\end{circuitikz}
```

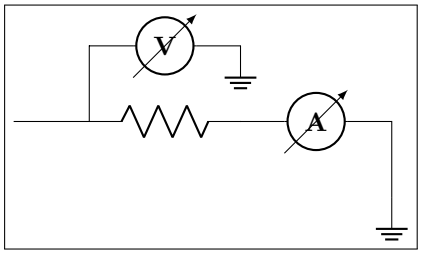
3.9.4 Measuring voltage and currents, multiple ways

This is the classical (legacy) option, with the `voltmeter` and `ammeter`. The problem is that elements are intrinsically horizontal, so they look funny if put in vertically.



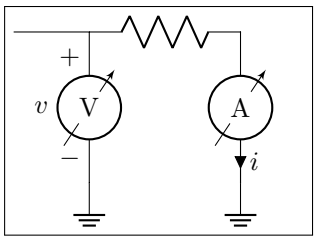

```
\begin{circuitikz}
    \draw (0,0) -- ++(1,0) to[R] ++(2,0)
    to [ammeter] ++(0,-2) node[ground]{};
    \draw (1,0) to[voltmeter] ++(0,-2)
    node[ground]{};
\end{circuitikz}
```

So the solution is often changing the structure to keep the meters in horizontal position.



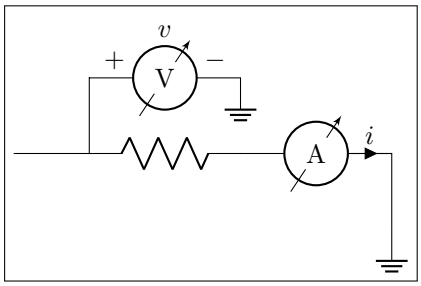

```
\begin{circuitikz}
    \draw (0,0) -- ++(1,0) to[R] ++(2,0)
    to [ammeter] ++(2,0) --
    ++(0,-1) node[ground]{};
    \draw (1,0) -- (1,1) to[voltmeter]
    ++(2,0) node[ground]{};
\end{circuitikz}
```

Since version 0.9.0 you have more options for the measuring instruments. You can use the generic `rmeterwa` (round meter with arrow), to which you can specify the internal symbol with the option `t=...` (and is fillable).



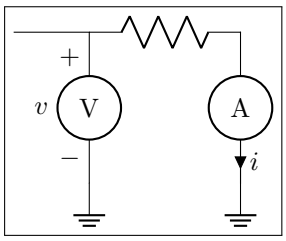

```
    \begin{circuitikz}[american]
    \draw (0,0) -- ++(1,0) to[R] ++(2,0)
    to [rmeterwa, t=A, i=$i$] ++(0,-2) node[ground]{};
    \draw (1,0) to[rmeterwa, t=V, v=$v$] ++(0,-2)
    node[ground]{};
\end{circuitikz}
```

This kind of component will keep the symbol horizontal, whatever the orientation:



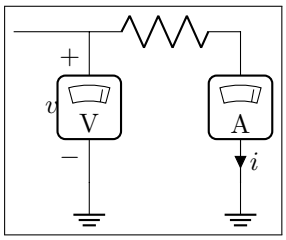

```
\begin{circuitikz}[american]
\draw (0,0) -- ++(1,0) to[R] ++(2,0)
to [rmeterwa, t=A, i=$i$] ++(2,0) --
++(0,-1) node[ground]{};
\draw (1,0) -- (1,1) to[rmeterwa, t=V, v^=$v
    $]
++(2,0) node[ground]{};
\end{circuitikz}
```

The plain `rmeter` is the same, without the measuring arrow:



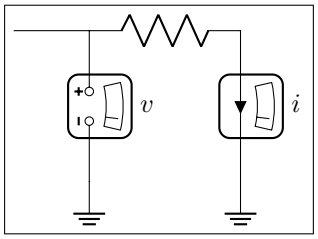

```
\begin{circuitikz}[american]
    \draw (0,0) -- ++(1,0) to[R] ++(2,0)
    to [rmeter, t=A, i=$i$] ++(0,-2) node[ground]{};
    \draw (1,0) to[rmeter, t=V, v=$v$] ++(0,-2)
    node[ground]{};
\end{circuitikz}
```

If you prefer it, you have the option to use square meters, in order to have more visual difference from generators:




```
\begin{circuitikz}[american]
    \draw (0,0) -- ++(1,0) to[R] ++(2,0)
    to [smeter, t=A, i=$i$] ++(0,-2) node[ground]{};
    \draw (1,0) to[smeter, t=V, v=$v$] ++(0,-2)
    node[ground]{};
\end{circuitikz}
```

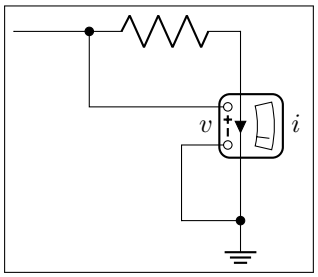
Another possibility is to use QUCS[9]-style probes, which have the nice property of explictly showing the type of connection (in series or parallel) of the meter:




```
\begin{circuitikz}[american]
    \draw (0,0) -- ++(1,0) to[R] ++(2,0)
    to [qiprobe, l=$i$] ++(0,-2) node[ground]{};
    \draw (1,0) to[qvprobe, l=$v$] ++(0,-2)
    node[ground]{};
\end{circuitikz}
```

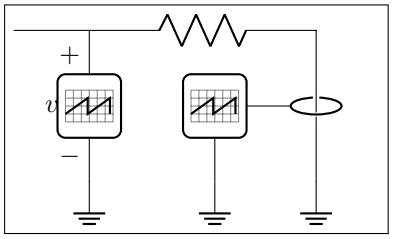
If you want to explicitly show a power measurement, you can use the power probe `qpprobe` and using the additional anchors `v+` and `v-` :

[9] QUCS is an open source circuit simulator: http://qucs.sourceforge.net/




```
\begin{circuitikz}[american]
    \draw (0,0) to[short,-*] ++(1,0) coordinate(b)
    to[R] ++(2,0) to [qpprobe, l=$i$, a=$v$, name=P]
    ++(0,-2.5) node[ground](GND){};
    \draw (P.v-) -| ++(-0.5,-1) coordinate(a)
    to [short, -*] (a-|GND);
    \draw (P.v+) -| (b);
\end{circuitikz}
```

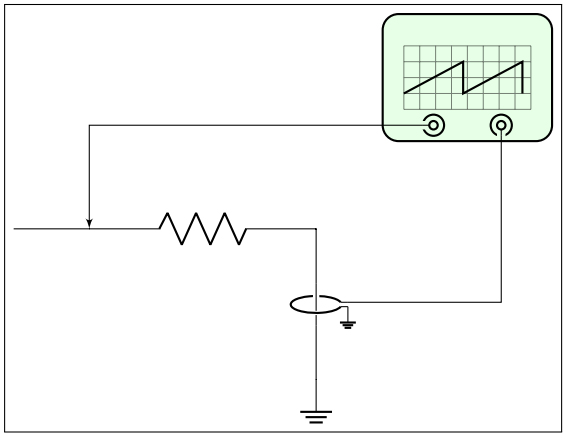
The final possibility is to use oscilloscopes. For example:



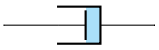
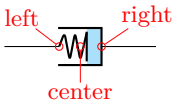
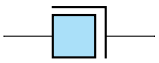

```
\begin{circuitikz}[american]
    \draw (0,0) -- ++(1,0) to[R] ++(3,0)
    to [iloop, mirror, name=I] ++(0,-2)
    node[ground] (GND){};
    \draw (1,0) to[oscope, v=$v$] ++(0,-2)
    node[ground]{};
    \draw (I.i) -- ++(-0.5,0) node[oscopeshape,
        anchor=right, name=O]{};
    \draw (O.south) -- (O.south |- GND) node[
        ground]{};
\end{circuitikz}
```

Or, if you want a more physical structure for the measurement setup:

```
\begin{circuitikz}[american]
    \draw (0,0) -- ++(1,0) to[R] ++(3,0) to [iloop2, name=I] ++(0,-2)
    node[ground] (GND){};
    \ctikzset{bipoles/oscope/width=1.6}\ctikzset{bipoles/oscope/height=1.2}
    \node [oscopeshape, fill=green!10](O) at (6,2){};
    \node [bnc, xscale=-1, anchor=zero](bnc1) at (O.in 1){};
    \node [bnc, , anchor=zero, rotate=-90](bnc2) at (O.in 2){};
    \draw [-latexslim] (bnc1.hot) -| (1,0);
    \draw (bnc2.hot) |- (I.i+);
    \draw (I.i-) node[ground, scale=0.5]{};
\end{circuitikz}
```



3.10 Mechanical Analogy

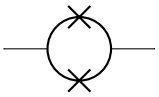
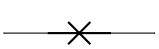
	**damper**: Mechanical Damping, `type: path-style, fillable, nodename: dampershape.` Class: `mechanicals`.
	spring: Mechanical Stiffness, `type: path-style, nodename: springshape.` Class: `mechanicals`.
	**viscoe**: Mechanical viscoelastic element[10], `type: path-style, fillable, nodename: viscoeshape.` Class: `mechanicals`.
	**mass**: Mechanical Mass, `type: path-style, fillable, nodename: massshape.` Class: `mechanicals`.

3.10.1 Mechanical elements customizations

You can change the scale of all the mechanical elements by setting the key `mechanicals/scale` to something different from the default `1.0`.

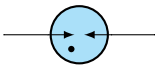
3.11 Miscellaneous bipoles

Here you'll find bipoles that are not easily grouped in the categories above.

	thermocouple: Thermocouple, `type: path-style, nodename: thermocoupleshape.` Class: `misc`.
	fuse: Fuse, `type: path-style, fillable, nodename: fuseshape.` Class: `misc`.
	afuse: Asymmetric fuse, `type: path-style, fillable, nodename: afuseshape.`Aliases: `asymmetric fuse`. Class: `misc`.
	**squid**: Squid, `type: path-style, nodename: squidshape.` Class: `misc`.
	**barrier**: Barrier, `type: path-style, nodename: barriershape.` Class: `misc`.

	european gas filled surge arrester: European gas filled surge arrester, `type: path-style, fillable, nodename: european gas filled surge arrestershape.` Class: `misc`.

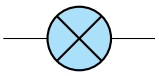
[10] Suggested by @Alex in https://tex.stackexchange.com/q/484268/38080



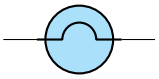
american gas filled surge arrester: American gas filled surge arrester, `type: path-style, fillable, nodename: american gas filled surge arrestershape.` Class: `misc`.

If (default behaviour) `europeangfsurgearrester` option is active (or the style `[european gas filled surge arrester]` is used), the shorthands `gas filled surge arrester` and `gf surge arrester` are equivalent to the european version of the component.

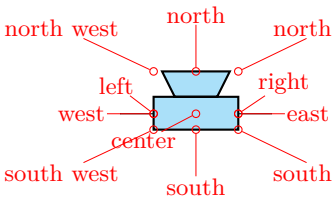
If otherwise `americangfsurgearrester` option is active (or the style `[american gas filled surge arrester]` is used), the shorthands the shorthands `gas filled surge arrester` and `gf surge arrester` are equivalent to the american version of the component.



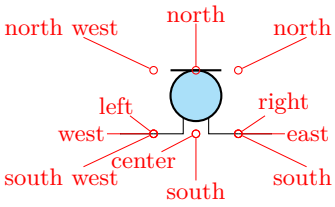
lamp: Lamp, `type: path-style, fillable, nodename: lampshape.` Class: `misc`.



bulb: Bulb, `type: path-style, fillable, nodename: bulbshape.` Class: `misc`.

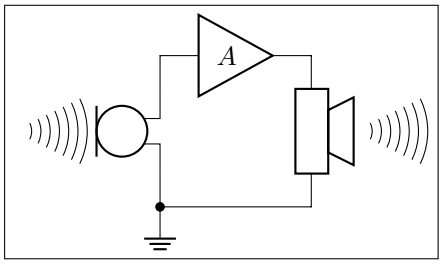


loudspeaker: loudspeaker, `type: path-style, fillable, nodename: loudspeakershape.` Class: `misc`.



mic: mic, `type: path-style, fillable, nodename: micshape.` Class: `misc`.

You can use microphones and loudspeakers with `waves` (see section 3.18) too:




```
\begin{circuitikz}
    \draw (0,0) to[mic, name=M] ++(0,2)
    to[amp, t=$A$] ++(2,0)
    to[loudspeaker, name=L] ++(0,-2)
    to[short, -*] (0,0) node[ground]{};
    \node [waves, scale=0.7, left=5pt]
        at(M.north) {};
    \node [waves, scale=0.7, right]
        at(L.north) {};
\end{circuitikz}
```

3.11.1 Miscellanous element customization

You can change the scale of all the miscellaneous elements by setting the key `misc/scale` to something different from the default `1.0`.

3.12 Crossings

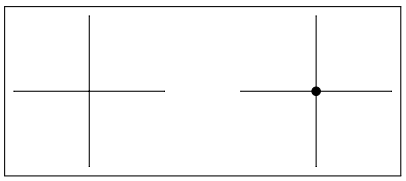
Path style:

	`crossing`: Jumper style non-contact crossing, `type: path-style, nodename: crossingshape.` Aliases: `xing`. Class: `default`.

Node style:

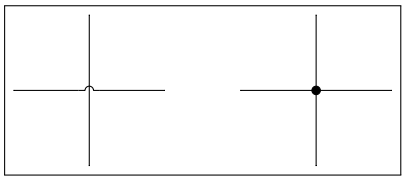
	Jumper-style crossing node, type: node (`node[jump crossing]{}`). No class.
	Plain style crossing node, type: node (`node[plain crossing]{}`). No class.

All circuit-drawing standards agree that to show a crossing without electric contact, a simple crossing of the wires suffices; the electrical contact must be explicitly marked with a filled dot.



```
\begin{circuitikz}[]
\draw(1,-1) to[short] (1,1)
   (0,0) to[short] (2,0);
\draw(4,-1) to[short] (4,1)
   (3,0) to[short] (5,0)
   (4,0) node[circ]{};
\end{circuitikz}
```

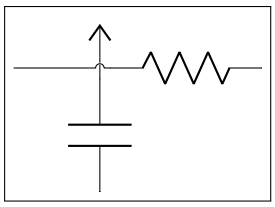
However, sometime it is advisable to mark the non-contact situation more explicitly. To this end, you can use a path-style component called `crossing`:



```
\begin{circuitikz}[]
\draw(1,-1) to[short] (1,1) (0,0) to[crossing]
    (2,0);
\draw(4,-1) to[short] (4,1) (3,0) to[short]
    (5,0)
   (4,0) node[circ]{};
\end{circuitikz}
```

That should suffice most of the time; the only problem is that the crossing jumper will be put in the center of the subpath where the `to[crossing]` is issued, so sometime a bit of trial and error is needed to position it.

For a more powerful (and elegant) way you can use the crossing nodes:



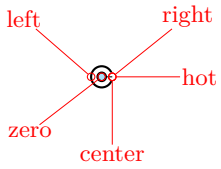
```
\begin{circuitikz}[]
   \node at (1,1)[jump crossing](X){};
   \draw (X.west) -- ++(-1,0);
   \draw (X.east) to[R] ++(2,0);
   \draw (X.north) node[vcc]{};
   \draw (X.south) to[C] ++(0,-1.5);
\end{circuitikz}
```

Notice that the `plain crossing` and the `jump crossing` have a small gap in the straight wire, to enhance the effect of crossing (as a kind of shadow).

The size of the crossing elements can be changed with the key `bipoles/crossing/size` (default 0.2).

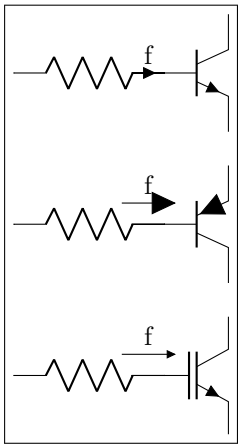
3.13 Arrows

These are pseudo-arrows used in lot of places in the packages (for transistors, flows, currents, and so on).

	Arrows (current and voltage), type: node (`node[currarrow]{}`). No class.
	Arrow to draw at its tip, useful for block diagrams., type: node (`node[inputarrow]{}`). No class.
	BNC connector, type: node, fillable (`node[bnc]{}`). No class.

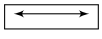
3.13.1 Arrows size

You can use the parameter `current arrow scale` to change the size of the arrows in various components and indicators; the normal value is 16, higher numbers give smaller arrows and so on. You need to use `circuitikz/current arrow scale` if you use it into a node.




```
\begin{circuitikz}
    \draw (0,0) to[R, i=f] ++(2,0) node[npn, anchor=B]{};
    \draw (0,-2) to[R, f=f, current arrow scale=8] ++(2,0)
           node[pnp, anchor=B, circuitikz/current arrow scale
               =8]{};
    \draw (0,-4) to[R, f=f, current arrow scale=24] ++(2,0)
           node[nigbt, anchor=B]{};
\end{circuitikz}
```

Moreover, you have the arrow tip `latexslim` which is an arrow similar to the old (in deprecated `arrows` library) `latex'` element:



```
\begin{circuitikz}[american,]
    \draw [latexslim-latexslim] (0,0) -- (1,0);
\end{circuitikz}
```

3.14 Terminal shapes

These are the so-called "bipole nodes" shapes, or poles (see section 4.6). These nodes are always filled; the "open" versions (starting with an `o`) are by default filled white, but you can override it with the `fill` parameter.

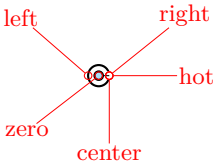
•	Connected terminal, type: node (`node[circ]{}`). No class.
○	Unconnected terminal, type: node (`node[ocirc]{}`). No class.
◆	Diamond-square terminal, type: node (`node[diamondpole]{}`). No class.
◇	Open diamond-square terminal, type: node (`node[odiamondpole]{}`). No class.
■	Square-shape terminal, type: node (`node[squarepole]{}`). No class.
□	Open square-shape terminal, type: node (`node[osquarepole]{}`). No class.

Since version 0.9.0, "bipole nodes" shapes have all the standard geographical anchors, so you can do things like these:

```
\begin{circuitikz}[american,]
    \draw (0,-1) node[draw](R){R};
    \draw (R.east) node[ocirc, right]{};
\end{circuitikz}
```

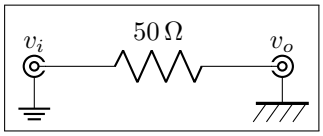
The size of the poles is controlled by the key `nodes width` (default `0.04`, relative to the basic length). Be sure to see section 4.6 for more usage and configurability.

3.14.1 BNC connector/terminal



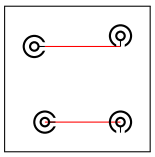
BNC connector, type: node, fillable (`node[bnc]{}`). No class.

The BNC connector is defined so that you can easily connect it as input or output (but remember that you need to flip the text if you flip the component):




```
\begin{circuitikz}
    \draw (0,0)
    node[bnc](B1){$v_i$} to[R=\SI{50}{\ohm}] ++(3,0)
    node[bnc, xscale=-1](B2){\scalebox{-1}[1]{$v_o$}};
    \node [ground] at (B1.shield) {};
    \node [eground] at (B2.shield){};
\end{circuitikz}
```

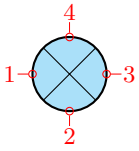
It also has a `zero` anchor if you need to rotate it about its real center.



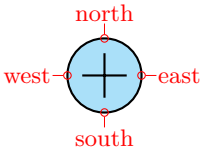
```
\begin{circuitikz}
    \draw[thin, red] (0,0) -- ++(1,0) (0,-1) -- ++(1,0);
    \path (0,0) node[bnc]{} ++(1,0) node[bnc, rotate=-90]{};
    \path (0,-1) node[bnc, anchor=zero]{} ++(1,0) node[bnc, anchor=
        zero, rotate=-90]{};
\end{circuitikz}
```

3.15 Block diagram components

Contributed by Stefan Erhardt.



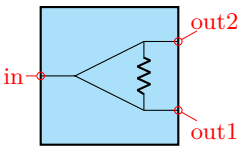
mixer, type: node, fillable (`node[mixer]{}`). Class: `blocks`.



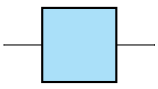
adder, type: node, fillable (`node[adder]{}`). Class: `blocks`.

oscillator, type: node, fillable (`node[oscillator]{}`). Class: `blocks`.

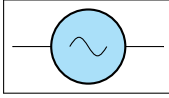
circulator, type: node, fillable (`node[circulator]{}`). Class: `blocks`.



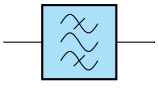
wilkinson divider, type: node, fillable (`node[wilkinson]{}`). Class: `blocks`.



`twoport`: generic two port[11], `type: path-style, fillable, nodename: twoportshape.` Class: `blocks`.

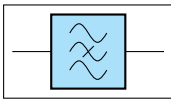


`vco`: vco, `type: path-style, fillable, nodename: vcoshape.` Class: `blocks`.

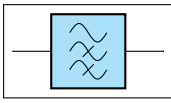


`bandpass`: bandpass, `type: path-style, fillable, nodename: bandpassshape.` Class: `blocks`.

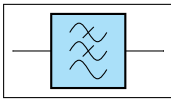
[11] To specify text to be put in the component: `twoport[t=text]`):



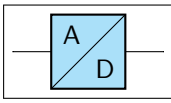
bandstop: bandstop, `type: path-style, fillable, nodename: bandstopshape.` Class: `blocks`.



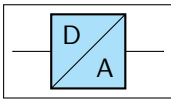
highpass: highpass, `type: path-style, fillable, nodename: highpassshape.` Class: `blocks`.



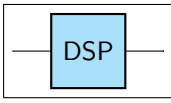
lowpass: lowpass, `type: path-style, fillable, nodename: lowpassshape.` Class: `blocks`.



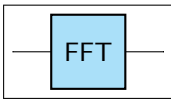
adc: A/D converter, `type: path-style, fillable, nodename: adcshape.` Class: `blocks`.



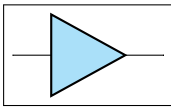
dac: D/A converter, `type: path-style, fillable, nodename: dacshape.` Class: `blocks`.



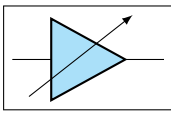
dsp: DSP, `type: path-style, fillable, nodename: dspshape.` Class: `blocks`.



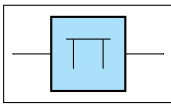
fft: FFT, `type: path-style, fillable, nodename: fftshape.` Class: `blocks`.



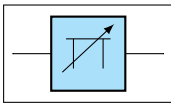
amp: amplifier, `type: path-style, fillable, nodename: ampshape.` Class: `blocks`.



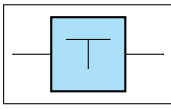
vamp: VGA, `type: path-style, fillable, nodename: vampshape.` Class: `blocks`.



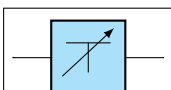
piattenuator: π attenuator, `type: path-style, fillable, nodename: piattenuatorshape.` Class: `blocks`.



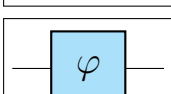
vpiattenuator: var. π attenuator, `type: path-style, fillable, nodename: vpiattenuatorshape.` Class: `blocks`.



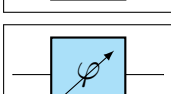
tattenuator: T attenuator, `type: path-style, fillable, nodename: tattenuatorshape.` Class: `blocks`.



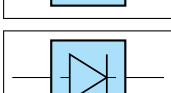
vtattenuator: var. T attenuator, `type: path-style, fillable, nodename: vtattenuatorshape.` Class: `blocks`.



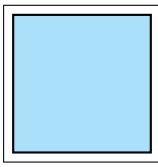
phaseshifter: phase shifter, `type: path-style, fillable, nodename: phaseshiftershape.` Class: `blocks`.



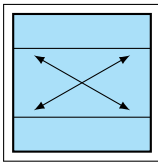
vphaseshifter: var. phase shifter, `type: path-style, fillable, nodename: vphaseshiftershape.` Class: `blocks`.



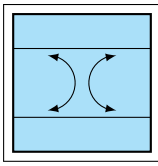
detector: detector, `type: path-style, fillable, nodename: detectorshape.` Class: `blocks`.



Generic fourport, type: node, fillable (`node[fourport]{}`). Class: `blocks`.



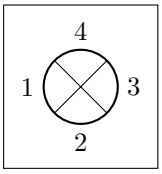
Coupler, type: node, fillable (`node[coupler]{}`). Class: `blocks`.



Coupler with rounded arrows, type: node, fillable (`node[coupler2]{}`). Class: `blocks`.

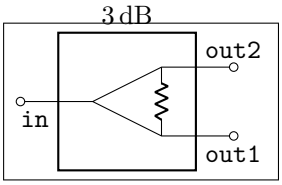
3.15.1 Blocks anchors

The ports of the mixer and adder can be addressed with numbers or `west`/`south`/`east`/`north`:



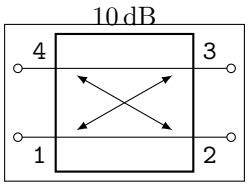

```
\begin{circuitikz} \draw
  (0,0) node[mixer] (mix) {}
  (mix.1) node[left] {1}
  (mix.2) node[below] {2}
  (mix.3) node[right] {3}
  (mix.4) node[above] {4}
;\end{circuitikz}
```

The Wilkinson divider has:

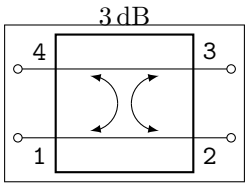



```
\begin{circuitikz} \draw
  (0,0) node[wilkinson] (w) {\SI{3}{dB}}
  (w.in) to[short,-o] ++(-0.5,0)
  (w.out1) to[short,-o] ++(0.5,0)
  (w.out2) to[short,-o] ++(0.5,0)
  (w.in) node[below left] {\texttt{in}}
  (w.out1) node[below right] {\texttt{out1}}
  (w.out2) node[above right] {\texttt{out2}}
  ;
\end{circuitikz}
```

The couplers have:




```
\begin{circuitikz} \draw
  (0,0) node[coupler] (c) {\SI{10}{dB}}
  (c.1) to[short,-o] ++(-0.5,0)
  (c.2) to[short,-o] ++(0.5,0)
  (c.3) to[short,-o] ++(0.5,0)
  (c.4) to[short,-o] ++(-0.5,0)
  (c.1) node[below left] {\texttt{1}}
  (c.2) node[below right] {\texttt{2}}
  (c.3) node[above right] {\texttt{3}}
  (c.4) node[above left] {\texttt{4}}
  ;
\end{circuitikz}
```

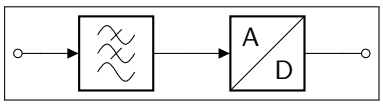



```
\begin{circuitikz} \draw
  (0,0) node[coupler2] (c) {\SI{3}{dB}}
  (c.1) to[short,-o] ++(-0.5,0)
  (c.2) to[short,-o] ++(0.5,0)
  (c.3) to[short,-o] ++(0.5,0)
  (c.4) to[short,-o] ++(-0.5,0)
  (c.1) node[below left] {\texttt{1}}
  (c.2) node[below right] {\texttt{2}}
  (c.3) node[above right] {\texttt{3}}
  (c.4) node[above left] {\texttt{4}}
  ;
\end{circuitikz}
```

3.15.2 Blocks customization

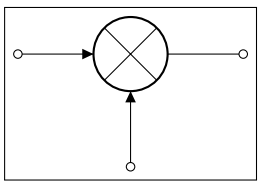
You can change the scale of all the block elements by setting the key `blocks/scale` to something different from the default `1.0`.

With the option `>` you can draw an arrow to the input of the block diagram symbols.



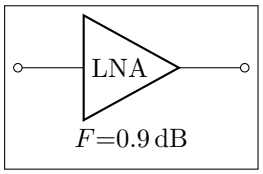

```
\begin{circuitikz} \draw
  (0,0) to[short,o-] ++(0.3,0)
  to[lowpass,>] ++(2,0)
  to[adc,>] ++(2,0)
  to[short,-o] ++(0.3,0);
\end{circuitikz}
```

3.15.2.1 Multi ports Since inputs and outputs can vary, input arrows can be placed as nodes. Note that you have to rotate the arrow on your own:



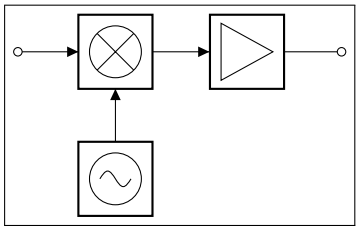
```
\begin{circuitikz} \draw
  (0,0) node[mixer] (m) {}
  (m.1) to[short,-o] ++(-1,0)
  (m.2) to[short,-o] ++(0,-1)
  (m.3) to[short,-o] ++(1,0)
  (m.1) node[inputarrow] {}
  (m.2) node[inputarrow,rotate=90] {};
\end{circuitikz}
```

3.15.2.2 Labels and custom two-port boxes Some two-ports have the option to place a normal label (`l=`) and a inner label (`t=`).



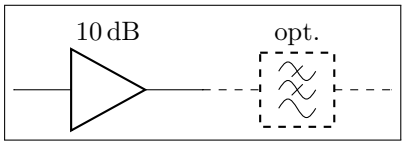

```
\begin{circuitikz}
 \ctikzset{bipoles/amp/width=0.9}
 \draw (0,0) to[amp,t=LNA,l_=$F{=}0.9\,$dB,o-o] ++(3,0);
\end{circuitikz}
```

3.15.2.3 Box option Some devices have the possibility to add a box around them. The inner symbol scales down to fit inside the box.



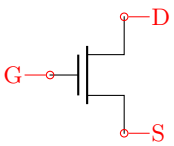
```
\begin{circuitikz} \draw
  (0,0) node[mixer,box,anchor=east] (m) {}
    to[amp,box,>,-o] ++(2.5,0)
  (m.west) node[inputarrow] {} to[short,-o]
      ++(-0.8,0)
  (m.south) node[inputarrow,rotate=90] {} --
    ++(0,-0.7) node[oscillator,box,anchor=north] {};
\end{circuitikz}
```

3.15.2.4 Dash optional parts To show that a device is optional, you can dash it. The inner symbol will be kept with solid lines.

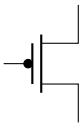



```
\begin{circuitikz}
 \draw (0,0) to[amp,l=\SI{10}{dB}] ++(2.5,0);
 \draw[dashed] (2.5,0) to[lowpass,l=opt.]
     ++(2.5,0);
\end{circuitikz}
```

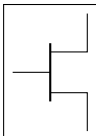
3.16 Transistors



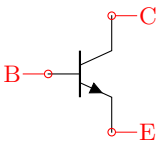
nmos, type: node (`node[nmos]{}`). Class: `transistors`.



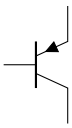
pmos, type: node (`node[pmos]{}`). Class: `transistors`.



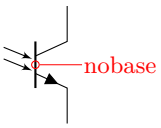
hemt, type: node (`node[hemt]{}`). Class: `transistors`.



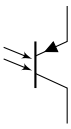
npn, type: node (`node[npn]{}`). Class: `transistors`.



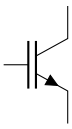
pnp, type: node (`node[pnp]{}`). Class: `transistors`.



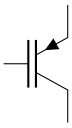
npn, type: node (`node[npn,photo]{}`). Class: `transistors`.



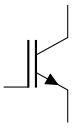
pnp, type: node (`node[pnp,photo]{}`). Class: `transistors`.



nigbt, type: node (`node[nigbt]{}`). Class: `transistors`.



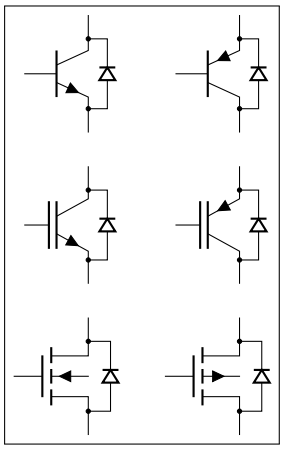
pigbt, type: node (`node[pigbt]{}`). Class: `transistors`.



Lnigbt, type: node (`node[Lnigbt]{}`). Class: `transistors`.

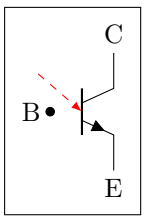
Lpigbt, type: node (`node[Lpigbt]{}`). Class: `transistors`.

For all transistors a body diode (or freewheeling diode) can automatically be drawn. Just use the global option bodydiode, or for single transistors, the tikz-option bodydiode:



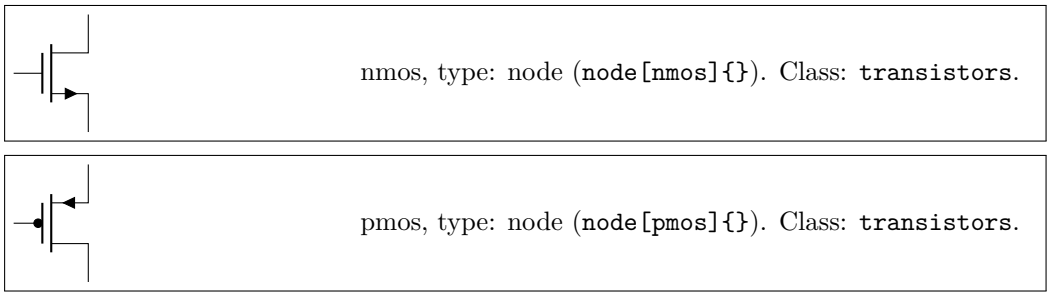
```
\begin{circuitikz}
  \draw (0,0) node[npn,bodydiode](npn){}++(2,0)node[pnp,
      bodydiode](npn){};
  \draw (0,-2) node[nigbt,bodydiode](npn){}++(2,0)node[
      pigbt,bodydiode](npn){};
  \draw (0,-4) node[nfet,bodydiode](npn){}++(2,0)node[
      pfet,bodydiode](npn){};
\end{circuitikz}
```

The Base/Gate connection of all transistors can be disabled by the options *nogate* or *nobase*, respectively. The Base/Gate anchors are floating, but there is an additional anchor ”nogate”/”nobase”, which can be used to point to the unconnected base:



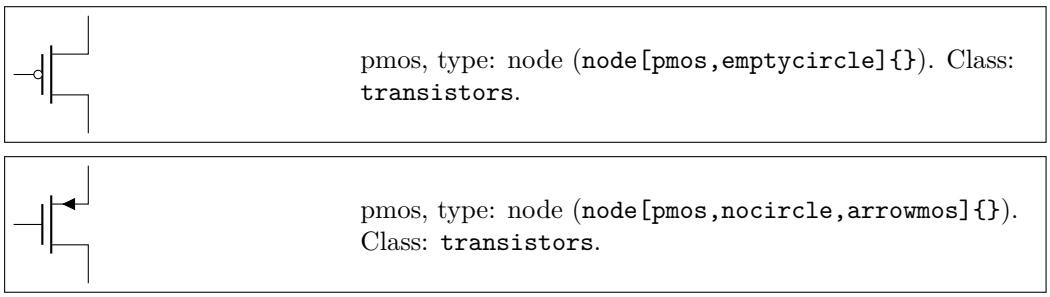

```
\begin{circuitikz}
  \draw (2,0) node[npn,nobase](npn){};
  \draw (npn.E) node[below]{E};
  \draw (npn.C) node[above]{C};
  \draw (npn.B) node[circ]{} node[left]{B};
  \draw[dashed,red,-latex] (1,0.5)--(npn.nobase);
\end{circuitikz}
```

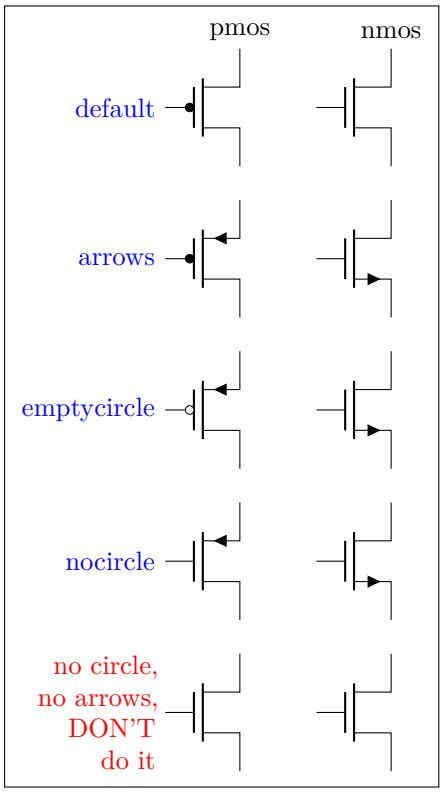
If the option `arrowmos` is used (or after the command `\ctikzset{tripoles/mos style/arrows}` is given), this is the output:



You can go back to the no-arrows mos with `noarrowmos` locally or with `\ctikzset{tripoles/mos style/no arrows}`.

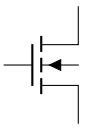
To draw the PMOS circle non-solid, use the option `emptycircle` or the command `\ctikzset{tripoles/pmos style/emptycircle}`. To remove the dot completely (only useful if you have `arrowmos` enabled, otherwise there will be no difference between P-MOS and N-MOS), you can use the option `nocircle` or `\ctikzset{tripoles/pmos style/nocircle}`.



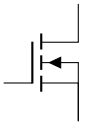



```
\begin{circuitikz}[
    info/.style={left=1cm, blue, text width=5
        em, align=right},]
    \draw (0,1) node{pmos} (2,1) node{nmos};
    \draw (0,0) node[info]{default} node[pmos
        ]{} (2,0) node[nmos]{};
    \ctikzset{tripoles/mos style/arrows}
    \draw (0,-2) node[info]{arrows} node[pmos
        ]{} (2,-2) node[nmos]{};
    \ctikzset{tripoles/pmos style/emptycircle}
    \draw (0,-4) node[info]{emptycircle} node[
        pmos]{} (2,-4) node[nmos]{};
    \ctikzset{tripoles/pmos style/nocircle}
    \draw (0,-6) node[info]{nocircle} node[
        pmos]{} (2,-6) node[nmos]{};
    \ctikzset{tripoles/mos style/no arrows}
    \draw (0,-8) node[info, red]{no circle, no
         arrows, DON'T do it}
         node[pmos]{} (2,-8) node[nmos]{};
\end{circuitikz}
```

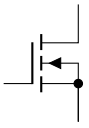
NFETs and PFETs have been incorporated based on code provided by Clemens Helfmeier and Theodor Borsche. Use the package options `fetsolderdot/nofetsolderdot` to enable/disable solderdot at some fet-transistors. Additionally, the solderdot option can be enabled/disabled for single transistors with the option "solderdot" and "nosolderdot", respectm ively.



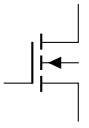
nfet, type: node (`node[nfet]{}`). Class: `transistors`.



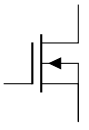
nigfete, type: node (`node[nigfete]{}`). Class: `transistors`.



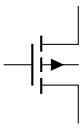
nigfete, type: node (`node[nigfete,solderdot]{}`). Class: `transistors`.



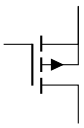
nigfetebulk, type: node (`node[nigfetebulk]{}`). Class: `transistors`.



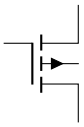
nigfetd, type: node (`node[nigfetd]{}`). Class: `transistors`.



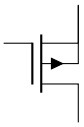
pfet, type: node (`node[pfet]{}`). Class: `transistors`.



pigfete, type: node (`node[pigfete]{}`). Class: `transistors`.

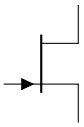


pigfetebulk, type: node (`node[pigfetebulk]{}`). Class: `transistors`.

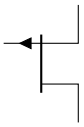


pigfetd, type: node (`node[pigfetd]{}`). Class: `transistors`.

NJFET and PJFET have been incorporated based on code provided by Danilo Piazzalunga:

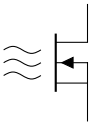


njfet, type: node (`node[njfet]{}`). Class: `transistors`.



pjfet, type: node (`node[pjfet]{}`). Class: `transistors`.

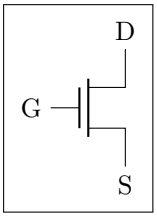
ISFET



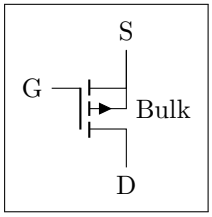
isfet, type: node (`node[isfet]{}`). Class: `transistors`.

3.16.1 Transistors anchors

For NMOS, PMOS, NFET, NIGFETE, NIGFETD, PFET, PIGFETE, and PIGFETD transistors one has `base`, `gate`, `source` and `drain` anchors (which can be abbreviated with `B`, `G`, `S` and `D`):

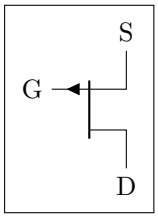



```
\begin{circuitikz} \draw
  (0,0) node[nmos] (mos) {}
  (mos.gate) node[anchor=east] {G}
  (mos.drain) node[anchor=south] {D}
  (mos.source) node[anchor=north] {S}
;\end{circuitikz}
```



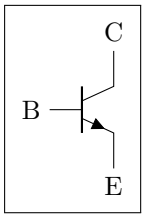

```
\begin{circuitikz} \draw
  (0,0) node[pigfete] (pigfete) {}
  (pigfete.G) node[anchor=east] {G}
  (pigfete.D) node[anchor=north] {D}
  (pigfete.S) node[anchor=south] {S}
  (pigfete.bulk) node[anchor=west] {Bulk}
;\end{circuitikz}
```

Similarly NJFET and PJFET have `gate`, `source` and `drain` anchors (which can be abbreviated with `G`, `S` and `D`):

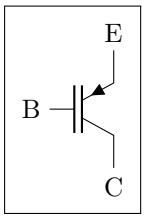



```
\begin{circuitikz} \draw
  (0,0) node[pjfet] (pjfet) {}
  (pjfet.G) node[anchor=east] {G}
  (pjfet.D) node[anchor=north] {D}
  (pjfet.S) node[anchor=south] {S}
;\end{circuitikz}
```

For NPN, PNP, NIGBT and PIGBT transistors, the anchors are `base`, `emitter` and `collector` anchors (which can be abbreviated with `B`, `E` and `C`):

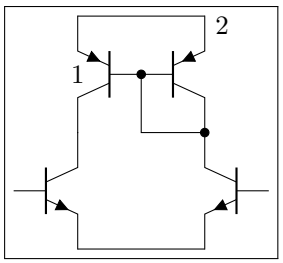



```
\begin{circuitikz} \draw
  (0,0) node[npn] (npn) {}
  (npn.base) node[anchor=east] {B}
  (npn.collector) node[anchor=south] {C}
  (npn.emitter) node[anchor=north] {E}
;\end{circuitikz}
```




```
\begin{circuitikz} \draw
  (0,0) node[pigbt] (pigbt) {}
  (pigbt.B) node[anchor=east] {B}
  (pigbt.C) node[anchor=north] {C}
  (pigbt.E) node[anchor=south] {E}
;\end{circuitikz}
```

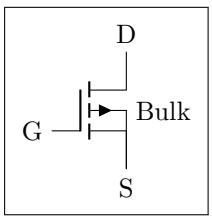
Here is one composite example (please notice that the `xscale=-1` style would also reflect the label of the transistors, so here a new node is added and its text is used, instead of that of `pnp1`):




```
\begin{circuitikz} \draw
  (0,0) node[pnp] (pnp2) {2}
  (pnp2.B) node[pnp, xscale=-1, anchor=B] (pnp1) {}
    (pnp1) node {1}
  (pnp1.C) node[npn, anchor=C] (npn1) {}
  (pnp2.C) node[npn, xscale=-1, anchor=C] (npn2) {}
  (pnp1.E) -- (pnp2.E) (npn1.E) -- (npn2.E)
  (pnp1.B) node[circ] {} |- (pnp2.C) node[circ] {}
;\end{circuitikz}
```

Notice that the text labels of transistors are somewhat buggy. It is better to se explicit anchors to set transistor's names.

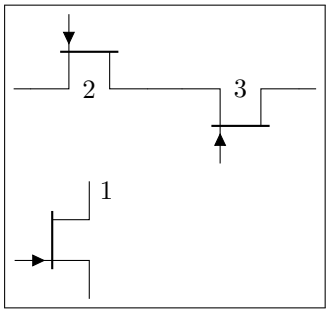
Similarly, transistors like other components can be reflected vertically:




```
\begin{circuitikz} \draw
  (0,0) node[pigfete, yscale=-1] (pigfete) {}
  (pigfete.bulk) node[anchor=west] {Bulk}
  (pigfete.G) node[anchor=east] {G}
  (pigfete.D) node[anchor=south] {D}
  (pigfete.S) node[anchor=north] {S}
;\end{circuitikz}
```

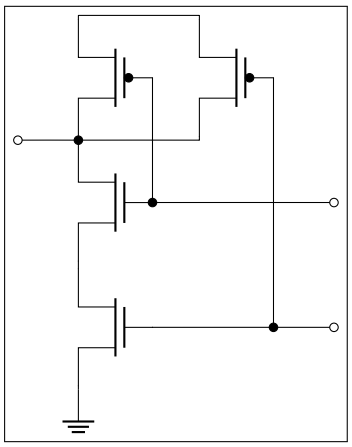
3.16.2 Transistor paths

For syntactical convenience transistors can be placed using the normal path notation used for bipoles. The transitor type can be specified by simply adding a "T" (for transistor) in front of the node name of the transistor. It will be placed with the base/gate orthogonal to the direction of the path:




```
\begin{circuitikz} \draw
  (0,0) node[njfet] {1}
  (-1,2) to[Tnjfet=2] (1,2)
    to[Tnjfet=3, mirror] (3,2);
;\end{circuitikz}
```

Access to the gate and/or base nodes can be gained by naming the transistors with the `n` or `name` path style:



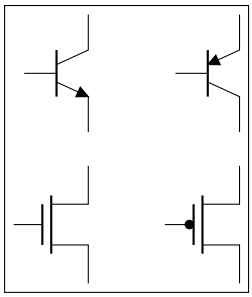
```
\begin{circuitikz} \draw[yscale=1.1, xscale=.8]
  (2,4.5) -- (0,4.5) to[Tpmos, n=p1] (0,3)
     to[Tnmos, n=n1] (0,1.5)
     to[Tnmos, n=n2] (0,0) node[ground] {}
  (2,4.5) to[Tpmos,n=p2] (2,3) to[short, -*] (0,3)
  (p1.G) -- (n1.G) to[short, *-o] ($(n1.G)+(3,0)$)
  (n2.G) ++(2,0) node[circ] {} -| (p2.G)
  (n2.G) to[short, -o] ($(n2.G)+(3,0)$)
  (0,3) to[short, -o] (-1,3)
;\end{circuitikz}
```

Transistor paths have the possibility to use the poles syntax (see section 4.6) but they have **no** voltage, current, flow, annotation options.

The `name` property is available also for bipoles; this is useful mostly for triac, potentiometer and thyristor (see 3.7.1).

3.16.3 Transistors customization

The default position of the arrows in transistors is somewhat in the middle of the terminal; if you prefer you can move them to the end with the style key `transistors/arrow pos=end` (the default value is `legacy`).

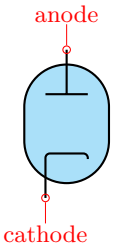


```
\begin{circuitikz}
  \ctikzset{transistors/arrow pos=end}
  \draw (0,0) node[npn, ](npn){};
  \draw (2,0) node[pnp, ](npn){};
  \draw (0,-2) node[nmos, ](npn){};
  \draw (2,-2) node[pmos, ](npn){};
\end{circuitikz}
```

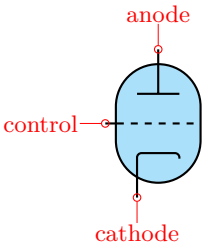
You can change the scale of all the transistors by setting the key `transistors/scale` (default `1.0`). The size of the arrows (if any) is controlled by the same parameters as `currarrow` (see section 3.13.1) and the dots on P-type transistors (if any) are the same as the nodes/poles (see section 4.6).

3.17 Electronic Tubes

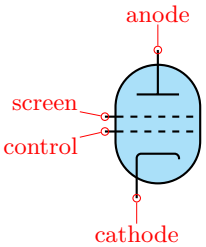
Electronic tubes, also known as vacuum tubes, control current flow between electrodes. They come in many different flavours. Contributed by J. op den Brouw (`J.E.J.opdenBrouw@hhs.nl`).



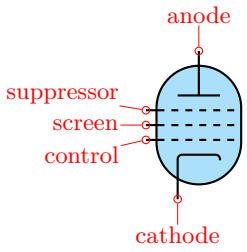
Tube Diode, type: node, fillable (`node[diodetube]{}`). Class: `tubes`.



Triode, type: node, fillable (`node[triode]{}`). Class: `tubes`.

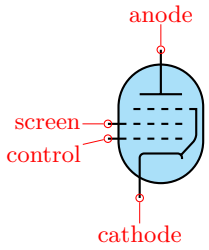


Tetrode, type: node, fillable (`node[tetrode]{}`). Class: `tubes`.



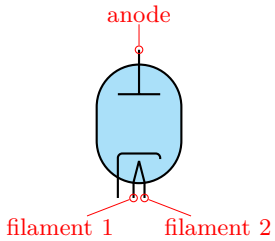
Pentode, type: node, fillable (`node[pentode]{}`). Class: `tubes`.

Some pentodes have the suppressor grid internally connected to the control grid, which saves a pin on the tube's housing.



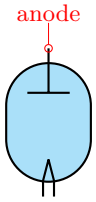
Pentode with suppressor grid connected to cathode, type: node, fillable (`node[pentode suppressor to cathode]{}`). Class: `tubes`.

Note that the `diodetube` is used as component name to avoid clashes with the semiconductor diode. Normally, the filament is not drawn. If you want a filament, put the `filament` option in the node description:



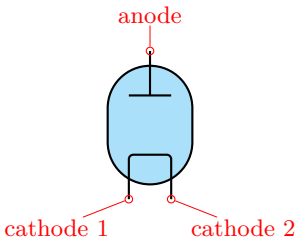
Tube Diode, type: node, fillable (`node[diodetube,filament]{}`). Class: `tubes`.

Sometimes, you don't want the cathode to be drawn (but you do want the filament). Use the `nocathode` option in the node description:



Tube Diode, type: node, fillable (`node[diodetube,filament,nocathode]{}`). Class: `tubes`.

If you want a full cathode to be drawn, use the `fullcathode` option in the node description. You can then use the anchors `cathode 1` and `cathode 2`.



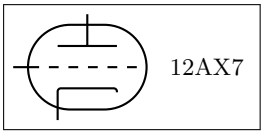
Tube Diode, type: node, fillable
(`node[diodetube,fullcathode]{}`). Class: `tubes`.

3.17.1 Tubes customization

The tubes can be scaled using the key `tubes/scale`, default `1.0`. In addition, they are fully configurable, and the attributes are described below:

Key	Default value	Description
`tubes/scale`	`1`	scale factor
`tubes/width`	`1`	relative width
`tubes/height`	`1.4`	relative height
`tubes/tube radius`	`0.40`	radius of tube circle
`tubes/anode distance`	`0.40`	distance from center
`tubes/anode width`	`0.40`	width of an anode/plate
`tubes/grid protrusion`	`0.25`	distance from center
`tubes/grid dashes`	`5`	number of grid dashes
`tubes/grid separation`	`0.2`	separation between grids
`tubes/grid shift`	`0.0`	y shift of grids from center
`tubes/cathode distance`	`0.40`	distance from grid
`tubes/cathode width`	`0.40`	width of a cathode
`tubes/cathode corners`	`0.06`	corners of the cathode wire
`tubes/cathode right extend`	`0.075`	extension at the right side
`tubes/filament distance`	`0.1`	distance from cathode
`tubes/filament angle`	`15`	angle from the centerpoint

Conventionally, the model of the tube is indicated at the `east` anchor:

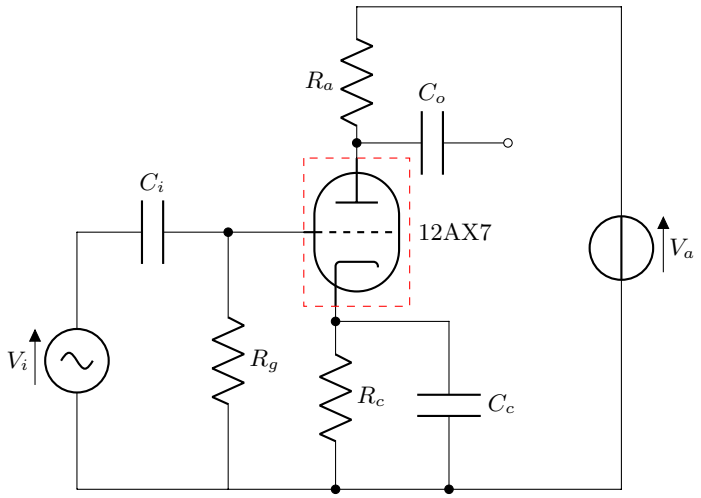



```
\ctikzset{tubes/width=1.4}
\ctikzset{tubes/height=1}

\begin{circuitikz}
\draw (0,0) node[triode] (Tri) {};
\draw (Tri.east) node[right] {12AX7};
\end{circuitikz}
```

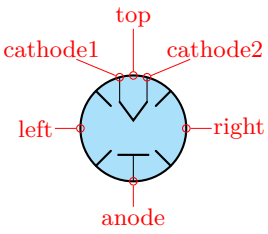
Example triode amplifier:

```
\begin{circuitikz}
\draw (0,0) node (start) {}
              to[sV=$V_i$] ++(0,2+\ctikzvalof{tubes/height})
              to[C=$C_i$] ++(2,0) node (Rg) {}
              to[R=$R_g$] (Rg |- start)
(Rg)          to[short,*-] ++(1,0)
              node[triode,anchor=control] (Tri) {} ++(2,0)
(Tri.cathode) to[R=$R_c$,-*] (Tri.cathode |- start)
(Tri.anode)   to [R=$R_a$] ++(0,2)
```

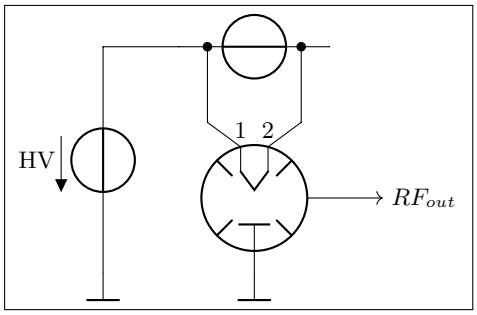
```
                to [short] ++(3.5,0) node(Vatop) {}
                to [V<=$V_a$] (Vatop |- start)
                to [short] (start)
(Tri.anode)   ++(0,0.2) to[C=$C_o$,*-o] ++(2,0)
(Tri.cathode) ++(0,-0.2) to[short,*-] ++(1.5,0) node(Cctop) {}
                to[C=$C_c$,-*] (start -| Cctop)
;
\draw[red,thin,dashed] (Tri.north west) rectangle (Tri.south east);
\draw (Tri.east) node[right] {12AX7};
\end{circuitikz}
```



The `magnetron` shape will also scale with `tubes/scale`.



Magnetron, type: node, fillable (`node[magnetron]{}`). Class: `tubes`.

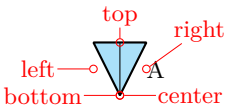



```
\begin{circuitikz}
\draw (0,-2)node[rground](gnd){} to[voltage
    source,v<={HV}]++(0,3)--++(1,0)to[V,n=
    DC]++(2,0);
\draw (2,-1) node[magnetron,scale=1](magn)
    {};
\draw (DC.left)++(-0.2,0)to [short,*-]
    ++(0,-1) to [short] (magn.cathode1);
\draw (DC.right)++(0.2,0)to [short,*-]
    ++(0,-1) to [short] (magn.cathode2);
\draw (magn.anode) to [short] (magn.anode|-
    gnd) node[rground]{};
\draw (magn.cathode1)node[above]{$1$};
\draw (magn.cathode2)node[above]{$2$};
\draw[->](magn.east) --++(1,0)node[right]{$
    RF_{out}$};
\end{circuitikz}
```

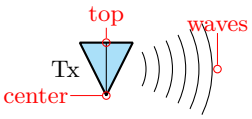
3.18 RF components

For the RF components, similarly to the grounds and supply rails, the `center` anchor is put on the connecting point of the symbol, so that you can use them directly in a `path` specification.

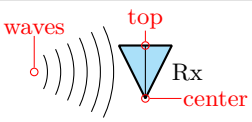
Notes that in the transmission and receiving antennas, the "waves" are outside the geographical anchors.



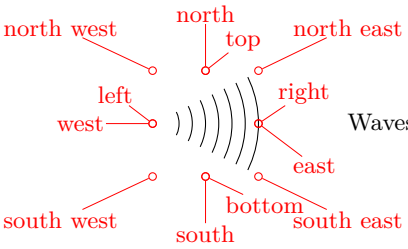
Bare Antenna, type: node, fillable (`node[bareantenna]{A}`). Class: `RF`.



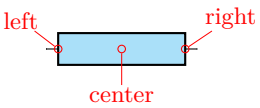
Bare TX Antenna, type: node, fillable (`node[bareTXantenna]{Tx}`). Class: `RF`.



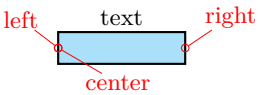
Bare RX Antenna, type: node, fillable (`node[bareRXantenna]{Rx}`). Class: `RF`.



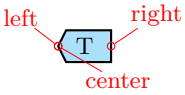
Waves, type: node (`node[waves]{}`). Class: `RF`.



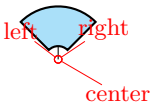
`mstline`: Microstrip transmission line[12], `type: path-style, fillable, nodename: mstlineshape.` Class: `RF`.



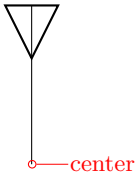
Microstrip linear stub, type: node, fillable (`node[mslstub]{text}`). Class: `RF`.



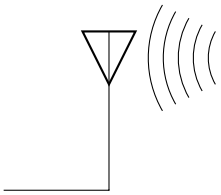
Microstrip port, type: node, fillable (`node[msport]{T}`). Class: `RF`.



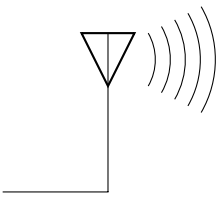
Microstrip radial stub, type: node, fillable (`node[msrstub]{}`). Class: `RF`.



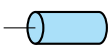
Legacy antenna (with tails), type: node (`node[antenna]{}`). Class: `RF`.



Legacy receiving antenna (with tails), type: node (`node[rxantenna]{}`). Class: `RF`.



Legacy transmitting antenna (with tails), type: node (`node[txantenna]{}`). Class: `RF`.



Transmission line stub, type: node, fillable (`node[tlinestub]{}`). Class: `RF`.

`TL`: Transmission line, `type: path-style, fillable, nodename: tlineshape`. Aliases: `transmission line, tline`. Class: `RF`.

match, type: node (`node[match]{}`). Class: `RF`.

3.18.1 RF elements customization

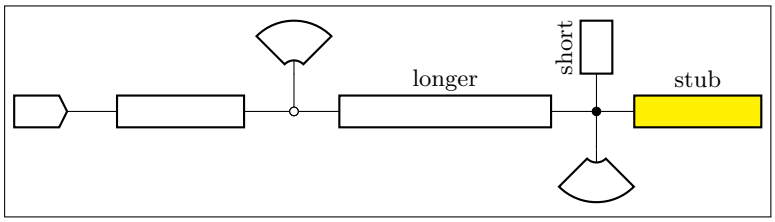
The RF elements can be scaled using the key `RF/scale`, default `1.0`.

3.18.2 Microstrip customization

The microstrip linear components' (`mstline`, `mslstub`, `msport`) heights can be changed by setting the parameter `bipoles/mstline/height` (for the three of them, default 0.3). The widths are specified in `bipoles/mstline/width` for the first two and by `monopoles/msport/width` for the port (defaults: 1.2, 0.5).

For the length parameter of the transmission line there is a shortcut in the form of the direct parameter `mstlinelen`.

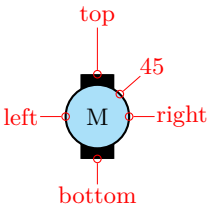
[12] This four components were suggested by `@tcpluess` on GitHub



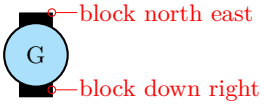

```
\begin{circuitikz}
    \draw (0,0) node[msport, right, xscale=-1]{}
    to[mstline, -o] ++(3,0) coordinate(there)
    to[mstline, mstlinelen=2, l=longer, o-*] ++(4,0)
        coordinate(here) -- ++(0.5,0) node[mslstub, fill=yellow]{stub}
        (here) -- ++(0,0.5) node[mslstub, rotate=90, mstlinelen=0.5]{short};
        \draw (there) to[short, o-] ++(0, 0.5) node[msrstub]{};
        \draw (here) -- ++(0, -0.5) node[msrstub, yscale=-1]{};
\end{circuitikz}
```

3.19 Electro-Mechanical Devices

The internal part of the motor and generator are, by default, filled white (to avoid compatibility problems with older versions of the package).



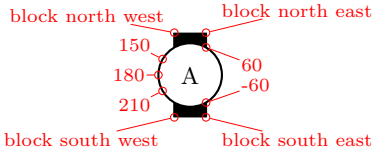
Motor, type: node, fillable (`node[elmech]{M}`). Class: `electromechanicals`.

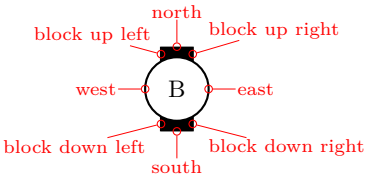


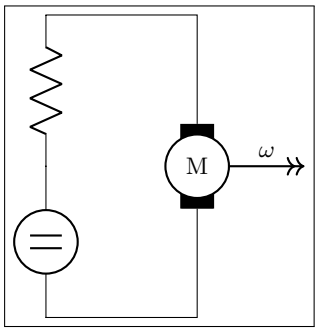
Generator, type: node, fillable (`node[elmech]{G}`). Class: `electromechanicals`.

3.19.1 Electro-Mechanical Devices anchors

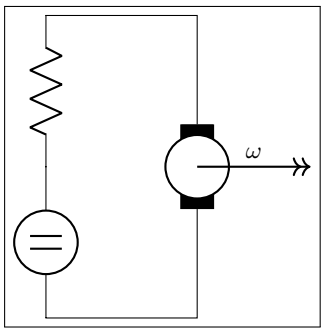
Apart from the standard geographical anchors, `elmech` has the border anchors (situated on the inner circle) and the following anchors on the "block":





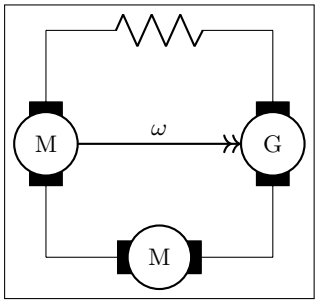



```
\begin{circuitikz}
\draw (2,0) node[elmech](motor){M};
\draw (motor.north) |-(0,2) to [R] ++(0,-2) to[
    dcvsource]++(0,-2) -| (motor.bottom);
\draw[thick,->>](motor.right)--++(1,0)node[midway,
    above]{$\omega$};
\end{circuitikz}
```




```
\begin{circuitikz}
\draw (2,0) node[elmech](motor){};
\draw (motor.north) |-(0,2) to [R] ++(0,-2) to[
    dcvsource]++(0,-2) -| (motor.bottom);
\draw[thick,->>](motor.center)--++(1.5,0)node[midway,
    above]{$\omega$};
\end{circuitikz}
```

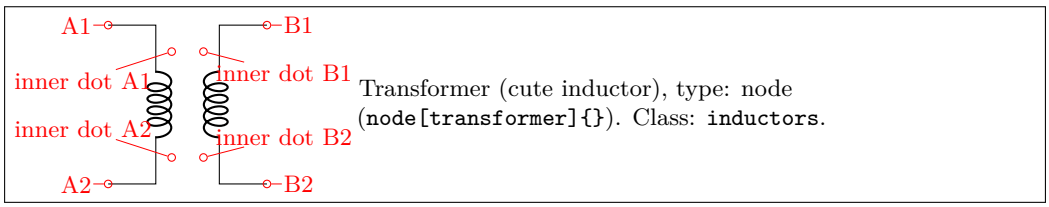
The symbols can also be used along a path, using the transistor-path-syntax(`T` in front of the shape name, see section 3.16.2). Don't forget to use parameter n to name the node and get access to the anchors:



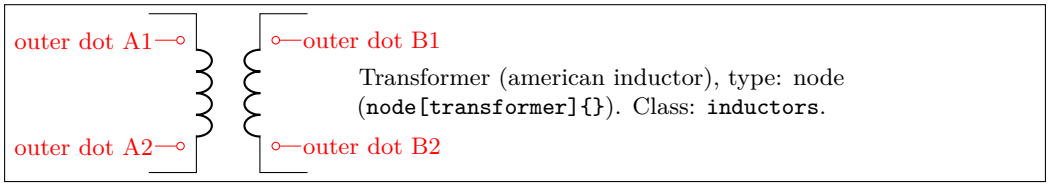

```
\begin{circuitikz}
\draw (0,0) to [Telmech=M,n=motor] ++(0,-3) to [
    Telmech=M] ++(3,0) to [Telmech=G,n=generator]
    ++(0,3) to [R] (0,0);
\draw[thick,->>](motor.left)--(generator.left)node[
    midway,above]{$\omega$};
\end{circuitikz}
```

3.20 Double bipoles (transformers)

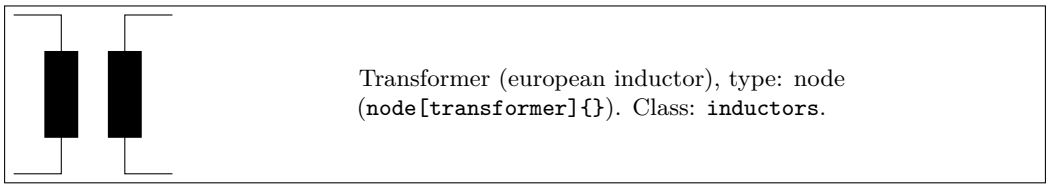
Transformers automatically use the inductor shape currently selected. These are the three possibilities:



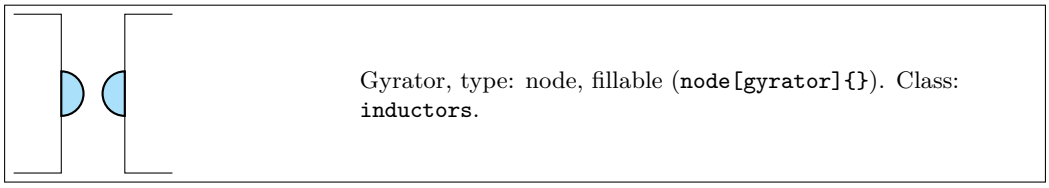
Transformer (cute inductor), type: node (`node[transformer]{}`). Class: `inductors`.



Transformer (american inductor), type: node (`node[transformer]{}`). Class: `inductors`.

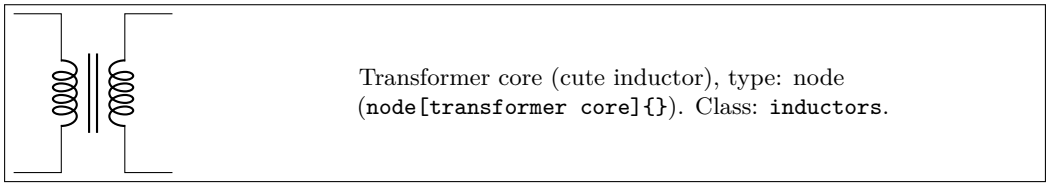


Transformer (european inductor), type: node (`node[transformer]{}`). Class: `inductors`.

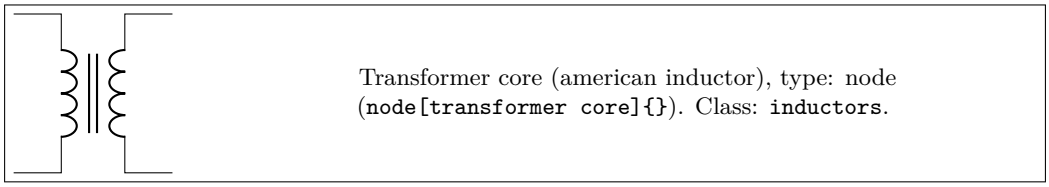


Gyrator, type: node, fillable (`node[gyrator]{}`). Class: `inductors`.

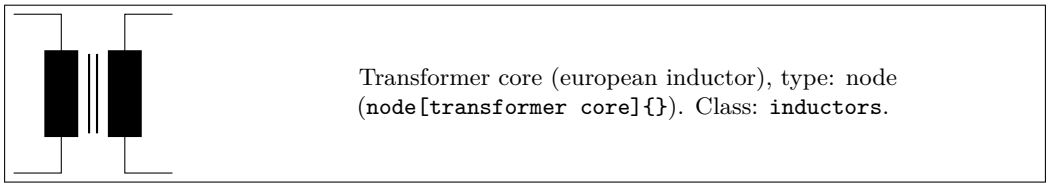
Transformers with core are also available:



Transformer core (cute inductor), type: node (`node[transformer core]{}`). Class: `inductors`.



Transformer core (american inductor), type: node (`node[transformer core]{}`). Class: `inductors`.

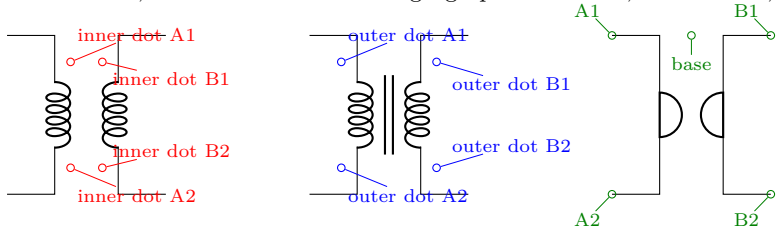


Transformer core (european inductor), type: node (`node[transformer core]{}`). Class: `inductors`.

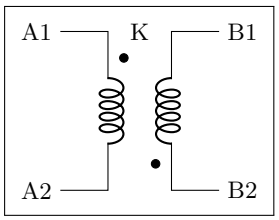
3.20.1 Double dipoles anchors

All the double bipoles/quadrupoles have the four anchors, two for each port. The first port, to the left, is port `A`, having the anchors `A1` (up) and `A2` (down); same for port `B`.

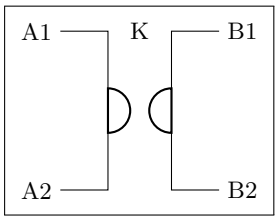
They also expose the `base` anchor, for labelling, and anchors for setting dots or signs to specify polarity. The set of anchors, to which the standard "geographical" `north`, `north east`, etc. is here:



Also, the standard "geographical" `north`, `north east`, etc. are defined. A couple of examples follow:



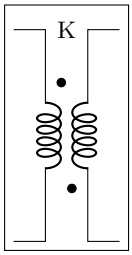

```
\begin{circuitikz} \draw
  (0,0) node[transformer] (T) {}
  (T.A1) node[anchor=east] {A1}
  (T.A2) node[anchor=east] {A2}
  (T.B1) node[anchor=west] {B1}
  (T.B2) node[anchor=west] {B2}
  (T.base) node{K}
  (T.inner dot A1) node[circ]{}
  (T.inner dot B2) node[circ]{}
;\end{circuitikz}
```




```
\begin{circuitikz} \draw
  (0,0) node[gyrator] (G) {}
  (G.A1) node[anchor=east] {A1}
  (G.A2) node[anchor=east] {A2}
  (G.B1) node[anchor=west] {B1}
  (G.B2) node[anchor=west] {B2}
  (G.base) node{K}
;\end{circuitikz}
```

3.20.2 Double dipoles customization

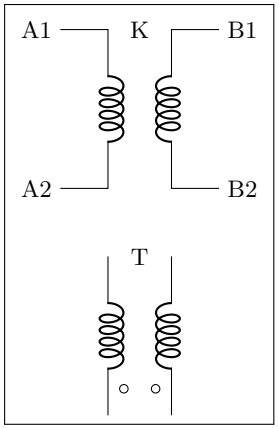
Transformers are in the `inductors` class (also the gyrator…), so they scale with the key `inductors/scale`. You can change the aspect of a quadpole using the corresponding parameters `quadpoles/*/width` and `quadpoles/*/heigth` (substitute the star for `transformer`, `transformer core` or `gyrator`; default value is `1.5` for all). You have to be careful to not choose value that overlaps the components!




```
\begin{circuitikz}
\ctikzset{quadpoles/transformer/width=1,
    quadpoles/transformer/height=2}
  \draw (0,0) node[transformer] (T) {}
    (T.base) node{K}
    (T.inner dot A1) node[circ]{}
    (T.inner dot B2) node[circ]{};
\end{circuitikz}
```

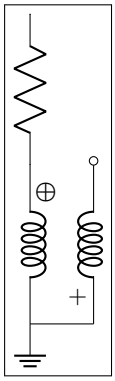
Transformers also inherits the `inductors/scale` (see 3.6.6) and similar parameters. It's your responsibility to set the aforementioned parameters if you change the scale or width of inductors.

Another very useful parameter is `quadpoles/*/inner` (default `0.4`) that determine which part of the component is the "vertical" one. So, setting that parameter to 1 will eliminate the horizontal part of the component (obviously, to maintain the general aspect ratio you need to change the width also):



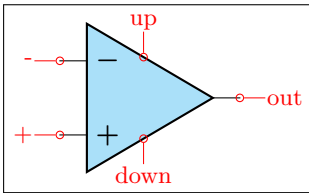

```
\begin{circuitikz}
\draw (0,0) node[transformer] (T) {}
  (T.A1) node[anchor=east] {A1}
  (T.A2) node[anchor=east] {A2}
  (T.B1) node[anchor=west] {B1}
  (T.B2) node[anchor=west] {B2}
  (T.base) node{K} ;
\ctikzset{quadpoles/transformer/inner=1, quadpoles/
     transformer/width=0.6}
\draw (0,-3) node[transformer] (P) {}
  (P.base) node{T}
  (P.inner dot A2) node[ocirc]{}
  (P.inner dot B2) node[ocirc]{};
\end{circuitikz}
```

This can be useful if you want to put seamlessly something in series with either side of the component; for simplicity, you have a style setting `quadpoles style` to toggle between the standard shape of double bipoles (called `inward`, default) and the one without horizontal leads (called `inline`):

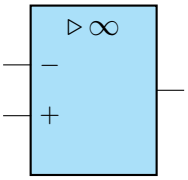


```
\begin{circuitikz}
\ctikzset{inductor=cute, quadpoles style=inline}
\draw
  (0,0) to[R] ++(0,-2)
  node[transformer, anchor=A1](T){}
  (T.A2) node[ground](GND){}
  (T.inner dot A1) node[font=\small\boldmath]{$\oplus$}
  (T.inner dot B2) node[]{$+$}
  (T.B1) node[above, ocirc]{}
  (T.B2) -- (GND);
\end{circuitikz}
```

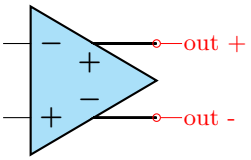
3.21 Amplifiers



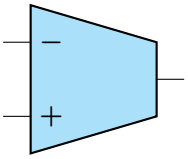
Operational amplifier, type: node, fillable (`node[op amp]{}`). Class: `amplifiers`.



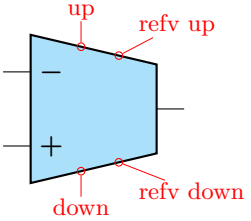
Operational amplifier compliant to DIN/EN 60617 standard, type: node, fillable (`node[en amp]{}`). Class: `amplifiers`.



Fully differential operational amplifier[13], type: node, fillable (`node[fd op amp]{}`). Class: `amplifiers`.

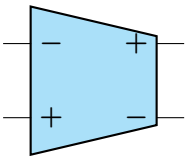


transconductance amplifier, type: node, fillable (`node[gm amp]{}`). Class: `amplifiers`.

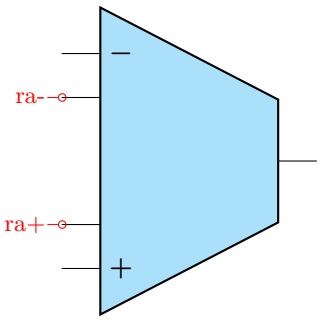


plain instrumentation amplifier, type: node, fillable (`node[inst amp]{}`). Class: `amplifiers`.

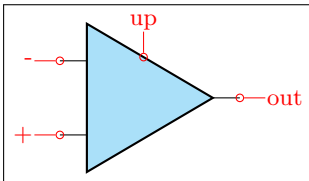
[13] Contributed by Kristofer M. Monisit.



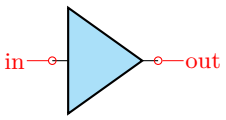
Fully differential instrumentation amplifier, type: node, fillable (`node[fd inst amp]{}`). Class: `amplifiers`.



instrumentation amplifier with amplification resistance terminals, type: node, fillable (`node[inst amp ra]{}`). Class: `amplifiers`.



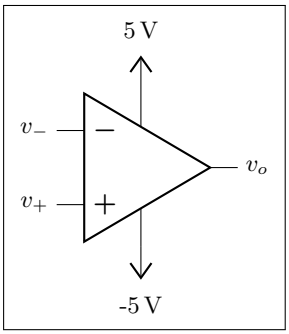
Plain amplifier, type: node, fillable (`node[plain amp]{}`). Class: `amplifiers`.



Buffer, type: node, fillable (`node[buffer]{}`). Class: `amplifiers`.

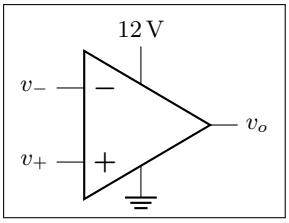
3.21.1 Amplifiers anchors

The op amp defines the inverting input (`-`), the non-inverting input (`+`) and the output (`out`) anchors:



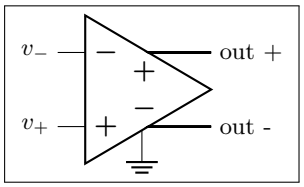

```
\begin{circuitikz} \draw
  (0,0) node[op amp] (opamp) {}
  (opamp.+) node[left] {$v_+$}
  (opamp.-) node[left] {$v_-$}
  (opamp.out) node[right] {$v_o$}
  (opamp.up) --++(0,0.5) node[vcc]{5\,\textnormal{V}}
  (opamp.down) --++(0,-0.5) node[vee]{-5\,\textnormal{V}}
;\end{circuitikz}
```

There are also two more anchors defined, `up` and `down`, for the power supplies:



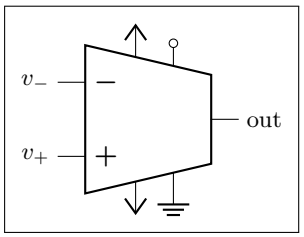

```
\begin{circuitikz} \draw
  (0,0) node[op amp] (opamp) {}
  (opamp.+) node[left] {$v_+$}
  (opamp.-) node[left] {$v_-$}
  (opamp.out) node[right] {$v_o$}
  (opamp.down) node[ground] {}
  (opamp.up) ++ (0,.5) node[above] {\SI{12}{\volt}}
    -- (opamp.up)
;\end{circuitikz}
```

The fully differential op amp defines two outputs:



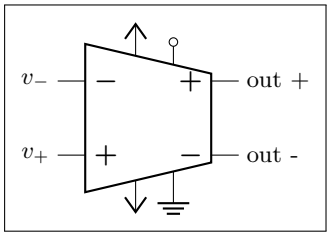

```
\begin{circuitikz} \draw
  (0,0) node[fd op amp] (opamp) {}
  (opamp.+) node[left] {$v_+$}
  (opamp.-) node[left] {$v_-$}
  (opamp.out +) node[right] {out +}
  (opamp.out -) node[right] {out -}
  (opamp.down) node[ground] {}
;\end{circuitikz}
```

The instrumentation amplifier inst amp defines also references (normally you use the ”down”, unless you are flipping the component):



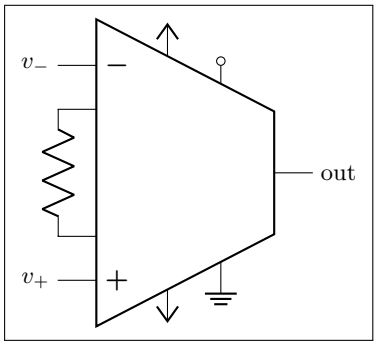

```
\begin{circuitikz} \draw
  (0,0) node[inst amp] (opamp) {}
  (opamp.+) node[left] {$v_+$}
  (opamp.-) node[left] {$v_-$}
  (opamp.out) node[right] {out}
  (opamp.up) node[vcc]{}
  (opamp.down) node[vee] {}
  (opamp.refv down) node[ground]{}
  (opamp.refv up) to[short, -o] ++(0,0.3)
;\end{circuitikz}
```

The fully diffential instrumentation amplifier inst amp defines two outputs:




```
\begin{circuitikz} \draw
  (0,0) node[fd inst amp] (opamp) {}
  (opamp.+) node[left] {$v_+$}
  (opamp.-) node[left] {$v_-$}
  (opamp.out +) node[right] {out +}
  (opamp.out -) node[right] {out -}
  (opamp.up) node[vcc]{}
  (opamp.down) node[vee] {}
  (opamp.refv down) node[ground]{}
  (opamp.refv up) to[short, -o] ++(0,0.3)
;\end{circuitikz}
```

The instrumentation amplifier with resistance terminals (`inst amp ra`) defines also terminals to add an amplification resistor:

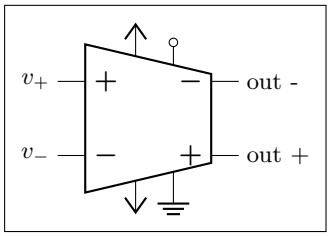



```
\begin{circuitikz} \draw
  (0,0) node[inst amp ra] (opamp) {}
  (opamp.+) node[left] {$v_+$}
  (opamp.-) node[left] {$v_-$}
  (opamp.out) node[right] {out}
  (opamp.up) node[vcc]{}
  (opamp.down) node[vee] {}
  (opamp.refv down) node[ground]{}
  (opamp.refv up) to[short, -o] ++(0,0.3)
  (opamp.ra-) to[R] (opamp.ra+)
;\end{circuitikz}
```

3.21.2 Amplifiers customization

You can scale the amplifiers using the key `amplifiers/scale` and setting it to something different from `1.0`. The font used for symbols will not scale, so it's your responsibility to change it if the need arises.

All these amplifier have the possibility to flip input and output (if needed) polarity. You can change polarity of the input with the `noinv input down` (default) or `noinv input up` key; and the output with `noinv output up` (default) or `noinv output down` key:




```
\begin{circuitikz} \draw
  (0,0) node[fd inst amp,
    noinv input up,
    noinv output down] (opamp) {}
  (opamp.+) node[left] {$v_+$}
  (opamp.-) node[left] {$v_-$}
  (opamp.out +) node[right] {out +}
  (opamp.out -) node[right] {out -}
  (opamp.up) node[vcc]{}
  (opamp.down) node[vee] {}
  (opamp.refv down) node[ground]{}
  (opamp.refv up) to[short, -o] ++(0,0.3)
;\end{circuitikz}
```

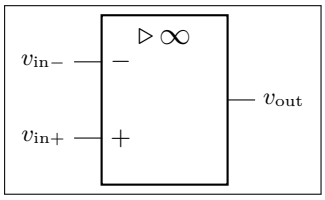
When you use the `noinv input/output ...` keys the anchors (`+`, `-`, `out +`, `out -`) will change with the effective position of the terminals. You have also the anchors `in up`, `in down`, `out up`, `out down` that will not change with the positive or negative sign.

3.21.2.1 European-style amplifier customization

Thanks to the suggestions from David Rouvel (david.rouvel@iphc.cnrs.fr) there are several possible customization for the European-style amplifiers.

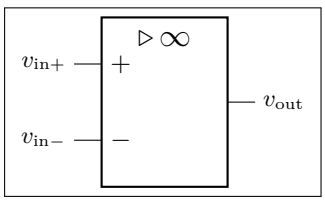
Since 0.9.0, the default appearance of the symbol has changed to be more in line with the standard; notice that to have a bigger triangle by default we should require more packages, and I fear ConTEXt compatibility; but see later on how to change it. Notice that the font used for the symbol is defined in `tripoles/en amp/font2` and that the font used for the `+` and `-` symbols is `tripoles/en amp/font`.

You can change the distances of the inputs, using `tripoles/en amp/input height` (default 0.3):



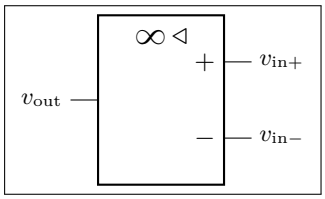

```
\begin{circuitikz}
   \ctikzset{tripoles/en amp/input height=0.45}
   \draw (0,0)node[en amp](E){}
      (E.out) node[right] {$v_{\mathrm{out}}$}
      (E.-) node[left] {$v_{\mathrm{in}-}$}
      (E.+) node[left] {$v_{\mathrm{in}+}$};
\end{circuitikz}
```

and of course the key `noinv input up` is fully functional:




```
\begin{circuitikz}
   \ctikzset{tripoles/en amp/input height=0.45}
   \draw (0,0)node[en amp, noinv input up](E){}
      (E.out) node[right] {$v_{\mathrm{out}}$}
      (E.-) node[left] {$v_{\mathrm{in}-}$}
      (E.+) node[left] {$v_{\mathrm{in}+}$};
\end{circuitikz}
```

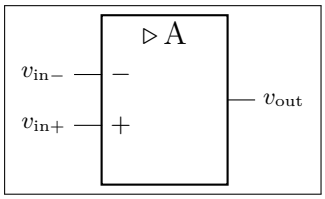
To flip the amplifier in the horizontal direction, you can use `xscale=-1` as usual:




```
\begin{circuitikz}
    \ctikzset{tripoles/en amp/input height=0.45}
    \draw (0,0)node[en amp, xscale=-1, noinv input up
        ](E){}
        (E.out) node[left] {$v_{\mathrm{out}}$}
        (E.-) node[right] {$v_{\mathrm{in}-}$}
        (E.+) node[right] {$v_{\mathrm{in}+}$};
\end{circuitikz}
```

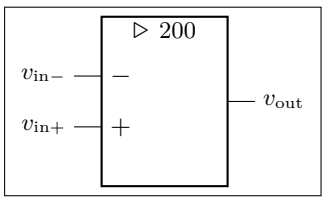
Notice that the label is fully mirrored, so check below for the generic way to change this.

You can use the new key `en amp text A` to change the infinity symbol with an A:




```
\begin{circuitikz}
    \draw (0,0)node[en amp, en amp text A](E){}
        (E.out) node[right] {$v_{\mathrm{out}}$}
        (E.-) node[left] {$v_{\mathrm{in}-}$}
        (E.+) node[left] {$v_{\mathrm{in}+}$} ;
\end{circuitikz}
```

And if you want, you can completely change the text using the key `en amp text=`, which by default is `$\mathstrut{\triangleright}\,{\infty}$`:

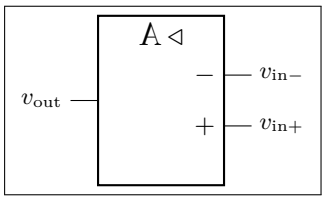



```
\begin{circuitikz}
    \draw (0,0)node[en amp, en amp text={%
            ${\triangleright}$ \small 200}](E){}
        (E.out) node[right] {$v_{\mathrm{out}}$}
        (E.-) node[left] {$v_{\mathrm{in}-}$}
        (E.+) node[left] {$v_{\mathrm{in}+}$} ;
\end{circuitikz}
```

Notice two things here: the first, that `\triangleright` is enclosed in braces to remove the default spacing it has as a binary operator, and that `en amp text A` is simply a shortcut for

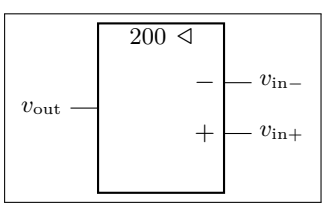
```
    en amp text={$\mathstrut{\triangleright}\,\mathrm{A}$}
```

To combine flipping with a generic label you just do:




```
\begin{circuitikz}
    \draw (0,0)node[en amp, xscale=-1, en amp text A](
        E){}
        (E.out) node[left] {$v_{\mathrm{out}}$}
        (E.-) node[right] {$v_{\mathrm{in}-}$}
        (E.+) node[right] {$v_{\mathrm{in}+}$} ;
\end{circuitikz}
```

But notice that the "A" is also flipped by the `xscale` parameter. So the solution in this case is to use `scalebox`, like this:




```
\begin{circuitikz}
    \draw (0,0)node[en amp, xscale=-1, en amp text={%
    ${\triangleright}$ \scalebox{-1}[1]{\small 200}}](
        E){}
        (E.out) node[left] {$v_{\mathrm{out}}$}
        (E.-) node[right] {$v_{\mathrm{in}-}$}
        (E.+) node[right] {$v_{\mathrm{in}+}$} ;
\end{circuitikz}
```

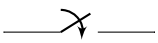
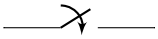
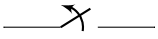
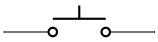
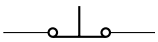
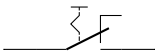
3.22 Switches and buttons

Switches and button come in to-style (the simple ones and the pushbuttons), and as nodes.

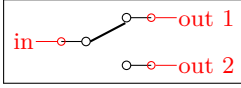
The switches can be scaled with the key `switches/scale` (default `1.0`). Notice that scaling the switches will not scale the poles, which are controlled with their own parameters (see section 3.14).

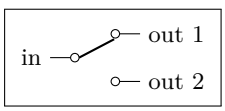
3.22.1 Traditional switches

These are all of the to-style type:

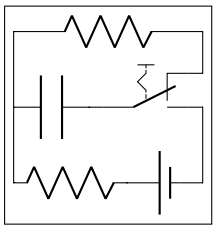
	**switch**: Switch, `type: path-style, nodename: cspstshape`. Aliases: `spst`. Class: `switches`.
	**closing switch**: Closing switch, `type: path-style, nodename: cspstshape`. Aliases: `cspst`. Class: `switches`.
	**opening switch**: Opening switch, `type: path-style, nodename: ospstshape`. Aliases: `ospst`. Class: `switches`.
	normal open switch: Normally open switch, `type: path-style, nodename: nosshape`. Aliases: `nos`. Class: `switches`.
	normal closed switch: Normally closed switch, `type: path-style, nodename: ncsshape`. Aliases: `ncs`. Class: `switches`.
	**push button**: Normally open push button, `type: path-style, nodename: pushbuttonshape`. Aliases: `normally open push button`, `nopb`. Class: `switches`.
	**normally closed push button**: Normally closed push button, `type: path-style, nodename: ncpushbuttonshape`. Aliases: `ncpb`. Class: `switches`.
	**toggle switch**: Toggle switch, `type: path-style, nodename: toggleswitchshape`. Class: `default`.

while this is a node-style component:

	spdt, type: node (`node[spdt]{}`). Class: `switches`.




```
\begin{circuitikz} \draw
  (0,0) node[spdt] (Sw) {}
  (Sw.in) node[left] {in}
  (Sw.out 1) node[right] {out 1}
  (Sw.out 2) node[right] {out 2}
;\end{circuitikz}
```

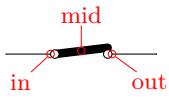
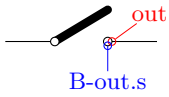


```
\begin{circuitikz} \draw
 (0,0) to[C] (1,0) to[toggle switch , n=Sw] (2.5,0)
   -- (2.5,-1) to[battery1] (1.5,-1) to[R] (0,-1) -| (0,0)
  (Sw.out 2) -| (2.5, 1) to[R] (0,1) -- (0,0)
;\end{circuitikz}
```

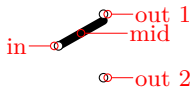
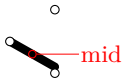
3.22.2 Cute switches

These switches have been introduced after version 0.9.0, and they come in also in to-style and in node-style, but they are size-matched so that they can be used together in a seamless way.

The path element (to-style) are:

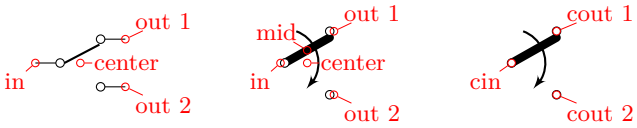
	**cute closed switch**: Cute closed switch, `type: path-style, nodename: cuteclosedswitchshape.` Aliases: `ccsw`. Class: `switches`.
	**cute open switch**: Cute open switch, `type: path-style, name=B, nodename: cuteopenswitchshape.` Aliases: `cosw`. Class: `switches`.
	cute closing switch: Cute closing switch, `type: path-style, nodename: cuteclosingswitchshape.` Aliases: `ccgsw`. Class: `switches`.
	cute opening switch: Cute opening switch, `type: path-style, nodename: cuteopeningswitchshape.` Aliases: `cogsw`. Class: `switches`.

while the node-style components are the single-pole, double-throw (`spdt`) ones:

	Cute spdt up, type: node (`node[cute spdt up]{}`). Class: `switches`.
	Cute spdt mid, type: node (`node[cute spdt mid]{}`). Class: `switches`.
	Cute spdt down, type: node (`node[cute spdt down]{}`). Class: `switches`.
	Cute spdt up with arrow, type: node (`node[cute spdt up arrow]{}`). Class: `switches`.
	Cute spdt mid with arrow, type: node (`node[cute spdt mid arrow]{}`). Class: `switches`.
	Cute spdt down with arrow, type: node (`node[cute spdt down arrow]{}`). Class: `switches`.

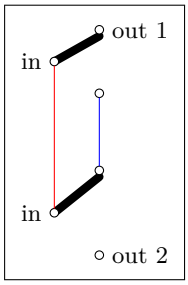
3.22.2.1 Cute switches anchors

The nodes-style switches have the following anchors:



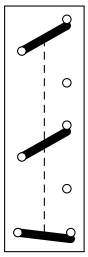
Please notice the position of the normal anchors at the border of the `ocirc` shape for the cute switches; they are thought to be compatible with an horizontal wire going out. Additionaly, you have the `cin`, `cout 1` y `cout 2` which are anchors on the center of the contacts.

For more complex situations, the contact nodes are available[14] using the syntax *name of the node*`-in`, …`-out 1` and …`-out 2`, with all their anchors.




```
\begin{circuitikz}
\draw (0,0) node[cute spdt up] (S1) {}
  (S1.in) node[left] {in}
  (S1.out 1) node[right] {out 1};
  \draw (0,-2) node[cute spdt up,
     /tikz/circuitikz/bipoles/cuteswitch/height=0.8] (S2) {}
  (S2.in) node[left] {in}
  (S2.out 2) node[right] {out 2};
  \draw [red] (S1-in.s) -- (S2-in.n);
  \draw [blue] (S1-out 2.s) -- (S2-out 1.n);
\end{circuitikz}
```

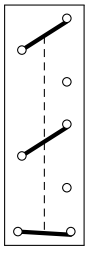
The `mid` anchor in the cute switches (both path- and node-style) can be used to combine switches to get more complex configurations:



```
\begin{circuitikz}
    \draw (0,1.4) node[cute spdt up](S1){};
    \draw (0,0) node[cute spdt up](S2){};
    \draw (0,-1) node[cuteclosedswitchshape, yscale=-1](S3){};
    \draw [densely dashed] (S1.mid)--(S2.mid)--(S3.mid);
\end{circuitikz}
```

3.22.2.2 Cute switches customization

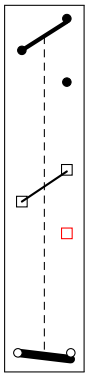
You can use the key `bipoles/cuteswitch/thickness` to decide the thickness of the switch lever. The units are the diameter of the `ocirc` connector, and the default is `1`.



```
\begin{circuitikz}
    \ctikzset{bipoles/cuteswitch/thickness=0.5}
    \draw (0,1.4) node[cute spdt up](S1){};
    \draw (0,0) node[cute spdt up](S2){};
    \draw (0,-1) node[cuteclosedswitchshape, yscale=-1](S3){};
    \draw [densely dashed] (S1.mid)--(S2.mid)--(S3.mid);
\end{circuitikz}
```

Finally, the switches are normally drawn using the `ocirc` shape, but you can change it, as in the following example, with the key `bipoles/cuteswitch/shape`. Be careful that the shape is used with its defaults (which can lead to strange results), and that the standard anchors will be correct only for `circ` and `ocirc` shapes, so you have to use the internal node syntax to connect it.

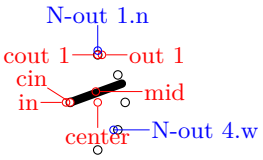
[14] Thanks to `@marmot` on tex.stackexchange.com.



```
\begin{circuitikz}
   \begin{scope}
      \ctikzset{bipoles/cuteswitch/thickness=0.5,
      bipoles/cuteswitch/shape=circ}
      \draw (0,2) node[cute spdt up](S1){};
      \ctikzset{bipoles/cuteswitch/thickness=0.25,
      bipoles/cuteswitch/shape=emptyshape}
      \draw (0,0) node[cute spdt up](S2){};
      \draw (S2.cin) node[draw, inner sep=2pt]{};
      \draw (S2.cout 1) node[draw, inner sep=2pt]{};
      \draw (S2.cout 2) node[draw=red, inner sep=2pt]{};
   \end{scope}
   \draw (0,-2) node[cuteclosedswitchshape, yscale=-1](S3){};
   \draw [densely dashed] (S1.mid)--(S2.mid)--(S3.mid);
\end{circuitikz}
```

3.22.3 Rotary switches

Rotary switches are a kind of generic multipole switches; they are implemented as a strongly customizable element (and a couple of styles to simplify its usage). The basic element is the following one, and it has the same basic anchors of the cute switches, included the access to internal nodes (shown in blue here).



Rotary switch, type: node (`node[rotaryswitch](N){}`). Class: `switches`.

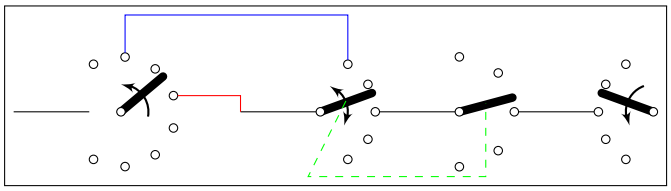
Notice that the name of the shape is `rotaryswitch`, no spaces. The default rotary switch component has 5 channels (this is set in the parameter `multipoles/rotary/channels`), spanning form $-60°$ to $60°$ (parameter `multipoles/rotary/angle`) and with the wiper at $20°$ (parameter `multipoles/rotary/wiper`).

Moreover, there are by default no arrows on the wiper; if needed, you can change this default setting the parameter `multipoles/rotary/arrow` which can assume the values `none`, `cw` (clockwise), `ccw` (counterclockwise) or `both`.

To simplify the usage of the component, a series of styles are defined: `rotary switch=<channels> in <angle> wiper <wiper angle>` (notice the space in the name of the style!). Using `rotary switch` without parameters will generate a default switch.

To add arrows, you can use the styles `rotary switch -` (no arrow, whatever the default), `rotary switch <-` (counterclockwise arrow), `rotary switch ->` (clockwise) and `rotary switch <->` (both).

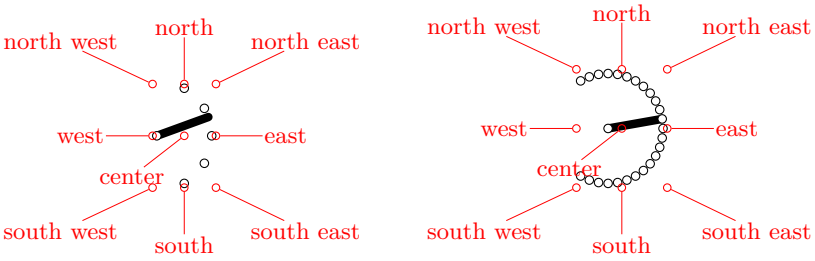
Notice that the defaults of the styles are the same as the default values of the parameters, but that if you change globally the defaults using the keys mentioned above, you only change the defaults for the "bare" component `rotaryswitch`, not for the styles.



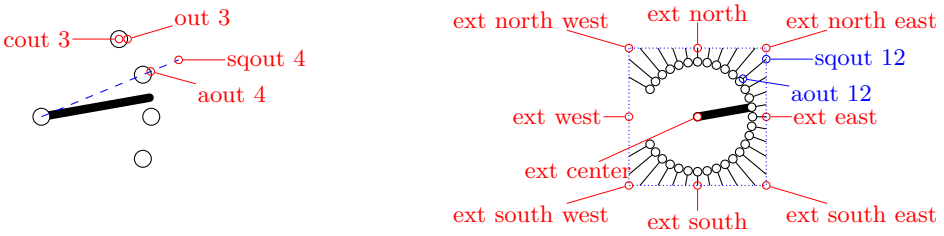
```
\begin{circuitikz}
\ctikzset{multipoles/rotary/arrow=both}
\draw (0,0) -- ++(1,0) node[rotary switch <-=8 in 120 wiper 40, anchor=in](A){};
\draw (3,0) -- ++(1,0) node[rotary switch, anchor=in](B){}; % default values
\draw[red] (A.out 4) -| (3,0);
\draw[blue] (A-out 2.n) -- ++(0,0.5) -| (B-out 1.n);
\draw (B.out 3) -- ++(1,0) node[rotary switch -=5 in 90 wiper 15, anchor=in](C){};
\draw (C.out 3) -- ++(1,0) node[rotary switch ->, xscale=-1, anchor=out 3](D){};
\draw[green, dashed] (B.mid) -- ++(-.5,-1) -| (C.mid);
\end{circuitikz}
```

3.22.3.1 Rotary switch anchors

Rotary switches anchors are basically the same as the cute switches, including access (with the `<node name>-<anchor name>` notation) to the internal connection nodes. The geographical anchors work as expected, marking the limits of the component.



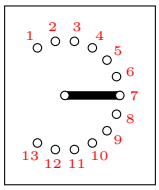
In addition to the anchors they have in common with the cute switches, the rotary switch has the so called "angled" anchors and the "external square anchors". *Angled anchors*, called `aout 1`, `aout 2` and so forth, are anchors placed on the output poles at the same angle as the imaginary lines coming from the input pole; *square anchors*, called `sqout 1`…, are located on an imaginary square surrounding the rotary switch on the same line.



The code for the diagram at the left, above, without the markings for the anchors, is:

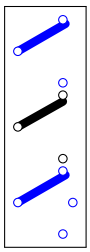
```
\begin{circuitikz}
    \draw (8,0) node[rotary switch -=31 in 150 wiper 10](D){};
    \foreach \i in {1,...,31} \draw (D.sqout \i) -- (D.aout \i);
    \draw[blue, densely dotted] (D.ext north west) rectangle (D.ext south east);
\end{circuitikz}
```

One possible application for the angled and the "on square" anchors is that you can use them to move radially from the output poles, for example for adding numbers:




```
\begin{circuitikz}
\draw (0,0) node[rotary switch=13 in 120 wiper 0](S){};
\foreach \i in {1,...,13} % requires "calc"
    \path ($(S.aout \i)!1ex!(S.sqout \i)$)
        node[font=\tiny\color{red}]{\i};
\end{circuitikz}
```

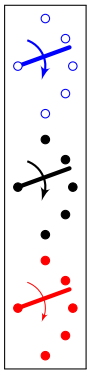
Finally, notice that the value of width for the rotary switches is taken from the one for the "cute switches" which in turn is taken from the width of traditional `spdt` switch, so that they match (notice that the "center" anchor is better centered in the rotary switch, so you have to explicitly align them).



```
\begin{circuitikz}
    \draw (0,0) node[color=blue, rotary switch=2 in 35 wiper 30,
        anchor=in](R){};
    \draw (0,-1) node[cute spdt up, anchor=in](C){};
    \draw (0,-2) node[color=blue, rotary switch=3 in 35 wiper 30,
        anchor=in](R){};
\end{circuitikz}
```

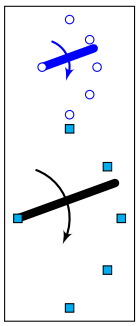
3.22.3.2 Rotary switch customization

Apart from the basic customization seen above (number of channels, etc.) you can change, as in the cute switches, the shape used by the connection points with the parameter `multipoles/rotary/shape`, and the thickness of the wiper with `multipoles/rotary/thickness`. The optional arrow has thickness equal to the standard bipole thickness `bipoles/thickness` (default 2).



```
\begin{circuitikz}
    \ctikzset{multipoles/rotary/thickness=0.5}
    \draw (0,1.6) node[rotary switch ->, color=blue](S1){};
    \ctikzset{multipoles/rotary/shape=circ}
    \draw (0,0) node[rotary switch ->](S2){};
    \ctikzset{bipoles/thickness=0.5}
    \draw (0,-1.6) node[rotary switch ->, color=red](S3){};
\end{circuitikz}
```

Finally, the size can be changed using the parameter `tripoles/spdt/width` (default 0.85).

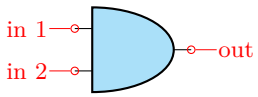


```
\begin{circuitikz}
    \draw (0,2) node[rotary switch ->, color=blue](S1){};
    \ctikzset{tripoles/spdt/width=1.6, fill=cyan,
        multipoles/rotary/shape=osquarepole}
    \draw (0,0) node[rotary switch ->](S2){};
\end{circuitikz}
```

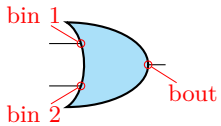
3.23 Logic gates

Logic gates, with two or more input, are supported. Albeit in principle these components are multipoles, the are considered tripoles here, for historical reasons (when they just had two inputs).

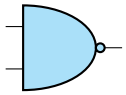
3.23.1 American Logic gates



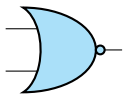
American AND port, type: node, fillable (`node[american and port]{}`). Class: `logic ports`.



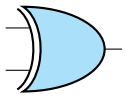
American OR port, type: node, fillable (`node[american or port]{}`). Class: `logic ports`.



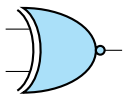
American NAND port, type: node, fillable (`node[american nand port]{}`). Class: `logic ports`.



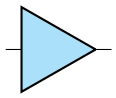
American NOR port, type: node, fillable (`node[american nor port]{}`). Class: `logic ports`.



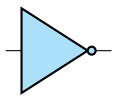
American XOR port, type: node, fillable (`node[american xor port]{}`). Class: `logic ports`.



American XNOR port, type: node, fillable (`node[american xnor port]{}`). Class: `logic ports`.



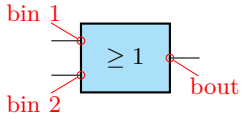
American BUFFER port, type: node, fillable (`node[american buffer port]{}`). Class: `logic ports`.



American NOT port, type: node, fillable (`node[american not port]{}`). Class: `logic ports`.

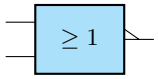
3.23.2 European Logic gates

European AND port, type: node, fillable (`node[european and port]{}`). Class: `logic ports`.

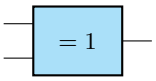


European OR port, type: node, fillable (`node[european or port]{}`). Class: `logic ports`.

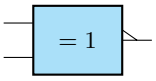
European NAND port, type: node, fillable (`node[european nand port]{}`). Class: `logic ports`.



European NOR port, type: node, fillable (`node[european nor port]{}`). Class: `logic ports`.



European XOR port, type: node, fillable (`node[european xor port]{}`). Class: `logic ports`.



European XNOR port, type: node, fillable (`node[european xnor port]{}`). Class: `logic ports`.

European BUFFER port, type: node, fillable (`node[european buffer port]{}`). Class: `logic ports`.

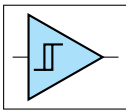
European NOT port, type: node, fillable (`node[european not port]{}`). Class: `logic ports`.

If (default behaviour) `americanports` option is active (or the style `[american ports]` is used), the shorthands `and port`, `or port`, `not port`, `nand port`, `not port`, `xor port`, and `xnor port` are equivalent to the american version of the respective logic port.

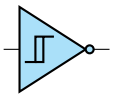
If otherwise `europeanports` option is active (or the style `[european ports]` is used), the shorthands `and port`, `or port`, `not port`, `nand port`, `not port`, `xor port`, and `xnor port` are equivalent to the european version of the respective logic port.

3.23.3 Special components

There is no "european" version of these symbols.



Non-Inverting SCHMITTTRIGGER, type: node, fillable (`node[schmitt]{}`). Class: `logic ports`.

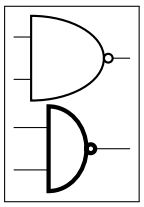


Inverting SCHMITTTRIGGER, type: node, fillable (`node[invschmitt]{}`). Class: `logic ports`.

3.23.4 Logic port customization

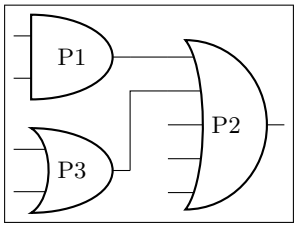
Logic port class is called `logic ports`, so you can scale them all with `logic ports/scale` (default `1.0`).

As for most components, you can change the width and height of the ports; the thickness is given by the parameter `tripoles/thickness` (default 2):



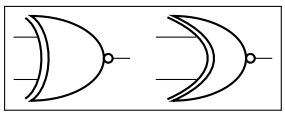
```
\tikz \draw (0,0) node[nand port] {}; \par
\ctikzset{tripoles/american nand port/input height=.2}
\ctikzset{tripoles/american nand port/port width=.4}
\ctikzset{tripoles/thickness=4}
\tikz \draw (0,0) node[nand port] {};
```

This is especially useful if you have ports with more than two inputs, which are instantiated with the parameter `number inputs` :




```
\begin{circuitikz}
\draw (0,3) node[american and port] (A) {P1};
\begin{scope}
    \ctikzset{tripoles/american or port/height=1.6}
    \draw (A.out) -- ++(0.5,0)
    node[american or port,
    number inputs=5,
    anchor=in 1] (B) {P2};
\end{scope}
\draw (0,1.5) node[american or port] (C) {P3};
\draw (C.out) |- (B.in 2);
\end{circuitikz}
```

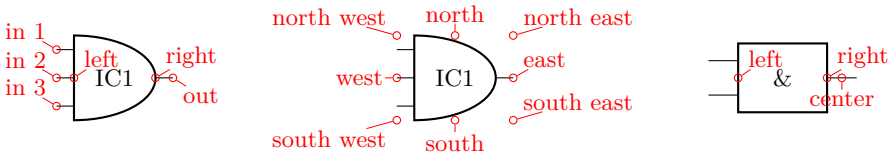
You can tweak the appearance of american "or" family (`or`, `nor`, `xor` and `xnor`) ports, too, with the parameters `inner` (how much the base circle go "into" the shape, default 0.3) and `angle` (the angle at which the base starts, default 70).



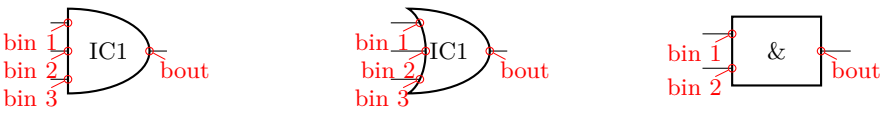
```
\tikz \draw (0,0) node[xnor port] {};
\ctikzset{tripoles/american xnor port/inner=.7}
\ctikzset{tripoles/american xnor port/angle=40}
\tikz \draw (0,0) node[xnor port] {};
```

3.23.5 Logic port anchors

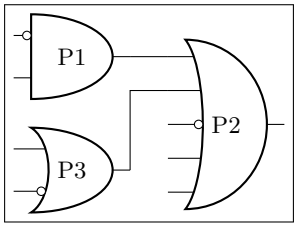
These are the anchors for logic ports:



You have also "border pin anchors":

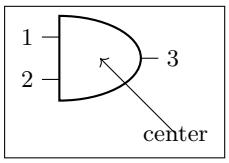


These anchors are especially useful if you want to negate inputs:

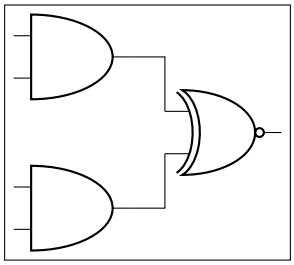



```
\begin{circuitikz}
\draw (0,3) node[american and port] (A) {P1};
\node at (A.bin 1) [ocirc, left]{} ;
\begin{scope}
    \ctikzset{tripoles/american or port/height=1.6}
    \draw (A.out) -- ++(0.5,0) node[american or port,
    number inputs=5, anchor=in 1] (B) {P2};
    \node at (B.bin 3) [ocirc, left]{} ;
\end{scope}
\draw (0,1.5) node[american or port] (C) {P3};
\node at (C.bin 2) [ocirc, left]{} ;
\draw (C.out) |- (B.in 2);
\end{circuitikz}
```

As you can see, the `center` anchor is (for historic reasons) not in the center at all. You can fix this with the command `\ctikzset{logic ports origin=center}`:

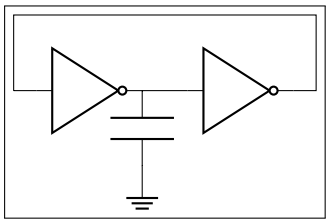



```
\begin{circuitikz}
\ctikzset{logic ports origin=center}
\draw (0,0) node[and port] (myand) {}
  (myand.in 1) node[anchor=east] {1}
  (myand.in 2) node[anchor=east] {2}
  (myand.out) node[anchor=west] {3};
\draw[<-] (myand.center) -- ++(1,-1)
   node{center};
\end{circuitikz}
```



```
\begin{circuitikz} \draw
  (0,2) node[and port] (myand1) {}
  (0,0) node[and port] (myand2) {}
  (2,1) node[xnor port] (myxnor) {}
  (myand1.out) -| (myxnor.in 1)
  (myand2.out) -| (myxnor.in 2)
;\end{circuitikz}
```

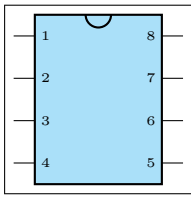
In the case of NOT, there are only `in` and `out` (although for compatibility reasons `in 1` is still defined and equal to `in`):



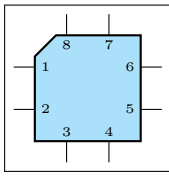
```
\begin{circuitikz} \draw
  (1,0) node[not port] (not1) {}
  (3,0) node[not port] (not2) {}
  (0,0) -- (not1.in)
  (not2.in) -- (not1.out)
  ++(0,-1) node[ground] {} to[C] (not1.out)
  (not2.out) -| (4,1) -| (0,0)
;\end{circuitikz}
```

3.24 Chips (integrated circuits)

`CircuiTikZ` supports two types of variable-pin chips: DIP (Dual-in-Line Package) and QFP (Quad-Flat Package).



Dual-in-Line Package chip, type: node, fillable (`node[dipchip]{}`). Class: `chips`.



Quad-Flat Package chip, type: node, fillable (`node[qfpchip]{}`). Class: `chips`.

3.24.1 DIP and QFP chips customization

You can scale chips with the key `chips/scale`. As ever, that will **not** scale text size of the labels, when they are printed.

You can customize the DIP chip with the key `multipoles/dipchip/width` (with a default of `1.2`) and the key `multipoles/dipchip/pin spacing` (default `0.4`) that are expressed in fraction of basic lengths (see section 3.1.2). The height of the chip will be equal to half the numbers of pins multiplied by the spacing, plus one spacing for the borders.

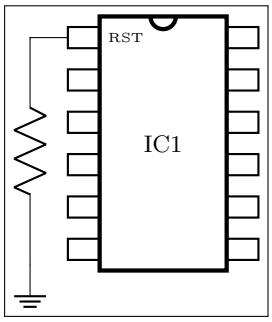
For the QFP chips, you can only chose the pin spacing with `multipoles/qfpchip/pin spacing` key.

The pins of the chip can be "hidden" (that is, just a spot in the border, optionally marked with a number) or "stick out" with a thin lead by setting `multipoles/external pins width` greater than 0 (default value is `0.2`, so you'll have leads as shown above). Moreover, you can transform the thin lead into a pad by setting the key `multipoles/external pad fraction` to something different form 0 (default is `0`); the value expresses the fraction of the pin spacing space that the pad will use on both sides of the pin.

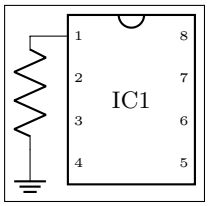
The number of pins is settable with the key `num pins`. **Please notice** that the number of pins **must** be *even* for `dipchip`s and *multiple of 4* for `qfpchip`s, otherwise havoc will ensue.

You can, if you want, avoid printing the numbers of the pin with `hide numbers` (default `show numbers`) if you prefer positioning them yourself (see the next section for the anchors you can use). The font used for the pins is adjustable with the key `multipoles/font` (default `\tiny`) For special use you can suppress the orientation mark with the key `no topmark` (default `topmark`).

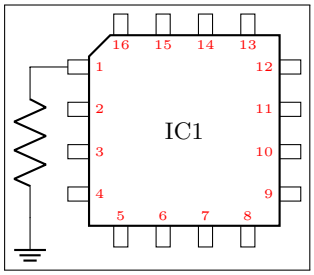
The line thickness of the main shape is controlled by `multipoles/thickness` (default 2) and the one of the external pins/pads with `multipoles/external pins thickness` (default 1).




```
\begin{circuitikz}
    \ctikzset{multipoles/thickness=4}
    \ctikzset{multipoles/external pins thickness=2}
    \draw (0,0) node[dipchip,
      num pins=12,
      hide numbers,
      external pins width=0.3,
      external pad fraction=4 ](C){IC1};
    \draw (C.pin 1) -- ++(-0.5,0) to[R]
      ++(0,-3) node[ground]{};
      \node [right, font=\tiny]
      at (C.bpin 1) {RST};
\end{circuitikz}
```




```
\begin{circuitikz}
    \draw (0,0) node[dipchip,
      num pins=8,
      external pins width=0.0](C){IC1};
    \draw (C.pin 1) -- ++(-0.5,0) to[R]
      ++(0,-1.5) node[ground]{};
\end{circuitikz}
```

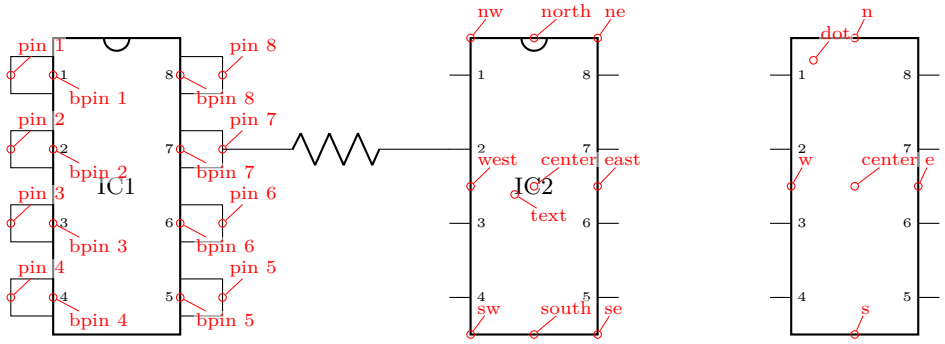



```
\begin{circuitikz}
    \ctikzset{multipoles/font={\color{red}\tiny}}
    \draw (0,0) node[qfpchip,
      num pins=16,
      external pad fraction=6](C){IC1};
    \draw (C.pin 1) -- ++(-0.5,0) to[R]
      ++(0,-2) node[ground]{};
\end{circuitikz}
```

3.24.2 Chips anchors

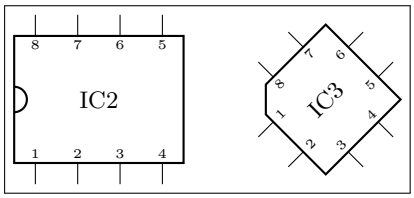
Chips have anchors on pins and global anchors for the main shape. The pin anchors to be used to connect wires to the chip are called `pin 1`, `pin 2` , ..., with just one space between `pin` and the number. Border pin anchors (`bpin 1`...) are always on the box border, and can be used to add numbers or whatever markings are needed. Obviously, in case of `multipoles/external pins width` equal to zero, border and normal pin anchors will coincide.

Additionally, you have geometrical anchors on the chip "box", see the following figure. The nodes are available with the full name (like `north`) and with the short abbreviations `n`, `nw`, `w`.... The `dot` anchor is useful to add a personalized marker if you use the `no topmark` key.



3.24.3 Chips rotation

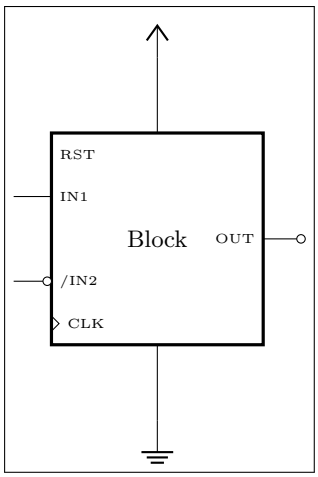
You can rotate chips, and normally the pin numbers are kept straight (option `straight numbers`, which is the default), but you can rotate them if you like with `rotated numbers`. Notice that the main label has to be (counter-) rotated manually in this case.




```
\begin{circuitikz}
    \draw (0,0) node[dipchip,
        rotate=90]{%
        \rotatebox{-90}{IC2}};
    \draw (3,0) node[qfpchip,
        rotated numbers,
        rotate=45]{IC3};
\end{circuitikz}
```

3.24.4 Chip special usage

You can use chips to have special, personalized blocks. Look at the following example, which is easily put into a macro.



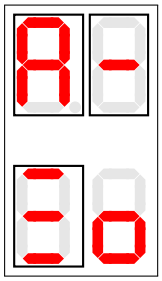

```
\begin{circuitikz}
    \ctikzset{multipoles/thickness=3}
    \ctikzset{multipoles/dipchip/width=2}
    \draw (0,0) node[dipchip,
        num pins=10, hide numbers, no topmark,
        external pins width=0](C){Block};
    \node [right, font=\tiny] at (C.bpin 1) {RST};
    \node [right, font=\tiny] at (C.bpin 2) {IN1};
    \node [right, font=\tiny] at (C.bpin 4) {/IN2};
    \node [left, font=\tiny] at (C.bpin 8) {OUT};
    \draw (C.bpin 2) -- ++(-0.5,0) coordinate(extpin);
    \node [ocirc, anchor=0](notin2) at (C.bpin 4) {};
    \draw (notin2.180) -- (C.bpin 4 -| extpin);
    \draw (C.bpin 8) to[short,-o] ++(0.5,0);
    \draw (C.bpin 5) ++(0,0.1) -- ++(0.1,-0.1)
        node[right, font=\tiny]{CLK} -- ++(-0.1,-0.1);
    \draw (C.n) -- ++(0,1) node[vcc]{};
    \draw (C.s) -- ++(0,-1) node[ground]{};
\end{circuitikz}
```

3.25 Seven segment displays

Seven segment display, type: node, fillable (`node[bare7seg]{}`). Class: `displays`.

The seven segment display lets you show values as if they were displayed in a classical seven segment display.[15]

The main "bare" component is the one shown above, but for simplicity a couple of style interfaces are defined:



```
\begin{circuitikz}
    \draw (0,0) node[seven segment val=A dot off box on]{};
    \draw (1,0) node[seven segment val=- dot none box on]{};
    \draw (0,-2) node[seven segment bits=1001001 dot empty box on
        ]{};
    \draw (1,-2) node[seven segment bits=0011101 dot none box off
        ]{};
\end{circuitikz}
```

There are two main configuration methods. The first one is `seven segment val`, which will take an hexadecimal number or value and display it: the possible values are `0,...,15`, plus `A, B, C, D, E, F` (or lowercase) and the symbol `-` (minus).

The other interface is `seven segment bits`, where you specify seven bits saying which segment must be on (please never specify a different number of bits, it will throw a very obscure error); you can see in the anchors the name of each segment.

[15] This component has been loosely inspired by the package `SevenSeg` by Germain Gondor, 2009, see TEXample.net.

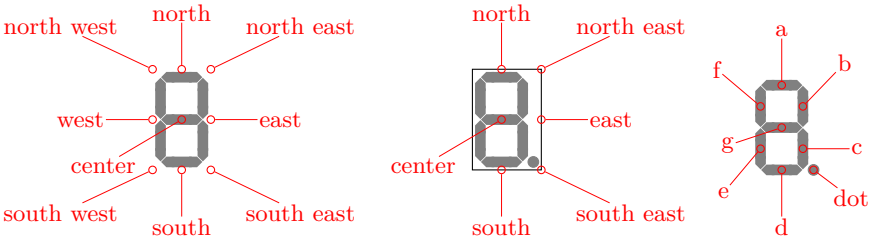
The option `dot` specifies if you want a decimal dot or not. The key `none` will remove the dot and the space it would take; `empty` will not show the dot at all but reserve the space, and `on` or `off` will show the dot in the corresponding state.

The option `box` (can be `on` or `off`) simply toggles the drawing of the external box. You can separate it from the display with the key `seven seg/box sep` (default `1pt`), and it will use the thickness specified in `multipoles/thickness` (The same as the chips).

You can use these option with the "bare" object `bare7seg` and the keys `seven seg/bits` (default `0000000`), `seven seg/dot` (default `none`) and `seven seg/box` (default `off`); there is no option equivalent to the `val` interface.

3.25.1 Seven segments anchors

These are the anchors for the seven segment displays; notice that when the `dot` parameter is not `none`, the cell is a bit wider at the right side.



3.25.2 Seven segments customization

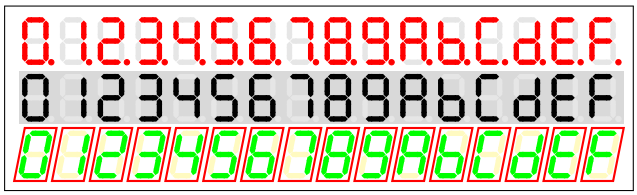
You can scale the seven segment display with the key `displays/scale`. This will scale the size of the digit, but not the absolute sizes shown below — if you want them to scale, yo have to do it manually.

You can change several parameters to adjust the displays:

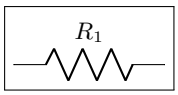
```
\ctikzset{seven seg/width/.initial=0.4}% relative to \pgf@circ@Rlen (scalable)
\ctikzset{seven seg/thickness/.initial=4pt}% segment thickness (not scaled)
\ctikzset{seven seg/segment sep/.initial=0.2pt}% gap between segments (not scaled)
\ctikzset{seven seg/box sep/.initial=1pt}% external box gap (not scaled)
\ctikzset{seven seg/color on/.initial=red}% color for segment "on"
\ctikzset{seven seg/color off/.initial=gray!20!white} % ...and "off"
```

A couple of examples are shown below.

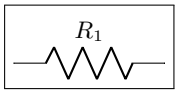
```
\begin{circuitikz}[scale=0.5]
\ctikzset{seven seg/width=0.2, seven seg/thickness=2pt}
\foreach \i in {0,...,15} \path (\i,0)
   node[seven segment val=\i dot on box off]{};
\ctikzset{seven seg/color on=black}
\foreach \i in {0,...,15} \path (\i,-1.5)
   node[seven segment val=\i dot off box off, fill=gray!30!white]{};
\ctikzset{seven seg/color on=green, seven seg/color off=yellow!30}
\foreach \i in {0,...,15} \path[color=red] (\i,-3)
   node[seven segment val=\i dot none box on, xslant=0.2]{};
\end{circuitikz}
```



4 Labels and similar annotations

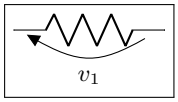



```
\begin{circuitikz}
  \draw (0,0) to[R, l=$R_1$] (2,0);
\end{circuitikz}
```

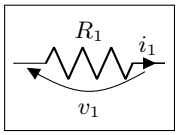



```
\begin{circuitikz}
  \draw (0,0) to[R=$R_1$] (2,0);
\end{circuitikz}
```

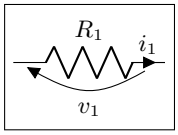
```
\begin{circuitikz}
  \draw (0,0) to[R, i=$i_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[R, v=$v_1$] (2,0);
\end{circuitikz}
```

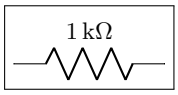



```
\begin{circuitikz}
  \draw (0,0) to[R=$R_1$, i=$i_1$, v=$v_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[R=$R_1$, i=$i_1$, v=$v_1$] (2,0);
\end{circuitikz}
```

Long names/styles for the bipoles can be used:

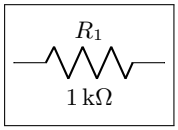



```
\begin{circuitikz}\draw
 (0,0) to[resistor=1<\kilo\ohm>] (2,0)
;\end{circuitikz}
```

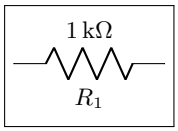
4.1 Labels and Annotations

Since Version 0.7, beside the original label (l) option, there is a new option to place a second label, called annotation (a) at each bipole. Up to now this is a beta-test and there can be problems. For example, up to now this option is not compatible with the concurrent use of voltage labels.

The position of (a) and (l) labels can be adjusted with _ and ^, respectively.

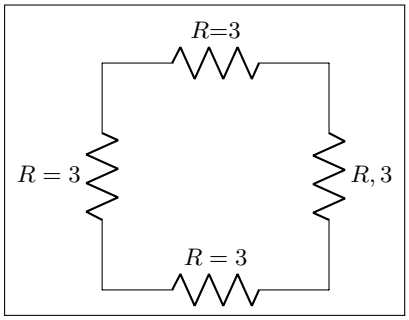



```
\begin{circuitikz}
  \draw (0,0) to[R, l=$R_1$,a=1<\kilo\ohm>] (2,0);
\end{circuitikz}
```



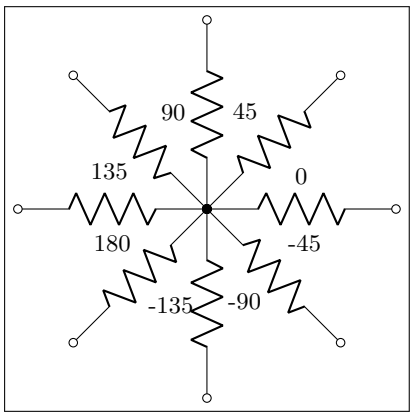

```
\begin{circuitikz}
  \draw (0,0) to[R, l_=$R_1$,a^=1<\kilo\ohm>] (2,0);
\end{circuitikz}
```

Caveat: notice that the way in which `circuitikz` processes the options, there will be problems if the label (or annotation, voltage, or current) contains one of the characters = (equal) or , (comma), giving unexpected errors and wrong output. These two characters must be protected to the option parser using an `\mbox` command, or redefining the characters with a TEX `\def`:

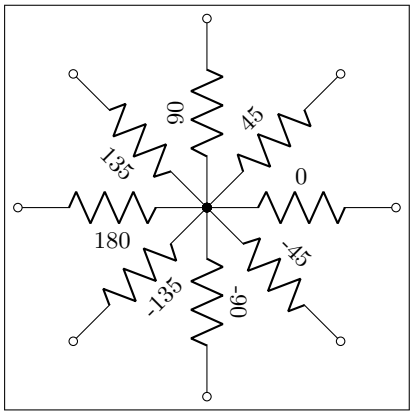



```
\def\eq{=}
\begin{circuitikz}
    % the following will fail:
    % \draw (0,0) to[R, l={$R=3}] (3,0);
    \draw (0,0) to[R, l=\mbox{$R=3$}] (3,0);
    \draw (0,0) to[R, l=$R\eq3$] (0,3);
    \draw (3,3) to[R, l=\mbox{$R,3$}] (3,0);
    % this works, but it has wrong spacing
    \draw (0,3) to[R, l=$R{=}3$] (3,3);
\end{circuitikz}
```

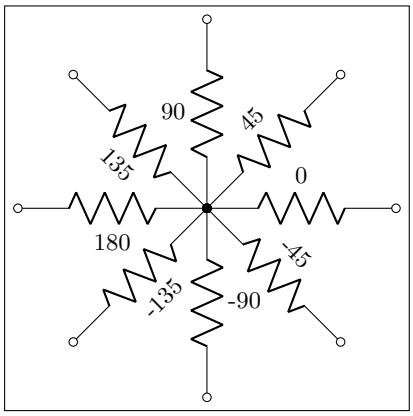
The default orientation of labels is controlled by the options `smartlabels`, `rotatelabels` and `straightlabels` (or the corresponding `label/align` keys). Here are examples to see the differences:




```
\begin{circuitikz}
\ctikzset{label/align = straight}
\def\DIR{0,45,90,135,180,-90,-45,-135}
\foreach \i in \DIR {
  \draw (0,0) to[R=\i, *-o] (\i:2.5);
}
\end{circuitikz}
```

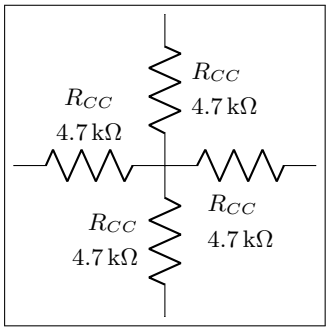



```
\begin{circuitikz}
\ctikzset{label/align = rotate}
\def\DIR{0,45,90,135,180,-90,-45,-135}
\foreach \i in \DIR {
  \draw (0,0) to[R=\i, *-o] (\i:2.5);
}
\end{circuitikz}
```

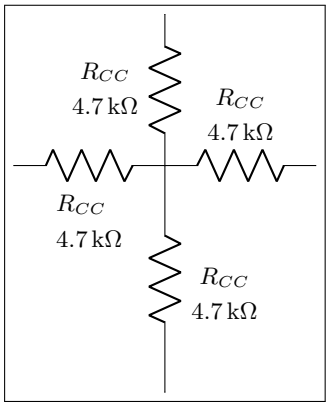



```
\begin{circuitikz}
\ctikzset{label/align = smart}
\def\DIR{0,45,90,135,180,-90,-45,-135}
\foreach \i in \DIR {
  \draw (0,0) to[R=\i, *-o] (\i:2.5);
}
\end{circuitikz}
```

You also can use stacked (two lines) labels. The example should be self-explanatory: the two lines are specified as `l2=`*line1* `and` *line2*. You can use the keys `l2 halign` to control horizontal position (`l`eft, `c`enter, `r`ight) and `l2 valign` to control the vertical one (`b`ottom, `c`center, `t`op).




```
\begin{circuitikz}[ american, ]
   %
   % default is l2 halign=l, l2 valign=c
   %
   \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\
       ohm},   , l2 valign=t] (2,0);
   \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\
       ohm},   ,          ] (0,2);
   \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\
       ohm}, l2 halign=c, l2 valign=b] (-2,0);
   \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\
       ohm}, l2 halign=r, l2 valign=c] (0, -2);
\end{circuitikz}
```




```
\begin{circuitikz}[ american, ]
   \draw (0,0) to[R, l2^=$R_{CC}$ and \SI{4.7}{k\
       ohm}, l2 halign=c, l2 valign=b] (2,0);
   \draw (0,0) to[R, l2^=$R_{CC}$ and \SI{4.7}{k\
       ohm}, l2 halign=c, ] (0,2);
   \draw (0,0) to[R, l2^=$R_{CC}$ and \SI{4.7}{k\
       ohm},   , l2 valign=t] (-2,0);
   \draw (0,0) to[R, l2^=$R_{CC}$ and \SI{4.7}{k\
       ohm}, l2 halign=c, l2 valign=t](0, -3);
\end{circuitikz}
```

4.2 Currents and voltages

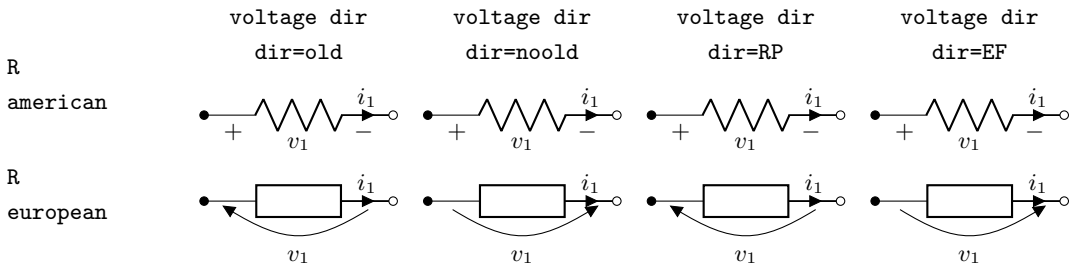
The default direction/sign for currents and voltages in the components is, unfortunately, not standard, and can change across country and sometime across different authors. This unfortunate situation created a bit of confusion in `circuitikz` across the versions, with several incompatible changes starting from version 0.5. From version 0.9.0 onward, the maintainers agreed a new policy for the directions of bipoles' voltages and currents, depending on 4 different possible options:

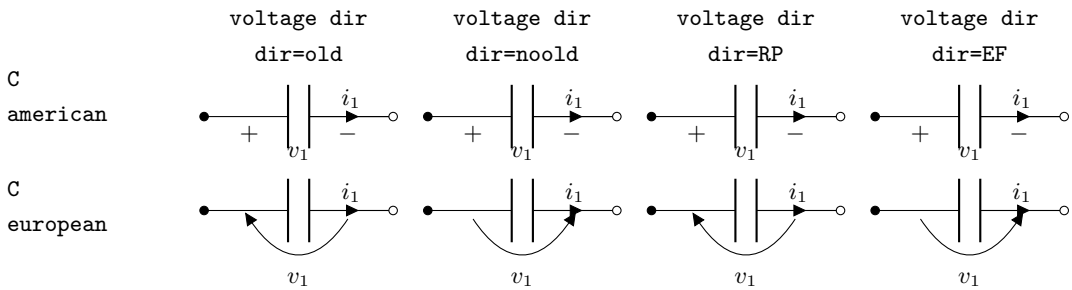
- `oldvoltagedirection`, or the key style `voltage dir=old`: Use old way of voltage direction having a difference between european and american direction, with wrong default labelling for batteries (it was the default before version 0.5);
- `nooldvoltagedirection`, or the key style `voltage dir=noold`: The standard from version 0.5 onward, utilize the (German?) standard of voltage arrows in the direction of electric fields (without fixing batteries);
- `RPvoltages` (meaning Rising Potential voltages), or the key style `voltage dir=RP`: the arrow is in direction of rising potential, like in `oldvoltagedirection`, but batteries and current sources are fixed so that they follow the passive/active standard: the default direction of `v` and `i` are chosen so that, when both values are positive:
  - in passive component, the element is *dissipating power*;
  - in active components (generators), the element is *generating power*.
- `EFvoltages` (meaning Electric Field voltages), or the key style `voltage dir=EF`: the arrow is in direction of the electric field, like in `nooldvoltagedirection`, but batteries are fixed;

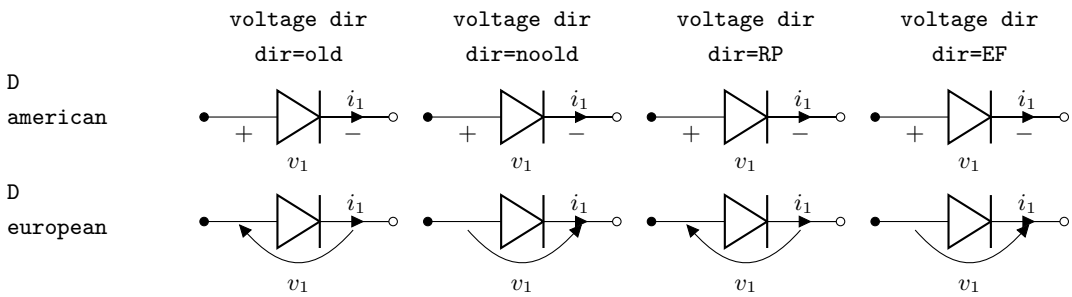
Notice that the four styles are designed to be used at the environment level: that is, you should use them at the start of your environment as in `\begin{circuitikz}[voltage dir=old] ...` and not as a key for single components, in which case the behaviour is not guaranteed.

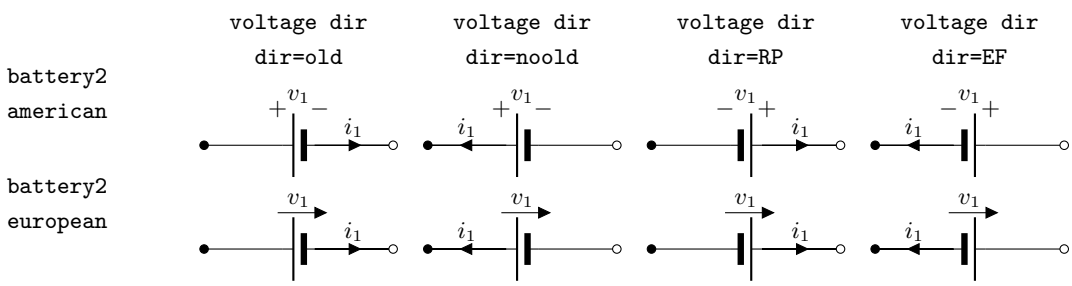
The standard direction of currents, flows and voltages are changed by these options; notice that the default drops in case of passive and active elements is normally different. Take care that in the case of `noold` and `EFvoltages` also the currents can switch directions. It is much easier to understand the several behaviors by looking at the following examples, that have been generated by the code:

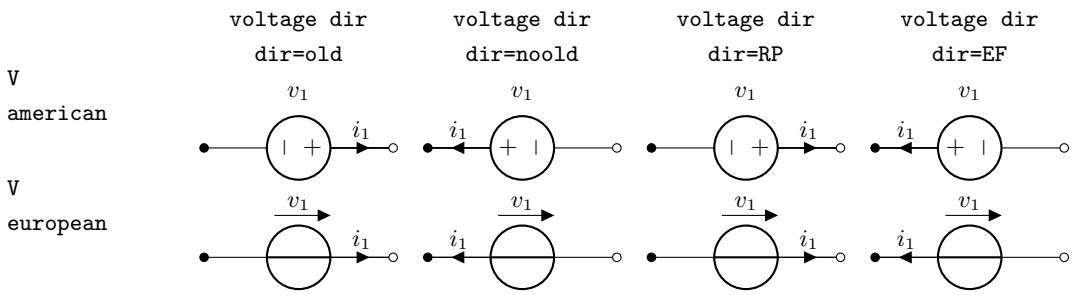
```
\foreach\element in {R, C, D, battery2, V, I, sV, cV, cI}{%
   \noindent\ttfamily
   \begin{tabular}{p{2cm}}
      \element \\ american \\[15pt]
      \element \\ european \\
   \end{tabular}
   \foreach\mode in {old, noold, RP, EF} {
      \begin{tabular}{@{}l@{}}
         \multicolumn{1}{c}{voltage dir} \\
         \multicolumn{1}{c}{dir=\mode} \\[4pt]
         \begin{tikzpicture}[
            american, voltage dir=\mode,
            ]
            \draw (0,0) to[\element, *-o, v=$v_1$, i=$i_1$, ] (2.5,0);
         \end{tikzpicture}\\
         \begin{tikzpicture}[
            european, voltage dir=\mode,
            ]
            \draw (0,0) to[\element, *-o, v=$v_1$, i=$i_1$, ] (2.5,0);
         \end{tikzpicture}
      \end{tabular}
      \medskip
   }
   \par
}
```

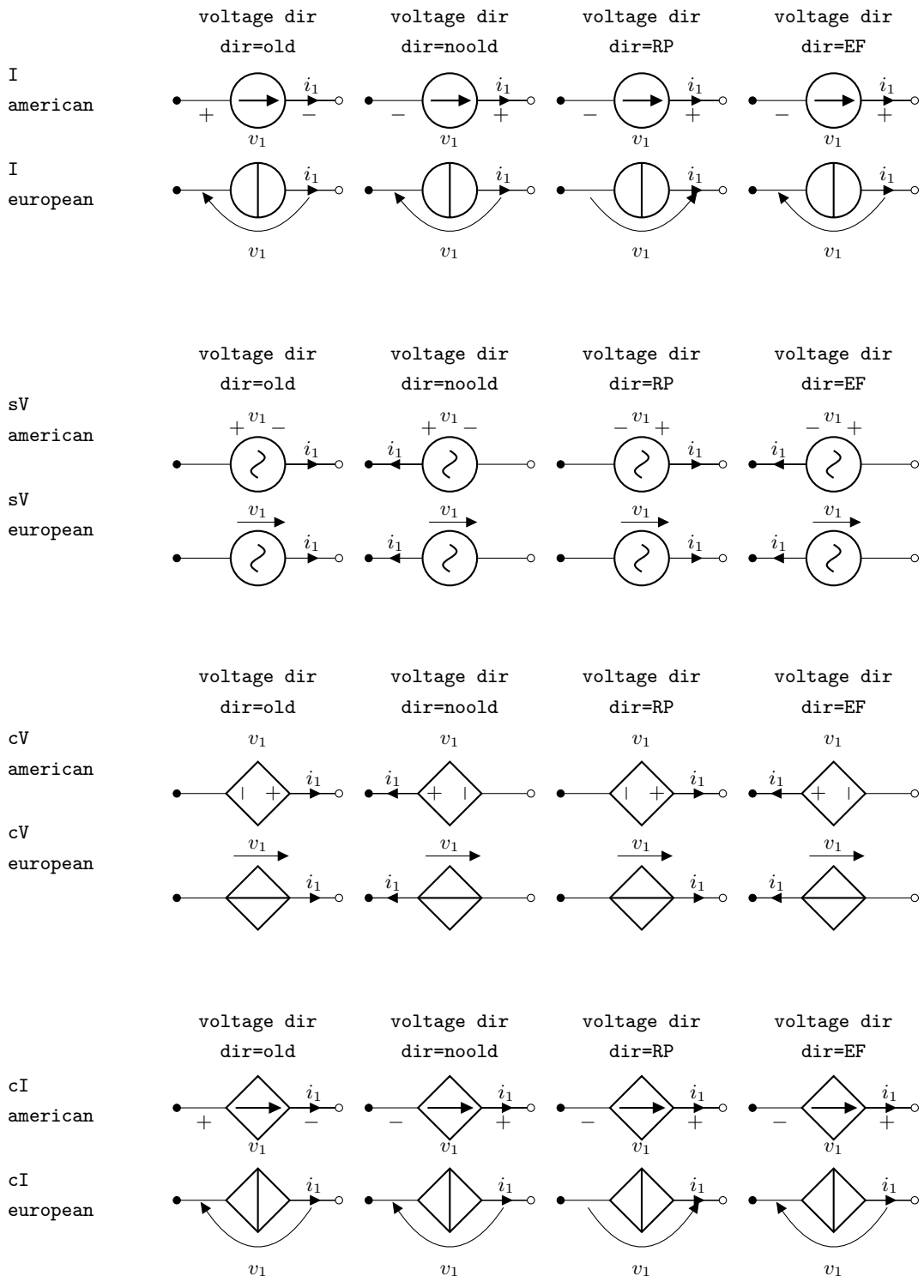

voltage dir
dir=old
voltage dir
dir=noold
voltage dir
dir=RP
voltage dir
dir=EF
R
american
R
european
i_1
v_1
+
−


voltage dir
dir=old
voltage dir
dir=noold
voltage dir
dir=RP
voltage dir
dir=EF
C
american
C
european
i_1
v_1
+
−


voltage dir
dir=old
voltage dir
dir=noold
voltage dir
dir=RP
voltage dir
dir=EF
D
american
D
european
i_1
v_1
+
−


voltage dir
dir=old
voltage dir
dir=noold
voltage dir
dir=RP
voltage dir
dir=EF
battery2
american
battery2
european
v_1
i_1
+
−


voltage dir
dir=old
voltage dir
dir=noold
voltage dir
dir=RP
voltage dir
dir=EF
V
american
V
european
v_1
i_1



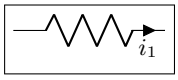
Obviously, you normally use just one between current and flows, but anyway you can change direction of the voltages, currents and flows using the complete keys `i_>`, `i^<`, `i>_`, `i>^`, as shown in the following examples.

This manual has been typeset with the option `RPvoltages`.

4.3 Currents

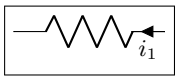
Inline (along the wire) currents are selected with `i_>`, `i^<`, `i>_`, `i>^`, and various simplification; the default position and direction is obtained with the key `i=`...

```
\begin{circuitikz}
  \draw (0,0) to[R, i^>=$i_1$] (2,0);
\end{circuitikz}
```



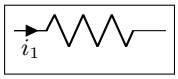

```
\begin{circuitikz}
  \draw (0,0) to[R, i_>=$i_1$] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, i^<=$i_1$] (2,0);
\end{circuitikz}
```

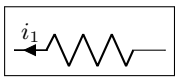



```
\begin{circuitikz}
  \draw (0,0) to[R, i_<=$i_1$] (2,0);
\end{circuitikz}
```

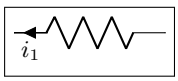
```
\begin{circuitikz}
  \draw (0,0) to[R, i>^=$i_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[R, i>_=$i_1$] (2,0);
\end{circuitikz}
```



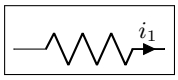

```
\begin{circuitikz}
  \draw (0,0) to[R, i<^=$i_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[R, i<_=$i_1$] (2,0);
\end{circuitikz}
```

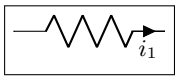
Also:

```
\begin{circuitikz}
  \draw (0,0) to[R, i<=$i_1$] (2,0);
\end{circuitikz}
```

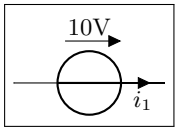



```
\begin{circuitikz}
  \draw (0,0) to[R, i>=$i_1$] (2,0);
\end{circuitikz}
```

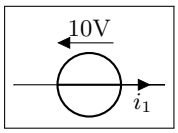
```
\begin{circuitikz}
  \draw (0,0) to[R, i^=$i_1$] (2,0);
\end{circuitikz}
```



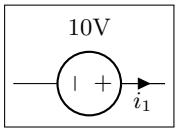

```
\begin{circuitikz}
  \draw (0,0) to[R, i_=$i_1$] (2,0);
\end{circuitikz}
```



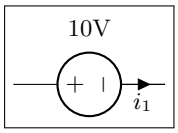

```
\begin{circuitikz}
  \draw (0,0) to[V=10V, i_=$i_1$] (2,0);
\end{circuitikz}
```



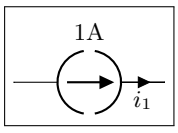

```
\begin{circuitikz}
  \draw (0,0) to[V<=10V, i_=$i_1$] (2,0);
\end{circuitikz}
```



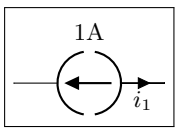

```
\begin{circuitikz}[american]
  \draw (0,0) to[V=10V, i_=$i_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}[american]
  \draw (0,0) to[V=10V,invert, i_=$i_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}[american]
  \draw (0,0) to[dcisource=1A, i_=$i_1$] (2,0);
\end{circuitikz}
```

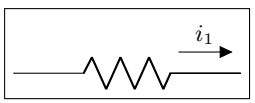



```
\begin{circuitikz}[american]
  \draw (0,0) to[dcisource=1A,invert, i_=$i_1$] (2,0);
\end{circuitikz}
```

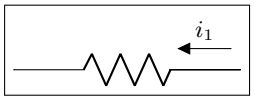
4.4 Flows

As an alternative for the current arrows, you can also use the following flows. They can also be used to indicate thermal or power flows. The syntax is pretty the same as for currents.

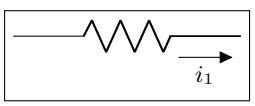
This is a new beta feature since version 0.8.3; therefore, please provide bug reports or hints to optimize this feature regarding placement and appearance! This means that the appearance may change in the future!



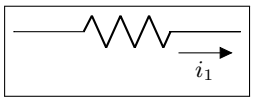

```
\begin{circuitikz}
  \draw (0,0) to[R, f=$i_1$] (3,0);
\end{circuitikz}
```



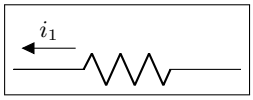

```
\begin{circuitikz}
  \draw (0,0) to[R, f<=$i_1$] (3,0);
\end{circuitikz}
```



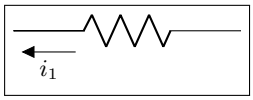

```
\begin{circuitikz}
  \draw (0,0) to[R, f_=$i_1$] (3,0);
\end{circuitikz}
```



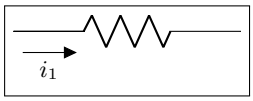

```
\begin{circuitikz}
  \draw (0,0) to[R, f_>=$i_1$] (3,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[R, f<^=$i_1$] (3,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[R, f<_=$i_1$] (3,0);
\end{circuitikz}
```



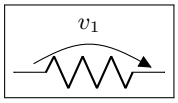

```
\begin{circuitikz}
  \draw (0,0) to[R, f>_=$i_1$] (3,0);
\end{circuitikz}
```

4.5 Voltages

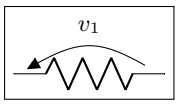
See introduction note at Currents (chapter 4.2, page 96)!

4.5.1 European style

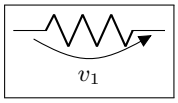
The default, with arrows. Use option `europeanvoltage` or style `[european voltages]`.



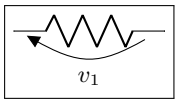

```
\begin{circuitikz}[european voltages]
  \draw (0,0) to[R, v^>=$v_1$] (2,0);
\end{circuitikz}
```



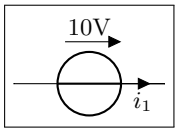

```
\begin{circuitikz}[european voltages]
  \draw (0,0) to[R, v^<=$v_1$] (2,0);
\end{circuitikz}
```



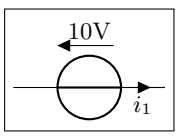

```
\begin{circuitikz}[european voltages]
  \draw (0,0) to[R, v_>=$v_1$] (2,0);
\end{circuitikz}
```



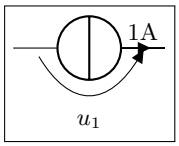

```
\begin{circuitikz}[european voltages]
  \draw (0,0) to[R, v_<=$v_1$] (2,0);
\end{circuitikz}
```



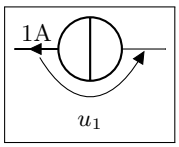

```
\begin{circuitikz}
  \draw (0,0) to[V=10V, i_=$i_1$] (2,0);
\end{circuitikz}
```



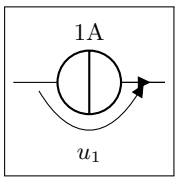

```
\begin{circuitikz}
  \draw (0,0) to[V<=10V, i_=$i_1$] (2,0);
\end{circuitikz}
```



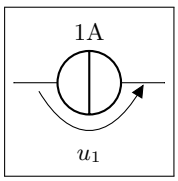

```
\begin{circuitikz}
  \draw (0,0) to[I=1A, v_=$u_1$] (2,0);
\end{circuitikz}
```



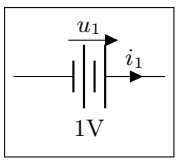

```
\begin{circuitikz}
  \draw (0,0) to[I<=1A, v_=$u_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[I=$~$,l=1A, v_=$u_1$] (2,0);
\end{circuitikz}
```



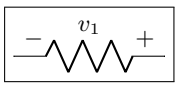

```
\begin{circuitikz}
  \draw (0,0) to[I,l=1A, v_=$u_1$] (2,0);
\end{circuitikz}
```



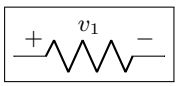

```
\begin{circuitikz}
  \draw (0,0) to[battery,l_=1V, v=$u_1$, i=$i_1$] (2,0);
\end{circuitikz}
```

4.5.2 American style

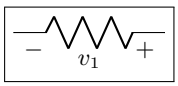
Use option `americanvoltage` or set `[american voltages]`.



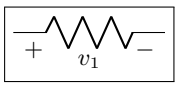

```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[R, v^>=$v_1$] (2,0);
\end{circuitikz}
```



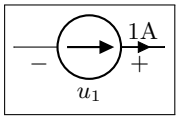

```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[R, v^<=$v_1$] (2,0);
\end{circuitikz}
```



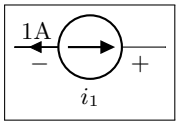

```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[R, v_>=$v_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[R, v_<=$v_1$] (2,0);
\end{circuitikz}
```



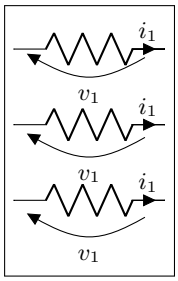

```
\begin{circuitikz}[american]
  \draw (0,0) to[I=1A, v_=$u_1$] (2,0);
\end{circuitikz}
```



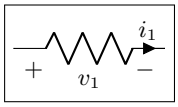

```
\begin{circuitikz}[american]
  \draw (0,0) to[I<=1A, v_=$i_1$] (2,0);
\end{circuitikz}
```

4.5.3 Voltage position

It is possible to move away the arrows and the plus or minus signs with the key `voltages shift` (default value is `0`, which gives the standard position):

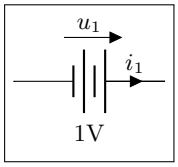



```
\begin{circuitikz}[]
  \draw (0,0) to[R, v=$v_1$, i=$i_1$] (2,0);
  \draw (0,-1) to[R, v=$v_1$, i=$i_1$,
    voltage shift=0.5] (2,-1);
  \draw (0,-2) to[R, v=$v_1$, i=$i_1$,
    voltage shift=1.0, ] (2,-2);
\end{circuitikz}
```

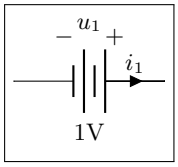



```
\begin{circuitikz}[american voltages, voltage shift=0.5]
  \draw (0,0) to[R, v=$v_1$, i=$i_1$] (2,0);
\end{circuitikz}
```

Notes that `american voltage` also affects batteries.



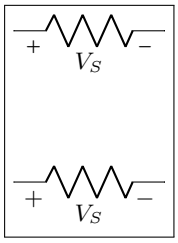

```
 \begin{circuitikz}[voltage shift=0.5]
  \draw (0,0) to[battery,l_=1V, v=$u_1$, i=$i_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}[american voltages, voltage shift=0.5]
  \draw (0,0) to[battery,l_=1V, v=$u_1$, i=$i_1$] (2,0);
\end{circuitikz}
```

4.5.4 American voltages customization

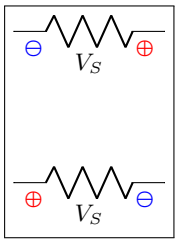
Since 0.9.0, you can change the font[16] used by the `american voltages` style, by setting to something different from nothing the key `voltage/american font` (default: nothing, using the current font) style:




```
\begin{circuitikz}[american]
   \begin{scope}
      \ctikzset{voltage/american font=\tiny\boldmath}
      \draw (0,0) to[R,v=$V_S$] ++(2,0);
   \end{scope}
   \draw (0,-2) to[R,v=$V_S$] ++(2,0);
\end{circuitikz}
```

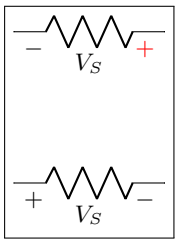
Also, if you want to change the symbols (sometime just the + sign is drawn, for example, or for highlighting something), using the keys `voltage/american plus` and `voltage/american minus` (default `$+$` and `$-$`).

[16] There where a bug before, noticed by the user `dzereb` on tex.stackexchange.com which made the symbols using different fonts in a basically random way. In the same page, user `campa` found the problem. Thanks!



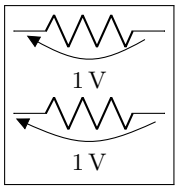

```
\begin{circuitikz}[american]
    \ctikzset{voltage/american font=\scriptsize\boldmath}
    \ctikzset{voltage/american plus=\textcolor{red}{$\oplus$}}
    \ctikzset{voltage/american minus=\textcolor{blue}{$\ominus$}}
    \draw (0,0) to[R,v_>=$V_S$] ++(2,0);
    \draw (0,-2) to[R,v_<=$V_S$] ++(2,0);
\end{circuitikz}
```

This could be especially useful if you define a style, to use like this:

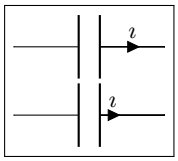



```
\tikzset{red plus/.style={
    circuitikz/voltage/american plus=\textcolor{red}{$+$},
}}
\begin{circuitikz}[american]
    \draw (0,0) to[R,v_>=$V_S$, red plus] ++(2,0);
    \draw (0,-2) to[R,v_<=$V_S$] ++(2,0);
\end{circuitikz}
```

4.5.5 Global properties of voltages and currents

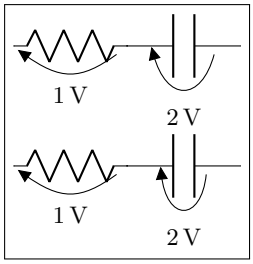



```
\tikz \draw (0,0) to[R, v=1<\volt>] (2,0); \par
\ctikzset{voltage/distance from node=.1}
\tikz \draw (0,0) to[R, v=1<\volt>] (2,0);
```




```
\tikz \draw (0,0) to[C, i=$\imath$] (2,0); \par
\ctikzset{current/distance = .2}
\tikz \draw (0,0) to[C, i=$\imath$] (2,0);
```

However, you can override the properties `voltage/distance from node` (how distant from the initial and final points of the path the arrow starts and ends) and `voltage/bump b` (how high the bump of the arrow is — how curved it is)[17] on a per-component basis, in order to fine-tune the voltages:



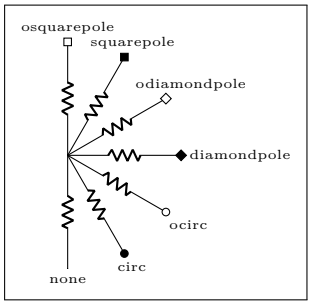

```
\tikz \draw (0,0) to[R, v=1<\volt>] (1.5,0)
      to[C, v=2<\volt>] (3,0); \par
\ctikzset{bipoles/capacitor/voltage/distance from node/.
    initial=.7}
\tikz \draw (0,0) to[R, v=1<\volt>] (1.5,0)
      to[C, v=2<\volt>] (3,0); \par
```

Note the `.initial`; you have to create such key the first time you use it.

[17] Prior to 0.9.4 you had also `voltage/european label distance` (how distant from the bipole the voltage label will be) but this is deprecated, and the european-style label is printed near the bump)

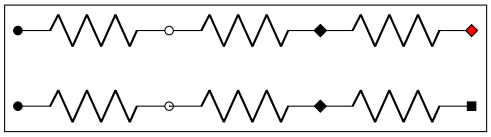
4.6 Nodes (also called poles)

You can add nodes to the bipoles, positioned at the coordinates surrounding the component. The general style to use is `bipole nodes={start}{stop}`, where `start` and `stop` are the nodes — to be chosen between `none`, `circ`, `ocirc`, `squarepole`, `osquarepole`, `diamondpole`, `odiamondpole` and `rectfill`[18] (see section 3.14).




```
\begin{circuitikz}
    \ctikzset{bipoles/length=.5cm, nodes width=0.1}%small
        components, big nodes
    \foreach \a/\p [evaluate=\a as \b using (\a+180)] in
    {-90/none, -60/circ, -30/ocirc, 0/diamondpole, 30/
        odiamondpole, 60/squarepole, 90/osquarepole}
        \draw (0,0) to[R, bipole nodes={none}{\p}] ++(\a:1.5)
            node[font=\tiny, anchor=\b]{\p};
\end{circuitikz}
```

These bipole nodes are added after the path is drawn, as every node in Ti*k*Z — this is the reason why they are always filled (with the main color the normal nodes, with white the open ones), in order to "hide" the wire below. You can override the fill color if you want; but notice that if you draw things in two different paths, you will have "strange" results; notice that in the second line of resistors the second wire is starting from the center of the white `ocirc` of the previous path.



```
\begin{circuitikz}
    \draw (0,0) to[R, *-o] ++(2,0) to[R, -d] ++(2,0)
        to[R, bipole nodes={diamondpole}{odiamondpole, fill=red}] ++(2,0);
    \draw (0,-1) to[R, *-o] ++(2,0) ;
    \draw (2,-1) to[R, -d] ++(2,0) to[R, bipole nodes={none}{squarepole}] ++(2,0);
\end{circuitikz}
```

You can define shortcuts for the `bipole nodes` you use most; for example if you want a shortcut for a bipole with open square node in red in the right side you can:

```
\begin{circuitikz}
    \ctikzset{-s/.style = {bipole nodes={none}{osquarepole, fill=red}}}
    \draw (0,0) to[R, -s] ++(2,0);
\end{circuitikz}
```

There are several predefined shorthand as the above; in the following pages you can see all of them.

```
\begin{circuitikz}
    \draw (0,0) to[R, o-o] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
    \draw (0,0) to[R, -o] (2,0);
\end{circuitikz}
```

[18] You can use other shapes too, but at your own risk…Moreover, notice that `none` is not really a node, just a special word used to say "do not put any node here".

```
\begin{circuitikz}
  \draw (0,0) to[R, o-] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, *-*] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, -*] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, *-] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, d-d] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, -d] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, d-] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, o-*] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, *-o] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, o-d] (2,0);
\end{circuitikz}
```

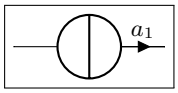
```
\begin{circuitikz}
  \draw (0,0) to[R, d-o] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, *-d] (2,0);
\end{circuitikz}
```

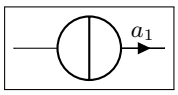
```
\begin{circuitikz}
  \draw (0,0) to[R, d-*] (2,0);
\end{circuitikz}
```

4.7 Special components

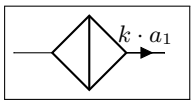
For some components label, current and voltage behave as one would expect:



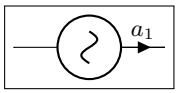

```
\begin{circuitikz}
  \draw (0,0) to[I=$a_1$] (2,0);
\end{circuitikz}
```



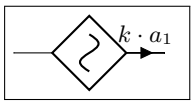

```
\begin{circuitikz}
  \draw (0,0) to[I, i=$a_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[cI=$k\cdot a_1$] (2,0);
\end{circuitikz}
```

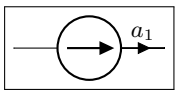



```
\begin{circuitikz}
  \draw (0,0) to[sI=$a_1$] (2,0);
\end{circuitikz}
```

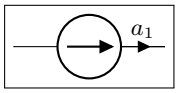



```
\begin{circuitikz}
  \draw (0,0) to[csI=$k\cdot a_1$] (2,0);
\end{circuitikz}
```

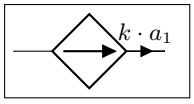
The following results from using the option `americancurrent` or using the style `[american currents]`.



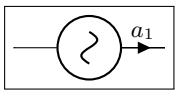

```
\begin{circuitikz}[american currents]
  \draw (0,0) to[I=$a_1$] (2,0);
\end{circuitikz}
```



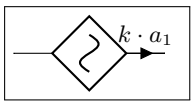

```
\begin{circuitikz}[american currents]
  \draw (0,0) to[I, i=$a_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}[american currents]
  \draw (0,0) to[cI=$k\cdot a_1$] (2,0);
\end{circuitikz}
```

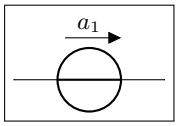



```
\begin{circuitikz}[american currents]
  \draw (0,0) to[sI=$a_1$] (2,0);
\end{circuitikz}
```

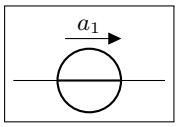



```
\begin{circuitikz}[american currents]
  \draw (0,0) to[csI=$k\cdot a_1$] (2,0);
\end{circuitikz}
```

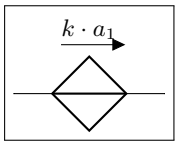
The same holds for voltage sources:



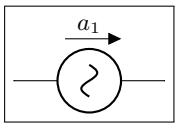

```
\begin{circuitikz}
  \draw (0,0) to[V=$a_1$] (2,0);
\end{circuitikz}
```



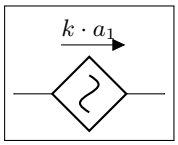

```
\begin{circuitikz}
  \draw (0,0) to[V, v=$a_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[cV=$k\cdot a_1$] (2,0);
\end{circuitikz}
```

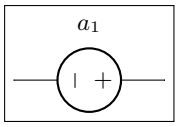



```
\begin{circuitikz}
  \draw (0,0) to[sV=$a_1$] (2,0);
\end{circuitikz}
```

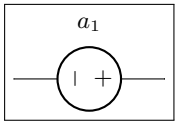



```
\begin{circuitikz}
  \draw (0,0) to[csV=$k\cdot a_1$] (2,0);
\end{circuitikz}
```

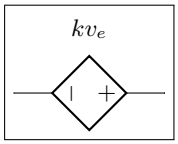
The following results from using the option `americanvoltage` or the style `[american voltages]`.



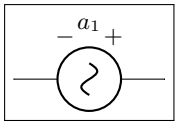

```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[V=$a_1$] (2,0);
\end{circuitikz}
```



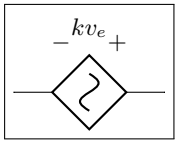

```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[V, v=$a_1$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[cV=$k v_e$] (2,0);
\end{circuitikz}
```



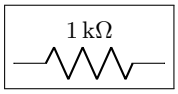

```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[sV=$a_1$] (2,0);
\end{circuitikz}
```



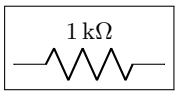

```
\begin{circuitikz}[american voltages]
  \draw (0,0) to[csV=$k v_e$] (2,0);
\end{circuitikz}
```

4.8 Integration with `siunitx`

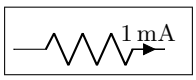
If the option `siunitx` is active (and *not* in ConTEXt), then the following are equivalent:




```
\begin{circuitikz}
  \draw (0,0) to[R, l=1<\kilo\ohm>] (2,0);
\end{circuitikz}
```

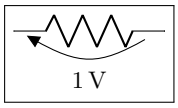



```
\begin{circuitikz}
  \draw (0,0) to[R, l=$\SI{1}{\kilo\ohm}$] (2,0);
\end{circuitikz}
```

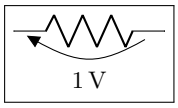



```
\begin{circuitikz}
  \draw (0,0) to[R, i=1<\milli\ampere>] (2,0);
\end{circuitikz}
```

```
\begin{circuitikz}
  \draw (0,0) to[R, i=$\SI{1}{\milli\ampere}$] (2,0);
\end{circuitikz}
```



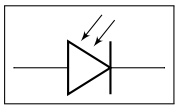

```
\begin{circuitikz}
  \draw (0,0) to[R, v=1<\volt>] (2,0);
\end{circuitikz}
```



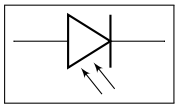

```
\begin{circuitikz}
  \draw (0,0) to[R, v=$\SI{1}{\volt}$] (2,0);
\end{circuitikz}
```

4.9 Mirroring and Inverting

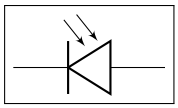
Bipole paths can also mirrored and inverted (or reverted) to change the drawing direction.



```
\begin{circuitikz}
  \draw (0,0) to[pD] (2,0);
\end{circuitikz}
```

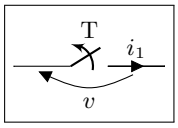


```
\begin{circuitikz}
  \draw (0,0) to[pD, mirror] (2,0);
\end{circuitikz}
```

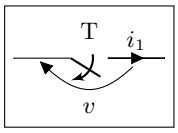


```
\begin{circuitikz}
  \draw (0,0) to[pD, invert] (2,0);
\end{circuitikz}
```

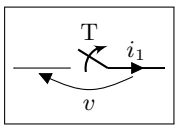
Placing labels, currents and voltages works also, please note, that mirroring and inverting does not influence the positioning of labels and voltages. Labels are by default above/right of the bipole and voltages below/left, respectively.



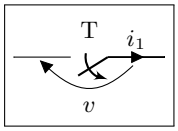

```
\begin{circuitikz}
  \draw (0,0) to[ospst=T, i=$i_1$, v=$v$] (2,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[ospst=T, mirror, i=$i_1$, v=$v$] (2,0);
\end{circuitikz}
```

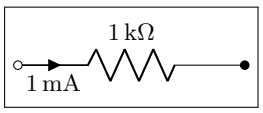



```
\begin{circuitikz}
  \draw (0,0) to[ospst=T, invert, i=$i_1$, v=$v$] (2,0);
\end{circuitikz}
```

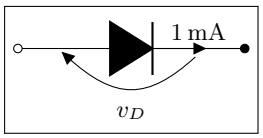



```
\begin{circuitikz}
  \draw (0,0) to[ospst=T,mirror,invert, i=$i_1$, v=$v$] (2,0);
\end{circuitikz}
```

4.10 Putting them together



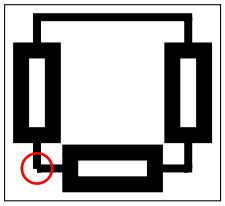

```
\begin{circuitikz}
  \draw (0,0) to[R=1<\kilo\ohm>,
    i>_=1<\milli\ampere>, o-*] (3,0);
\end{circuitikz}
```




```
\begin{circuitikz}
  \draw (0,0) to[D*, v=$v_D$,
    i=1<\milli\ampere>, o-*] (3,0);
\end{circuitikz}
```

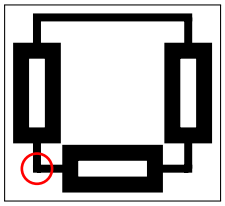
4.11 Line joins between Path Components

Line joins should be calculated correctly - if they are on the same path, and the path is not closed. For example, the following path is not closed correctly (*–cycle* does not work here!):



```
\begin{tikzpicture}[line width=3pt,european]
\draw (0,0) to[R]++(2,0)to[R]++(0,2)
  --++(-2,0)to[R]++(0,-2);
\draw[red,line width=1pt] circle(2mm);
\end{tikzpicture}
```

To correct the line ending, there are support shapes to fill the missing rectangle. They can be used like the support shapes (*,o,d) using a dot (.) on one or both ends of a component (have a look at the last resistor in this example:



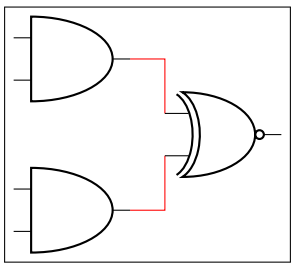
```
\begin{tikzpicture}[line width=3pt,european]
\draw (0,0) to[R]++(2,0)to[R]++(0,2)
  --++(-2,0)to[R,-.]++(0,-2);
\draw[red,line width=1pt] circle(2mm);
\end{tikzpicture}
```

5 Colors

5.1 Shape colors

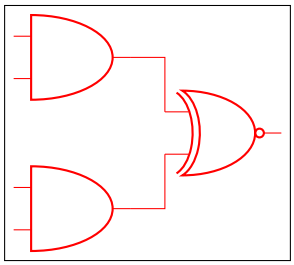
The color of the components is stored in the key `\circuitikzbasekey/color`. CircuiTi*k*Z tries to follow the color set in Ti*k*Z, although sometimes it fails. If you change color in the picture, please do not use just the color name as a style, like `[red]`, but rather assign the style `[color=red]`.

Compare for instance



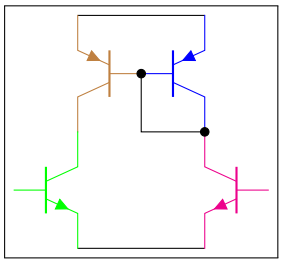
```
\begin{circuitikz} \draw[red]
  (0,2) node[and port] (myand1) {}
  (0,0) node[and port] (myand2) {}
  (2,1) node[xnor port] (myxnor) {}
  (myand1.out) -| (myxnor.in 1)
  (myand2.out) -| (myxnor.in 2)
;\end{circuitikz}
```

and



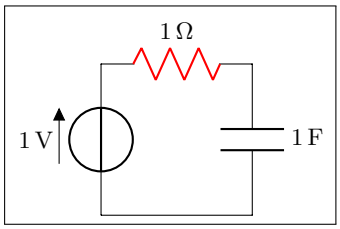
```
\begin{circuitikz} \draw[color=red]
  (0,2) node[and port] (myand1) {}
  (0,0) node[and port] (myand2) {}
  (2,1) node[xnor port] (myxnor) {}
  (myand1.out) -| (myxnor.in 1)
  (myand2.out) -| (myxnor.in 2)
;\end{circuitikz}
```

One can of course change the color *in medias res*:



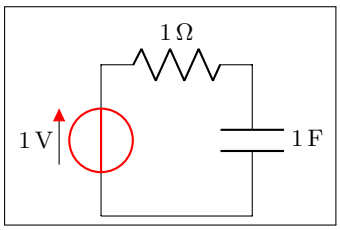
```
\begin{circuitikz} \draw
  (0,0) node[pnp, color=blue] (pnp2) {}
  (pnp2.B) node[pnp, xscale=-1, anchor=B, color=brown] (pnp1) {}
  (pnp1.C) node[npn, anchor=C, color=green] (npn1) {}
  (pnp2.C) node[npn, xscale=-1, anchor=C, color=magenta] (npn2) {}
  (pnp1.E) -- (pnp2.E) (npn1.E) -- (npn2.E)
  (pnp1.B) node[circ] {} |- (pnp2.C) node[circ] {}
;\end{circuitikz}
```

The all-in-one stream of bipoles poses some challanges, as only the actual body of the bipole, and not the connecting lines, will be rendered in the specified color. Also, please notice the curly braces around the `to`:



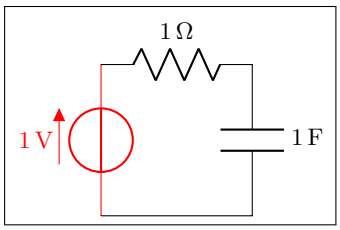

```
\begin{circuitikz} \draw
  (0,0) to[V=1<\volt>] (0,2)
      { to[R=1<\ohm>, color=red] (2,2) }
        to[C=1<\farad>] (2,0) -- (0,0)
;\end{circuitikz}
```

Which, for some bipoles, can be frustrating:




```
\begin{circuitikz} \draw
  (0,0){to[V=1<\volt>, color=red] (0,2) }
        to[R=1<\ohm>] (2,2)
        to[C=1<\farad>] (2,0) -- (0,0)
;\end{circuitikz}
```

The only way out is to specify different paths:

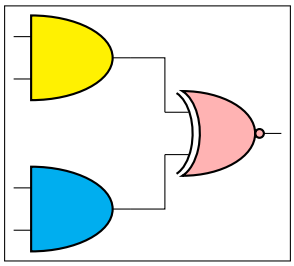



```
\begin{circuitikz} \draw[color=red]
  (0,0) to[V=1<\volt>, color=red] (0,2);
  \draw (0,2) to[R=1<\ohm>] (2,2)
        to[C=1<\farad>] (2,0) -- (0,0)
;\end{circuitikz}
```

And yes: this is a bug and *not* a feature…

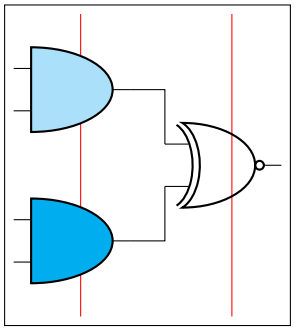
5.2 Fill colors

Since version 0.9.0, you can also fill most shapes with a color (the manual specifies which ones are fillable or not). The syntax is quite intuitive:



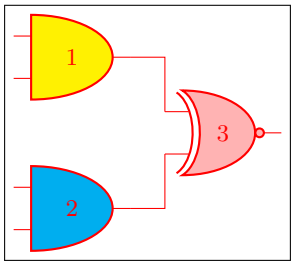
```
\begin{circuitikz} \draw
    (0,2) node[and port, fill=yellow] (myand1) {}
    (0,0) node[and port, fill=cyan] (myand2) {}
    (2,1) node[xnor port,fill=red!30!white] (myxnor) {}
  (myand1.out) -| (myxnor.in 1)
  (myand2.out) -| (myxnor.in 2)
;\end{circuitikz}
```

This fill color will override any color defined by the style (see section 3.3.2). If you want to override a style fill color with no-fill for a specific component, you need to override the style — it's a bit unfortunate but it should be an exceptional thing anyway:



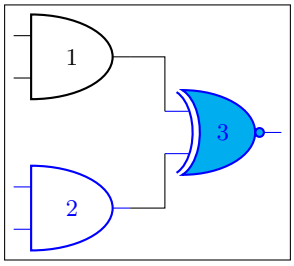
```
\begin{circuitikz}
    \ctikzset{logic ports/fill=cyan!30!white}
    \draw[red] (-0.5,3) -- (-0.5, -1);
    \draw[red] (1.5,3) -- (1.5, -1);
    \draw
    (0,2) node[and port, ] (myand1) {}
    (0,0) node[and port, fill=cyan] (myand2) {}
    (2,1) node[xnor port, circuitikz/logic ports/fill=none] (
        myxnor) {}
  (myand1.out) -| (myxnor.in 1)
  (myand2.out) -| (myxnor.in 2)
;\end{circuitikz}
```

You can combine shape colors with fill colors, too, but you should use the `draw` color option style for this:



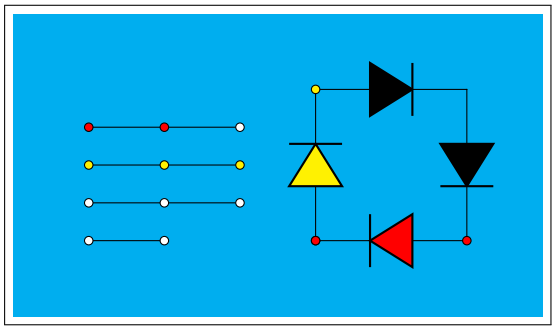

```
\begin{circuitikz} \draw[color=red]
   (0,2) node[and port, fill=yellow] (myand1) {1}
   (0,0) node[and port, fill=cyan] (myand2) {2}
   (2,1) node[xnor port,fill=red!30!white] (myxnor) {3}
  (myand1.out) -| (myxnor.in 1)
  (myand2.out) -| (myxnor.in 2)
;\end{circuitikz}
```

This is because, as you can see from the following example in port 2, you can't specify both a fill and a color in the node (yes, it's a bug too, but it's quite complex to solve given the current circuitTi*k*Z architecture). A workaround is shown in port 3:




```
\begin{circuitikz} \draw
  (0,2) node[and port, color=black] (myand1) {1}
  (0,0) node[and port, color=blue, fill=cyan] (myand2)
      {2}
  (2,1) {[color=blue] node[xnor port, fill=cyan] (myxnor
      ) {3}}
  (myand1.out) -| (myxnor.in 1)
  (myand2.out) -| (myxnor.in 2)
;\end{circuitikz}
```

Notice also that the connection point are always filled, although the color *tries* to follow the color of the filling of the component:



```
\begin{circuitikz}
   \fill[cyan] (0,3.0) rectangle (7,7);
   \draw [fill=yellow, ] (4,4) to [D,o-o] ++(0,2) to[D*, fill=yellow] ++(2,0)
   to[D*] ++(0,-2) to[D, fill=red, o-o] ++(-2,0);
   \draw (1,4) node[ocirc]{} -- ++(1,0) node[ocirc]{};
   \draw (1,4.5) to[short, o-o] ++(1,0) to[short, -o] ++(1,0);
   \draw[fill=yellow] (1,5) to[short, o-o] ++(1,0) to[short, -o] ++(1,0);
   \draw (1,5.5) to[short, fill=red, o-o] ++(1,0) to[short, -o] ++(1,0);
\end{circuitikz}
```

6 FAQ

Q: When using `\tikzexternalize` I get the following error:

```
! Emergency stop.
```

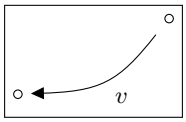
A: The Ti*k*Z manual states:

> Furthermore, the library assumes that all LaTeX pictures are ended with `\end{tikzpicture}`.

Just substitute every occurrence of the environment `circuitikz` with `tikzpicture`. They are actually pretty much the same.

Q: How do I draw the voltage between two nodes?

A: Between any two nodes there is an open circuit!




```
\begin{circuitikz} \draw
 node[ocirc] (A) at (0,0) {}
 node[ocirc] (B) at (2,1) {}
 (A) to[open, v=$v$] (B)
;\end{circuitikz}
```

Q: I cannot write `to[R = $R_1=12V$]` nor `to[ospst = open, 3s]`: I get errors.

A: It is a limitation of the parser.

Use `\def{\eq}{=}` `to[R = $R_1\eq 12V$]` and `to[ospst = open{,} 3s]` instead; see caveat in section 4.1.

Q: I tried to change the direction of the y axis with `yscale=-1`, but the circuit is completely messed up.

A: Yes, it's a known bug (or misfeature, or limitation). See section 1.7. Don't do that.

Q: I tried to put a diode in a `pic`, but it's coming out badly rotated.

A: Yes, it's a known bug (or misfeature, or limitation). See section 1.7. CircuiTi*k*Z is not compatible with `pic`s at this point.

7 Defining new components

Per me si va ne la città dolente,
per me si va ne l'etterno dolore,
per me si va tra la perduta gente.
...
Lasciate ogne speranza, voi ch'intrate.[19]

Big fat warning: this material is reserved to TEX-hackers; do not delve into this if you have no familiarity with (at least) a bit of core TEX programming and to the basic Ti*k*Z layer. You have been warned.

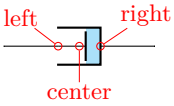
7.1 Suggested setup

The suggested way to start working on a new component is to use the utilities of the CircuiTi*k*Z manual for checking and testing your device. Basically, find (or download) the source code of the last version of CircuiTi*k*Z and find the file `ctikzmanutils.sty`; copy it in your directory and prepare a file like this:

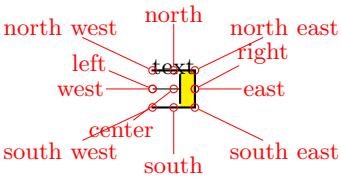
```
\documentclass[a4paper, titlepage]{article}
\usepackage{a4wide} %smaller borders
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\parindent=0pt
\parskip=4pt plus 6pt minus 2pt
\usepackage[siunitx, RPvoltages]{circuitikz}
\usepackage{ctikzmanutils}
\makeatletter
%% Test things here
% defines

% components

% paths
\makeatother

\begin{document}

\circuitdescbip*{damper}{Mechanical damping}{}(left/135/0.2, right/45/0.2,
    center/-90/0.3)

\geolrcoord{dampershape, fill=yellow}

\begin{LTXexample}[varwidth]
\begin{circuitikz}
   \draw (0,0) to[R] ++(2,0)
   to[damper] ++(2,0);
\end{circuitikz}
\end{LTXexample}
\end{document}
```

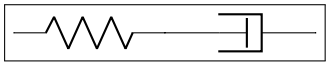
This will compile to something like this (in this case, we are using a couple of existing components to check everything is ok):

[19] https://classicsincontext.wordpress.com/2010/02/28/canto-iii-per-me-si-va-ne-la-citta-dolente/



damper: Mechanical damping, **type: path-style, fillable, nodename: dampershape.** Class: **mechanicals**.





```
\begin{circuitikz}
    \draw (0,0) to[R] ++(2,0)
    to[damper] ++(2,0);
\end{circuitikz}
```

The command **circuitdescbip*** is used to show the component description (you can check the definition and the usage looking at **ctikzmanutils.sty** file, and the **\geolrcoord** is used to show the main anchors (geographical plus **left** and **right**) of the component.

From now on, you can add the new commands for the component between the **\makeatletter** and **\makeatother** commands and, modifying the example, check the results.

7.2 Path-style component

Let's define for example a path style component, like the one suggested by the user **@alex** on TEX stack-exchange site. The component will be a mix of the **damper** and the **spring** components already present.

The first step is to check if we can use the definition already existing for similar elements (for coherence of size) or if we need to define new ones; for this you have to check the file **pgfcirc.defines.tex**: we find

```
\ctikzset{bipoles/spring/height/.initial=.5}
\ctikzset{bipoles/spring/width/.initial=.5}
\ctikzset{bipoles/damper/height/.initial=.35}
\ctikzset{bipoles/damper/length/.initial=.3}
\ctikzset{bipoles/damper/width/.initial=.4}
```

We will use them; at this stage you can decide to add other parameters if you need them. (Notice, however, than although flexibility is good, these parameters should be described in the manual, otherwise they're as good as a fixed number in the code).

To define the new component we will look into **pgfcircbipoles.tex** and we will copy, for example, the definition of the damper into our code, just changing the name:

```
%% mechanical resistor - damper
\pgfcircdeclarebipolescaled{mechanicals}
{}                                  % extra anchors
{\ctikzvalof{bipoles/damper/height}} % depth (under the path line)
{viscoe}                            % name
{\ctikzvalof{bipoles/damper/height}} % height (above the path line)
{\ctikzvalof{bipoles/damper/width}} % width
{
    \pgfpathrectanglecorners{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
        pgf@circ@res@right}{\pgf@circ@res@down}}{\pgfpoint{\pgf@circ@res@right
        }{\pgf@circ@res@up}}
    \pgf@circ@maybefill

    % line into the damper
```

```
    \pgfpathmoveto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@zero}}
    \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
        pgf@circ@res@right}
       {\pgf@circ@res@zero}}
    \pgfusepath{stroke}

    % damper box
    \pgf@circ@setlinewidth{bipoles}{\pgfstartlinewidth}
    \pgfpathmoveto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@down}}
    \pgfpathlineto{\pgfpoint{\pgf@circ@res@right}{\pgf@circ@res@down}}
    \pgfpathlineto{\pgfpoint{\pgf@circ@res@right}{\pgf@circ@res@up}}
    \pgfpathlineto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@up}}

    \pgfsetrectcap
    \pgfsetmiterjoin
    \pgfusepath{stroke}

    % damper vertical element
    \pgfpathmoveto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
        pgf@circ@res@right}
       {.8\pgf@circ@res@down}}
    \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
        pgf@circ@res@right}
       {.8\pgf@circ@res@up}}
    \pgfsetbuttcap
    \pgfusepath{stroke}

}
```

This command will define a shape that is named `viscoeshape`, with all the correct geographical anchors based on the depth, height and width defined in the parameters of `\pgfcircdeclarebipolescaled`. Moreover, the element is assigned to the class `mechanicals` for styling.

To be coherent with the styling, you should use (when needed) the length `\pgf@circ@scaled@Rlen` as the "basic" length for drawing, using the fill functions defined at the start of `pgfcirc.defines.tex` to fill and stroke — so that the operation will follow the style parameters and, finally, use the macro `\pgf@circ@setlinewidth` to set the line thickness /the first argument is the "legacy" class, if you do not want to assign one you can use the pseudo-legacy class `none`.

The anchors for the bipole (which then set the lengths `\pgf@circ@res@left`) are already scaled for your use. You can use these lenghts (which defines, normally, the geographical anchors of the element) to draw your shapes.

This is not sufficient for using the element in a `to[]` path command; you need to "activate" it with (this commands are normally in `pgfcircpath.tex`):

```
\def\pgf@circ@viscoe@path#1{\pgf@circ@bipole@path{viscoe}{#1}}
\compattikzset{viscoe/.style = {\circuitikzbasekey,
       /tikz/to path=\pgf@circ@dviscoe@path, l=#1}}
```

And now you can show it with:

```
\circuitdescbip*{viscoe}{Mechanical viscoelastic element}{}(left/135/0.2,
    right/45/0.2, center/-90/0.3)

\geolrcoord{viscoeshape, fill=yellow}

\begin{LTXexample}[varwidth]
```

```
\begin{circuitikz}
    \draw (0,0) to[spring] ++(2,0)
    to[viscoe] ++(2,0);
\end{circuitikz}
\end{LTXexample}
```

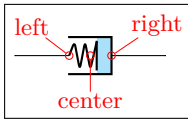
Obviously, at first you you just have a component that is the same as the one you copied with another name. It is now just a matter of modifying it so that it has the desired shape; in the example above you can already see the new symbol after the changes.

When doing the drawing, the `\pgfcircdeclarebipole` will setup the lengths `\pgf@circ@res@right` and `\pgf@circ@res@up` as the x-y coordinates of the upper right corner, and `\pgf@circ@res@left` and `\pgf@circ@res@down` as the x-y coordinates of the lower left corner of your shape. The `center` coordinate is usually at $(0pt, 0pt)$.

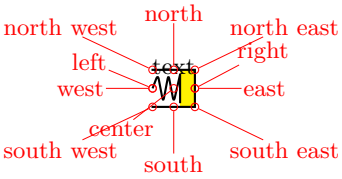
Looking at the implementation of the `spring` element, a possible implementation is changing the lines between lines 12 and 16 with:

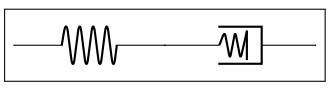
```
    % spring into the damper
    \pgfscope
        \pgfpathmoveto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@zero}}
        \pgf@circ@setlinewidth{bipoles}{\pgfstartlinewidth}
        \pgfsetcornersarced{\pgfpoint{.25\pgf@circ@res@up}{.25\pgf@circ@res@up}}
        \pgfpathlineto{\pgfpoint{.75\pgf@circ@res@left}{.75\pgf@circ@res@up}}
        \pgfpathlineto{\pgfpoint{.5\pgf@circ@res@left}{-.75\pgf@circ@res@up}}
        \pgfpathlineto{\pgfpoint{.25\pgf@circ@res@left}{.75\pgf@circ@res@up}}
        \pgfpathlineto{\pgfpoint{0pt}{-.75\pgf@circ@res@up}}
        \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
            pgf@circ@res@right}{.75\pgf@circ@res@up}}
        \pgfusepath{stroke}
    \endpgfscope
```

which leads to:



viscoe: Mechanical viscoelastic element, `type: path-style, fillable, nodename: viscoeshape.` Class: `mechanicals`.





```
\begin{circuitikz}
    \draw (0,0) to[spring] ++(2,0)
    to[viscoe] ++(2,0);
\end{circuitikz}
```

As a final note, notice that the `viscoe` element is already added to the standard package.

7.3 Node-style component

Adding a node-style component is much more straightforward. Just define it by following examples in, for example, `pgfcirctripoles.tex` or the other files; be careful that you should define all the geographical anchors of the shape if you want that the Ti*k*Z positioning options (like `left`, `above`, etc.) behave correctly with your component.

To have a scalable component, for example in the `transistors` class, you should use something like

```
\savedmacro{\ctikzclass}{\edef\ctikzclass{transistors}}
\saveddimen{\scaledRlen}{\pgfmathsetlength{\pgf@x}{\ctikzvalof{\
    ctikzclass/scale}\pgf@circ@Rlen}}
```

at the start of anchors and macros definition, and use (for example, the exact code will change greatly depending on your component):

```
\savedanchor\northeast{% upper right
    \pgfmathsetlength{\pgf@circ@scaled@Rlen}{\ctikzvalof{\ctikzclass/
        scale}\pgf@circ@Rlen}
    \pgf@y=\pgf@circ@scaled@Rlen
    \pgf@y=0.5\pgf@y
    \pgf@x=0.3\pgf@y
}
```

in all the `savedanchors`.

Then, in the drawing part, you should start with:

```
\pgf@circ@scaled@Rlen=\scaledRlen
```

and then use `\pgf@circ@scaled@Rlen` (or the anchors) as default lengths while you draw it.

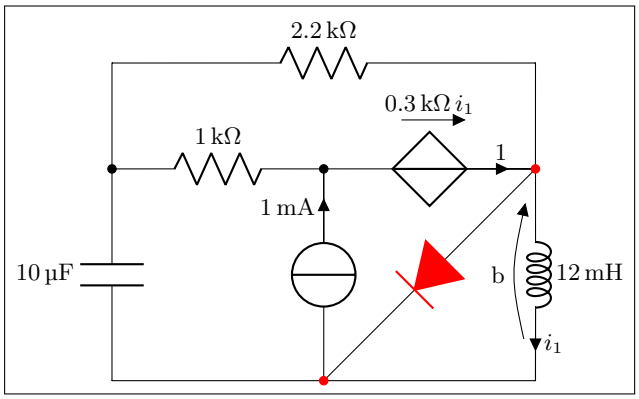
7.3.1 Finishing your work

Once you have a satisfactory element, you should

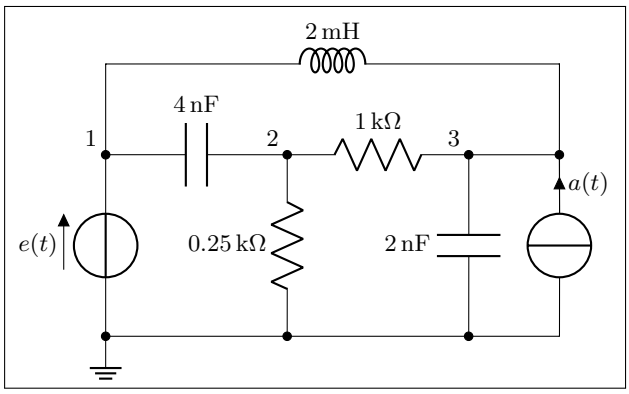
- Clean up your code;
- write a piece of documentation explaining its use, with an example;
- Propose the element for inclusion in the GitHub page of the project (you will have to license this as explained in that page, of course).

The best way of contributing is forking the project, adding your component in the correct files, modifying the manual and creating a pull request for the developers to merge. Anyway, if this is a problem, just open an issue and someone (when they have time…) will answer.

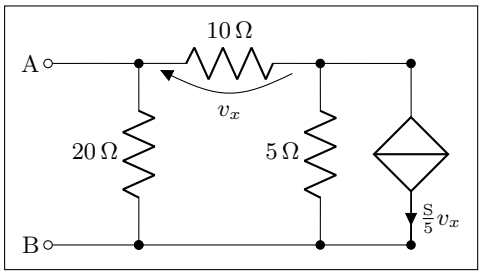
8 Examples



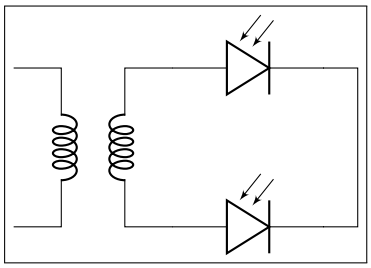

```
\begin{circuitikz}[scale=1.4]\draw
  (0,0) to[C, l=10<\micro\farad>] (0,2) -- (0,3)
        to[R, l=2.2<\kilo\ohm>] (4,3) -- (4,2)
        to[L, l=12<\milli\henry>, i=$i_1$,v=b] (4,0) -- (0,0)
  (4,2) { to[D*, *-*, color=red] (2,0) }
  (0,2) to[R, l=1<\kilo\ohm>, *-] (2,2)
        to[cV, i=1,v=$\SI{.3}{\kilo\ohm}\, i_1$] (4,2)
  (2,0) to[I, i=1<\milli\ampere>, -*] (2,2)
;\end{circuitikz}
```



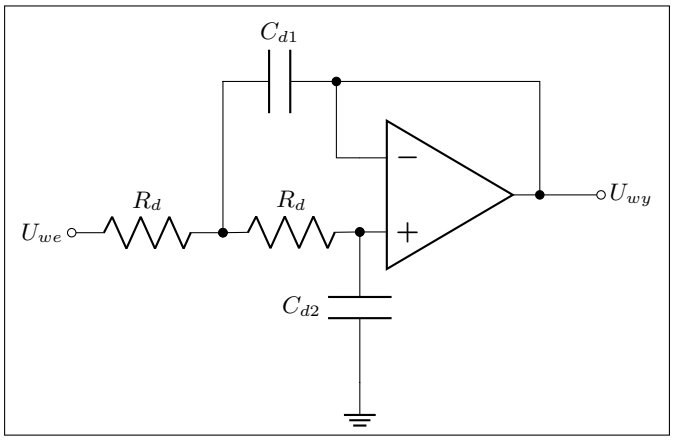

```
\begin{circuitikz}[scale=1.2]\draw
  (0,0) node[ground] {}
        to[V=$e(t)$, *-*] (0,2) to[C=4<\nano\farad>] (2,2)
        to[R, l_=.25<\kilo\ohm>, *-*] (2,0)
  (2,2) to[R=1<\kilo\ohm>] (4,2)
        to[C, l_=2<\nano\farad>, *-*] (4,0)
  (5,0) to[I, i_=$a(t)$, -*] (5,2) -- (4,2)
  (0,0) -- (5,0)
  (0,2) -- (0,3) to[L, l=2<\milli\henry>] (5,3) -- (5,2)

 {[anchor=south east] (0,2) node {1} (2,2) node {2} (4,2) node {3}}
;\end{circuitikz}
```



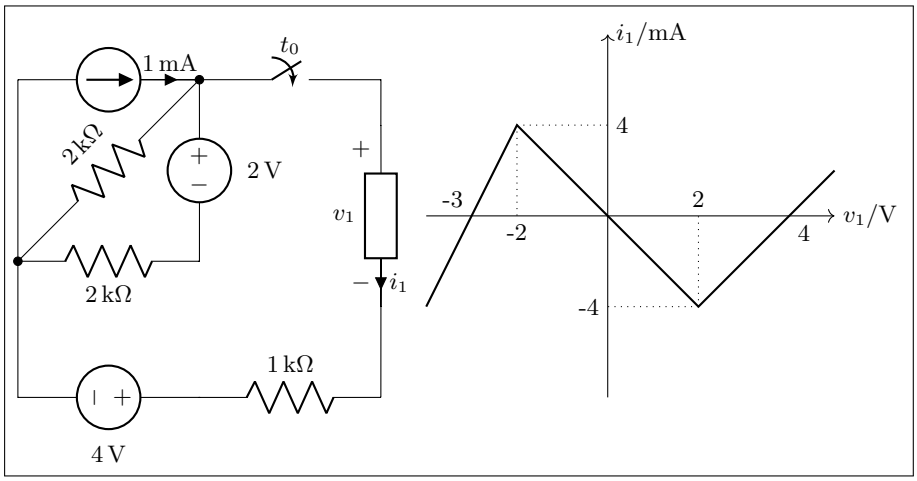

```
\begin{circuitikz}[scale=1.2]\draw
  (0,0) node[anchor=east] {B}
        to[short, o-*] (1,0)
        to[R=20<\ohm>, *-*] (1,2)
        to[R=10<\ohm>, v=$v_x$] (3,2) -- (4,2)
        to[cI=$\frac{\siemens}{5} v_x$, *-*] (4,0) -- (3,0)
        to[R=5<\ohm>, *-*] (3,2)
  (3,0) -- (1,0)
  (1,2) to[short, -o] (0,2) node[anchor=east]{A}
;\end{circuitikz}
```



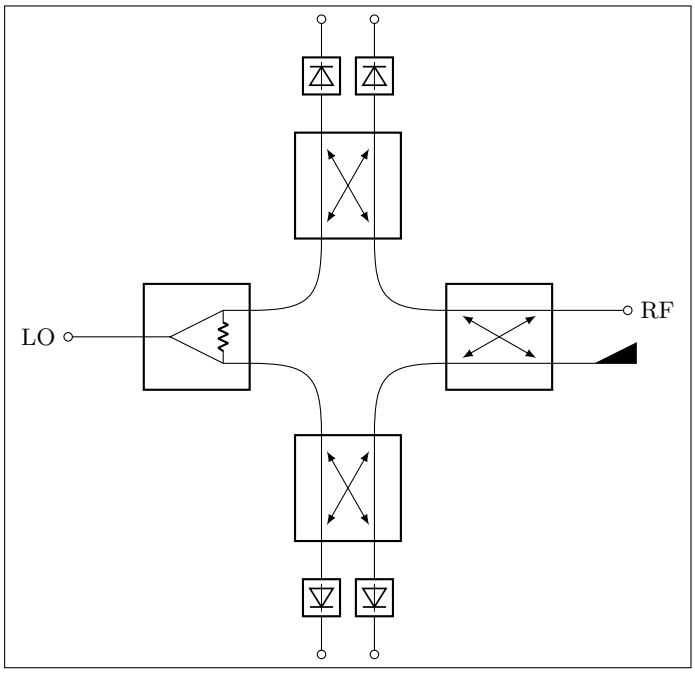
```
\begin{circuitikz}[scale=1]\draw
   (0,0) node[transformer] (T) {}
   (T.B2) to[pD] ($(T.B2)+(2,0)$) -| (3.5, -1)
   (T.B1) to[pD] ($(T.B1)+(2,0)$) -| (3.5, -1)
;\end{circuitikz}
```



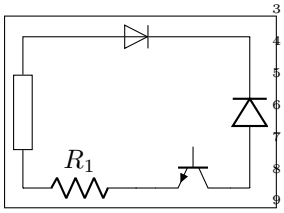

```
\begin{circuitikz}[scale=1]\draw
   (5,.5) node [op amp] (opamp) {}
   (0,0) node [left] {$U_{we}$} to [R, l=$R_d$, o-*] (2,0)
   to [R, l=$R_d$, *-*] (opamp.+)
   to [C, l_=$C_{d2}$, *-] ($(opamp.+)+(0,-2)$) node [ground] {}
   (opamp.out) |- (3.5,2) to [C, l_=$C_{d1}$, *-] (2,2) to [short] (2,0)
   (opamp.-) -| (3.5,2)
   (opamp.out) to [short, *-o] (7,.5) node [right] {$U_{wy}$}
;\end{circuitikz}
```




```
\begin{circuitikz}[scale=1.2, american]\draw
 (0,2) to[I=1<\milli\ampere>] (2,2)
       to[R, l_=2<\kilo\ohm>, *-*] (0,0)
       to[R, l_=2<\kilo\ohm>] (2,0)
       to[V, v_=2<\volt>] (2,2)
       to[cspst, l=$t_0$] (4,2) -- (4,1.5)
       to [generic, i=$i_1$, v=$v_1$] (4,-.5) -- (4,-1.5)
 (0,2) -- (0,-1.5) to[V, v_=4<\volt>] (2,-1.5)
       to [R, l=1<\kilo\ohm>] (4,-1.5);

  \begin{scope}[xshift=6.5cm, yshift=.5cm]
   \draw [->] (-2,0) -- (2.5,0) node[anchor=west] {$v_1/\volt$};
   \draw [->] (0,-2) -- (0,2) node[anchor=west] {$i_1/\SI{}{\milli\ampere}$} ;
   \draw (-1,0) node[anchor=north] {-2} (1,0) node[anchor=south] {2}
        (0,1) node[anchor=west] {4} (0,-1) node[anchor=east] {-4}
        (2,0) node[anchor=north west] {4}
        (-1.5,0) node[anchor=south east] {-3};
   \draw [thick] (-2,-1) -- (-1,1) -- (1,-1) -- (2,0) -- (2.5,.5);
   \draw [dotted] (-1,1) -- (-1,0) (1,-1) -- (1,0)
        (-1,1) -- (0,1) (1,-1) -- (0,-1);
  \end{scope}
\end{circuitikz}
```




```
\begin{circuitikz}[scale=1]
    \ctikzset{bipoles/detector/width=.35}
    \ctikzset{quadpoles/coupler/width=1}
    \ctikzset{quadpoles/coupler/height=1}
    \ctikzset{tripoles/wilkinson/width=1}
    \ctikzset{tripoles/wilkinson/height=1}
    %\draw[help lines,red,thin,dotted] (0,-5) grid (5,5);
    \draw
    (-2,0) node[wilkinson](w1){}
    (2,0) node[coupler] (c1) {}
    (0,2) node[coupler,rotate=90] (c2) {}
    (0,-2) node[coupler,rotate=90] (c3) {}
    (w1.out1) .. controls ++(0.8,0) and ++(0,0.8) .. (c3.3)
    (w1.out2) .. controls ++(0.8,0) and ++(0,-0.8) .. (c2.4)
    (c1.1) .. controls ++(-0.8,0) and ++(0,0.8) .. (c3.2)
    (c1.4) .. controls ++(-0.8,0) and ++(0,-0.8) .. (c2.1)
    (w1.in) to[short,-o] ++(-1,0)
    (w1.in) node[left=30] {LO}
    (c1.2) node[match,yscale=1] {}
    (c1.3) to[short,-o] ++(1,0)
    (c1.3) node[right=30] {RF}
    (c2.3) to[detector,-o] ++(0,1.5)
    (c2.2) to[detector,-o] ++(0,1.5)
    (c3.1) to[detector,-o] ++(0,-1.5)
    (c3.4) to[detector,-o] ++(0,-1.5)
    ;
\end{circuitikz}
```




```
\documentclass{standalone}

\usepackage{tikz}
\usetikzlibrary{circuits.ee.IEC}
\usetikzlibrary{positioning}

\usepackage[compatibility]{circuitikzgit}
\ctikzset{bipoles/length=.9cm}

\begin{document}
 \begin{tikzpicture}[circuit ee IEC]
  \draw (0,0) to [resistor={name=R}] (0,2)
   to[diode={name=D}] (3,2);
  \draw (0,0) to[*R=$R_1$] (1.5,0) to[*Tnpn] (3,0)
    to[*D](3,2);
 \end{tikzpicture}
\end{document}
```

9 Changelog and Release Notes

The major changes among the different circuitikz versions are listed here. See https://github.com/circuitikz/circuitikz/commits for a full list of changes.

- Version 0.9.5 (unrelease)
  - Bumped version number
- Version 0.9.4 (2019-08-30)
  This release introduces two changes: a big one, which is the styling of the components (please look at the manual for details) and a change to how voltage labels and arrows are positioned. This one should be backward compatible *unless* you used `voltage shift` introduced in 0.9.0, which was broken when using the global `scale` parameter.
  The styling additions are quite big, and, although in principle they are backward compatible, you can find corner cases where they are not, especially if you used to change parameters for `pgfcirc.defines.tex`; so a snapshot for the 0.9.3 version is available.
  - Fixed a bug with "inline" gyrators, now the circle will not overlap
  - Fixed a bug in input anchors of european not ports
  - Fixed "tlinestub" so that it has the same default size than "tline" (TL)
  - Fixed the "transistor arrows at end" feature, added to styling
  - Changed the behavior of "voltage shift" and voltage label positioning to be more robust
  - Added several new anchors for "elmech" element
  - Several minor fixes in some component drawings to allow fill and thickness styles
  - Add 0.9.3 version snapshots.
  - Added styling of relative size of components (at a global or local level)
  - Added styling for fill color and thickeness
  - Added style files
- Version 0.9.3 (2019-07-13)
  - Added the option to have "dotless" P-MOS (to use with arrowmos option)
  - Fixed a (puzzling) problem with coupler2
  - Fixed a compatibility problem with newer PGF (>3.0.1a)
- Version 0.9.2 (2019-06-21)
  - (hopefully) fixed ConTeXt compatibility. Most new functionality is not tested; testers and developers for the ConTeXt side are needed.
  - Added old ConTeXt version for 0.8.3
  - Added tailless ground
- Version 0.9.1 (2019-06-16)
  - Added old LaTeX versions for 0.8.3, 0.7, 0.6 and 0.4
  - Added the option to have inline transformers and gyrators
  - Added rotary switches
  - Added more configurable bipole nodes (connectors) and more shapes
  - Added 7-segment displays
  - Added vacuum tubes by J. op den Brouw
  - Made the open shape of dcisources configurable
  - Made the arrows on vcc and vee configurable

  - Fixed anchors of diamondpole nodes
  - Fixed a bug (#205) about unstable anchors in the chip components
  - Fixed a regression in label placement for some values of scaling
  - Fixed problems with cute switches anchors

- Version 0.9.0 (2019-05-10)
  - Added Romano Giannetti as contributor
  - Added a CONTRIBUTING file
  - Added options for solving the voltage direction problems.
  - Adjusted ground symbols to better match ISO standard, added new symbols
  - Added new sources (cute european versions, noise sources)
  - Added new types of amplifiers, and option to flip inputs and outputs
  - Added bidirectional diodes (diac) thanks to Andre Lucas Chinazzo
  - Added L,R,C sensors (with european, american and cute variants)
  - Added stacked labels (thanks to the original work by Claudio Fiandrino)
  - Make the position of voltage symbols adjustable
  - Make the position of arrows in FETs and BJTs adjustable
  - Added chips (DIP, QFP) with a generic number of pins
  - Added special anchors for transformers (and fixed the wrong center anchor)
  - Changed the logical port implementation to multiple inputs (thanks to John Kormylo) with border anchors.
  - Added several symbols: bulb, new switches, new antennas, loudspeaker, microphone, coaxial connector, viscoelastic element
  - Make most components fillable
  - Added the oscilloscope component and several new instruments
  - Added viscoelastic element
  - Added a manual section on how to define new components
  - Fixed american voltage symbols and allow to customize them
  - Fixed placement of straightlabels in several cases
  - Fixed a bug about straightlabels (thanks to @fotesan)
  - Fixed labels spacing so that they are independent on scale factor
  - Fixed the position of text labels in amplifiers

- Version 0.8.3 (2017-05-28)
  - Removed unwanted lines at to-paths if the starting point is a node without a explicit anchor.
  - Fixed scaling option, now all parts are scaled by bipoles/length
  - Surge arrester appears no more if a to path is used without []-options
  - Fixed current placement now possible with paths at an angle of around 280°
  - Fixed voltage placement now possible with paths at an angle of around 280°
  - Fixed label and annotation placement (at some angles position not changable)
  - Adjustable default distance for straight-voltages: 'bipoles/voltage/straight label distance'
  - Added Symbol for bandstop filter
  - New annotation type to show flows using f=… like currents, can be used for thermal, power or current flows

- Version 0.8.2 (2017-05-01)

  - Fixes pgfkeys error using alternatively specified mixed colors(see pgfplots manual section “4.7.5 Colors”)
  - Added new switches “ncs” and “nos”
  - Reworked arrows at spst-switches
  - Fixed direction of controlled american voltage source
  - “v<=” and “i<=” do not rotate the sources anymore(see them as “counting direction indication”, this can be different then the shape orientation); Use the option “invert” to change the direction of the source/apperance of the shape.
  - current label “i=” can now be used independent of the regular label “l=” at current sources
  - rewrite of current arrow placement. Current arrows can now also be rotated on zero-length paths
  - New DIN/EN compliant operational amplifier symbol “en amp”
- Version 0.8.1 (2017-03-25)
  - Fixed unwanted line through components if target coordinate is a name of a node
  - Fixed position of labels with subscript letters.
  - Absolute distance calculation in terms of ex at rotated labels
  - Fixed label for transistor paths (no label drawn)
- Version 0.8 (2017-03-08)
  - Allow use of voltage label at a [short]
  - Correct line joins between path components (to[…])
  - New Pole-shape .-. to fill perpendicular joins
  - Fixed direction of controlled american current source
  - Fixed incorrect scaling of magnetron
  - Fixed: Number of american inductor coils not adjustable
  - Fixed Battery Symbols and added new battery2 symbol
  - Added non-inverting Schmitttrigger
- Version 0.7 (2016-09-08)
  - Added second annotation label, showing, e.g., the value of an component
  - Added new symbol: magnetron
  - Fixed name conflict of diamond shape with tikz.shapes package
  - Fixed varcap symbol at small scalings
  - New packet-option ”straightvoltages, to draw straight(no curved) voltage arrows
  - New option “invert” to revert the node direction at paths
  - Fixed american voltage label at special sources and battery
  - Fixed/rotated battery symbol(longer lines by default positive voltage)
  - New symbol Schmitttrigger
- Version 0.6 (2016-06-06)
  - Added Mechanical Symbols (damper,mass,spring)
  - Added new connection style diamond, use (d-d)
  - Added new sources voosource and ioosource (double zero-style)
  - All diode can now drawn in a stroked way, just use globel option “strokediode” or stroke instead of full/empty, or D-. Use this option for compliance with DIN standard EN-60617

  - Improved Shape of Diodes:tunnel diode, Zener diode, schottky diode (bit longer lines at cathode)
  - Reworked igbt: New anchors G,gate and new L-shaped form Lnigbt, Lpigbt
  - Improved shape of all fet-transistors and mirrored p-chan fets as default, as pnp, pmos, pfet are already. This means a backward-incompatibility, but smaller code, because p-channels mosfet are by default in the correct direction(source at top). Just remove the 'yscale=-1' from your p-chan fets at old pictures.

- Version 0.5 (2016-04-24)
  - new option boxed and dashed for hf-symbols
  - new option solderdot to enable/disable solderdot at source port of some fets
  - new parts: photovoltaic source, piezo crystal, electrolytic capacitor, electromechanical device(motor, generator)
  - corrected voltage and current direction(option to use old behaviour)
  - option to show body diode at fet transistors

- Version 0.4
  - minor improvements to documentation
  - comply with TDS
  - merge high frequency symbols by Stefan Erhardt
  - added switch (not opening nor closing)
  - added solder dot in some transistors
  - improved ConTeXt compatibility

- Version 0.3.1
  - different management of color…
  - fixed typo in documentation
  - fixed an error in the angle computation in voltage and current routines
  - fixed problem with label size when scaling a tikz picture
  - added gas filled surge arrester
  - added compatibility option to work with Tikz's own circuit library
  - fixed infinite in arctan computation

- Version 0.3.0
  - fixed gate node for a few transistors
  - added mixer
  - added fully differential op amp (by Kristofer M. Monisit)
  - now general settings for the drawing of voltage can be overridden for specific components
  - made arrows more homogeneous (either the current one, or latex' bt pgf)
  - added the single battery cell
  - added fuse and asymmetric fuse
  - added toggle switch
  - added varistor, photoresistor, thermocouple, push button
  - added thermistor, thermistor ptc, thermistor ptc
  - fixed misalignment of voltage label in vertical bipoles with names
  - added isfet
  - added noiseless, protective, chassis, signal and reference grounds (Luigi «Liverpool»)

- Version 0.2.4
  - added square voltage source (contributed by Alistair Kwan)
  - added buffer and plain amplifier (contributed by Danilo Piazzalunga)
  - added squid and barrier (contributed by Cor Molenaar)
  - added antenna and transmission line symbols contributed by Leonardo Azzinnari
  - added the changeover switch spdt (suggestion of Fabio Maria Antoniali)
  - rename of context.tex and context.pdf (thanks to Karl Berry)
  - updated the email address
  - in documentation, fixed wrong (non-standard) labelling of the axis in an example (thanks to prof. Claudio Beccaria)
  - fixed scaling inconsistencies in quadrupoles
  - fixed division by zero error on certain vertical paths
  - introduced options straighlabels, rotatelabels, smartlabels
- Version 0.2.3
  - fixed compatibility problem with label option from tikz
  - Fixed resizing problem for shape ground
  - Variable capacitor
  - polarized capacitor
  - ConTeXt support (read the manual!)
  - nfet, nigfete, nigfetd, pfet, pigfete, pigfetd (contribution of Clemens Helfmeier and Theodor Borsche)
  - njfet, pjfet (contribution of Danilo Piazzalunga)
  - pigbt, nigbt
  - *backward incompatibility* potentiometer is now the standard resistor-with-arrow-in-the-middle; the old potentiometer is now known as variable resistor (or vR), similarly to variable inductor and variable capacitor
  - triac, thyristor, memristor
  - new property “name” for bipoles
  - fixed voltage problem for batteries in american voltage mode
  - european logic gates
  - *backward incompatibility* new american standard inductor. Old american inductor now called “cute inductor”
  - *backward incompatibility* transformer now linked with the chosen type of inductor, and version with core, too. Similarly for variable inductor
  - *backward incompatibility* styles for selecting shape variants now end are in the plural to avoid conflict with paths
  - new placing option for some tripoles (mostly transistors)
  - mirror path style
- Version 0.2.2 - 20090520
  - Added the shape for lamps.
  - Added options `europeanresistor`, `europeaninductor`, `americanresistor` and `americaninductor`, with corresponding styles.
  - FIXED: error in transistor arrow positioning and direction under negative `xscale` and `yscale`.
- Version 0.2.1 - 20090503

  - Op-amps added
  - added options arrowmos and noarrowmos, to add arrows to pmos and nmos

- Version 0.2 - 20090417 First public release on CTAN
  - *Backward incompatibility*: labels ending with `:`*angle* are not parsed for positioning anymore.
  - Full use of Ti*k*Z keyval features.
  - White background is not filled anymore: now the network can be drawn on a background picture as well.
  - Several new components added (logical ports, transistors, double bipoles, …).
  - Color support.
  - Integration with `{siunitx}`.
  - `Voltage, american style.`
  - `Better code, perhaps. General cleanup at the very least.`

- Version 0.1 - 2007-10-29 First public release

Index of the components

adc, 55
adder, 54
afuse, 49
ageneric, 31
american and port, 86
american buffer port, 86
american controlled current source, 41
american controlled voltage source, 41
american current source, 40
american gas filled surge arrester, 49
american inductive sensor, *see* sL
american inductor, *see* L
american nand port, 86
american nor port, 86
american not port, 86
american or port, 86
american potentiometer, *see* pR, *see* pR
american resisitive sensor, *see* sR
american resistor, *see* resistor, *see* R
american voltage source, 39
american xnor port, 86
american xor port, 86
ammeter, 20, 44
amp, 55
antenna, 70
asymmetric fuse, *see* afuse

bandpass, 54
bandstop, 54
bare7seg, 92
bareantenna, 69
bareRXantenna, 69
bareTXantenna, 69
barrier, 49
battery, 39
battery1, 39
battery2, 39
biD*, *see* full bidirectionaldiode
biDo, *see* empty bidirectionaldiode
bnc, 52, 53
buffer, 76
bulb, 50

C, *see* capacitor
capacitive sensor, 33
capacitor, 33
cceI, *see* cute european controlled current source
cceV, *see* cute european controlled voltage source
ccgsw, *see* cute closing switch
ccsw, *see* cute closed switch
ceI, *see* cute european current source
ceV, *see* cute european voltage source
cground, 28
circ, 53
circulator, 54
cisourceC, *see* cute european controlled current source
cisourcesin, *see* controlled sinusoidal current source
closing switch, 80
cogsw, *see* cute opening switch
controlled isourcesin, *see* controlled sinusoidal current source
controlled sinusoidal current source, 42
controlled sinusoidal voltage source, 41
controlled vsourcesin, *see* controlled sinusoidal voltage source
cosw, *see* cute open switch
coupler, 56
coupler2, 56
crossing, 51
csI, *see* controlled sinusoidal current source
cspst, *see* closing switch
csV, *see* controlled sinusoidal voltage source
currarrow, 52
cute choke, 34
cute closed switch, 81
cute closing switch, 81
cute european controlled current source, 41
cute european controlled voltage source, 41
cute european current source, 40
cute european voltage source, 39
cute inductive sensor, *see* sL
cute inductor, *see* L
cute open switch, 81
cute opening switch, 81
cute spdt down, 81
cute spdt down arrow, 22, 81
cute spdt mid, 81
cute spdt mid arrow, 81
cute spdt up, 81
cute spdt up arrow, 81
cvsourceC, *see* cute european controlled voltage source
cvsourcesin, *see* controlled sinusoidal voltage source

D*, *see* full diode
D-, *see* stroke diode
dac, 55
damper, 49, 116
dcisource, 43
dcvsource, 43
detector, 55
diamondpole, 53
diodetube, 65
diodetube,filament, 66
diodetube,filament,nocathode, 66
diodetube,fullcathode, 67

dipchip, 89
Do, *see* empty diode
dsp, 55

eC, *see* ecapacitor
ecapacitor, 33
ecsource, *see* empty controlled source
eground, 28
eground2, 28
elko, *see* ecapacitor
elmech, 71
empty bidirectionaldiode, 36
empty controlled source, 41
empty diode, 36
empty led, 36
empty photodiode, 36
empty Schottky diode, 36
empty thyristor, 38
empty triac, 38
empty tunnel diode, 36
empty varcap, 36
empty Zener diode, 36
empty ZZener diode, 36
en amp, 75
esource, 43
european and port, 86
european buffer port, 87
european controlled current source, 41
european controlled voltage source, 41
european current source, 39
european gas filled surge arrester, 49
european inductive sensor, *see* sL
european inductor, *see* L
european nand port, 86
european nor port, 86
european not port, 87
european or port, 86
european potentiometer, *see* pR
european resistive sensor, *see* sR
european resistor, *see* R
european voltage source, 39
european xnor port, 87
european xor port, 87

fd inst amp, 76
fd op amp, 75
fft, 55
fourport, 56
full bidirectionaldiode, 37
full diode, 36
full led, 37
full photodiode, 36
full Schottky diode, 36
full thyristor, 38
full triac, 38
full tunnel diode, 36
full varcap, 37
full Zener diode, 36
full ZZener diode, 36
fuse, 49

generic, 30
gm amp, 75
ground, 28
gyrator, 73

hemt, 58
highpass, 55

iloop, 45
iloop2, 45
inputarrow, 52
inst amp, 75
inst amp ra, 76
invschmitt, 87
ioosource, 43
isfet, 62
isourceC, *see* cute european current source
isourceN, *see* noise current source
isourcesin, *see* sinusoidal current source

jump crossing, 51

L, 34
lamp, 50
leD*, *see* full led
leD-, *see* stroke led
leDo, *see* empty led
Lnigbt, 59
loudspeaker, 50
lowpass, 55
Lpigbt, 59

magnetron, 68
mass, 49
match, 70
memristor, 31
mic, 50
mixer, 54
Mr, *see* memristor
mslstub, 69
msport, 70
msrstub, 70
mstline, 69

ncpb, *see* normally closed push button
ncs, *see* normal closed switch
nfet, 61
nground, 28
nI, *see* noise current source
nigbt, 59
nigfetd, 61
nigfete, 61
nigfete,solderdot, 61
nigfetebulk, 61
njfet, 62
nmos, 58, 60
noise current source, 42
noise voltage source, 42
nopb, *see* push button

normal closed switch, 80
normal open switch, 80
normally closed push button, 80
normally open push button, *see* push button
nos, *see* normal open switch
npn, 22, 58
npn,photo, 59
nV, *see* noise voltage source

ocirc, 53
odiamondpole, 53
ohmmeter, 44
op amp, 75
open, 30
opening switch, 80
oscillator, 54
oscope, 44
ospst, *see* opening switch
osquarepole, 53

pC, *see* polar capacitor
pD*, *see* full photodiode
pD-, *see* stroke photodiode
pDo, *see* empty photodiode
pentode, 66
pentode suppressor to cathode, 66
pfet, 62
pground, 28
phaseshifter, 55
photoresistor, *see* phR
phR, 31
piattenuator, 55
piezoelectric, 33
pigbt, 59
pigfetd, 62
pigfete, 62
pigfetebulk, 62
pjfet, 62
plain amp, 22, 76
plain crossing, 51
pmos, 58, 60
pmos,emptycircle, 60
pmos,nocircle,arrowmos, 60
pnp, 59
pnp,photo, 59
polar capacitor, 33
pR, 20, *see* pR, 31
push button, 80
pvsource, 43
PZ, *see* piezoelectric

qfpchip, 89
qiprobe, 44
qpprobe, 44
qvprobe, 44

R, *see* resistor, 31
resistor, 20
rground, 28
rmeter, 44
rmeterwa, 44
rotaryswitch, 22, 83
rxantenna, 70

sC, *see* capacitive sensor
schmitt, 87
sD*, *see* full Schottky diode
sD-, *see* stroke Schottky diode
sDo, *see* empty Schottky diode
sground, 28
short, 30
sI, *see* sinusoidal current source
sinusoidal current source, 40
sinusoidal voltage source, 40
sL, 34
smeter, 44
spdt, 80
spring, 49
spst, *see* switch
square voltage source, 43
squarepole, 53
squid, 49
sqV, *see* square voltage source
sR, 31
stroke diode, 37
stroke led, 37
stroke photodiode, 37
stroke Schottky diode, 37
stroke thyristor, 38
stroke tunnel diode, 37
stroke varcap, 37
stroke Zener diode, 37
stroke ZZener diode, 37
sV, *see* sinusoidal voltage source
switch, 80

tattenuator, 55
tD*, *see* full tunnel diode
tD-, *see* stroke tunnel diode
tDo, *see* empty tunnel diode
tetrode, 65
tgeneric, 30
tground, 28
thermistor, *see* thR
thermistor ntc, *see* thRn
thermistor ptc, *see* thRp
thermocouple, 49
thR, 32
thRn, 32
thRp, 32
thyristor, 38
TL, 70
tlground, 28
tline, *see* TL
tlinestub, 70
toggle switch, 80
Tr, *see* triac
Tr*, *see* full triac

transformer, 72, 73
transformer core, 73
transmission line, *see* TL
triac, 38
triode, 65
Tro, *see* empty triac
tV, *see* vsourcetri
twoport, 54
txantenna, 70
Ty, *see* thyristor
Ty*, *see* full thyristor
Ty-, *see* stroke thyristor
Tyo, *see* empty thyristor

vamp, 55
variable american inductor, *see* vL
variable american resistor, *see* vR
variable capacitor, 33
variable cute inductor, *see* vL
variable european inductor, *see* vL
variable european resistor, *see* vR
varistor, 31
vC, *see* variable capacitor
VC*, *see* full varcap
VC-, *see* stroke varcap
vcc, 29
VCo, *see* empty varcap
vco, 54
vee, 29
viscoe, 49, 119
vL, 34
voltmeter, 44
voosource, 43
vphaseshifter, 55
vpiattenuator, 55
vR, 31
vsourceC, *see* cute european voltage source
vsourceN, *see* noise voltage source
vsourcesin, *see* sinusoidal voltage source
vsourcesquare, *see* square voltage source
vsourcetri, 43
vtattenuator, 55

waves, 69
wilkinson, 54

xing, *see* crossing

zD*, *see* full Zener diode
zD-, *see* stroke Zener diode
zDo, *see* empty Zener diode
zzD*, *see* full ZZener diode
zzD-, *see* stroke ZZener diode
zzDo, *see* empty ZZener diode