



CircuiTikZ
version 1.6.3-unreleased-91346a3 (2023/06/02)

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)

June 2, 2023

Contents

1	Introduction	8
1.1	About	8
1.2	License	8
1.3	Loading the package	8
1.4	Installing a new version of the package.	9
1.5	Requirements	9
1.6	Incompatible packages	9
1.7	Known bugs and limitation	10
1.8	Scale factor inaccuracies	10
1.9	Incompabilities between versions	11
1.10	Feedback	13
1.11	Package options	14
2	Tutorials	17
2.1	Getting started with CircuiTikZ: a current shunt	17
2.2	A non-inverting op-amp amplifier	20
2.2.1	Reusing the circuit: the easy way	22
2.3	A transistor-based amplifier	23
2.4	A logic circuit	28
3	The components: usage	31
3.1	Path-style components	31
3.1.1	Anchors	31
3.1.2	Border anchors	32
3.1.3	Relative coordinates	32
3.1.4	Customization	33
3.1.4.1	Components size	33
3.1.4.2	Mirroring and flipping path-style components	35
3.1.4.3	Thickness of the lines	35
3.1.4.4	Shape of the components	35
3.1.5	Descriptions	36
3.2	Node-style components	36
3.2.1	Mirroring and flipping	37
3.2.2	Anchors	37
3.2.3	Descriptions	38
3.3	Styling circuits and components	38
3.3.1	Relative size	39
3.3.2	Fill color	41
3.3.3	Line thickness	42

3.3.4	Style files	42
3.3.5	Style files: how to write them	43
3.4	Subcircuits	44
3.4.1	Subcircuit definition	44
3.4.2	Using the subcircuit	46
3.4.2.1	Scaling, flipping and rotating subcircuits	46
3.4.3	Parameters in subcircuits	46
4	The components: list	48
4.1	Grounds and supply voltages	48
4.1.1	Grounds	48
4.1.1.1	Grounds anchors	48
4.1.1.2	Grounds customization	49
4.1.2	Power supplies	49
4.1.2.1	Power supply anchors	49
4.1.2.2	Power supplies customization	49
4.2	Resistive bipoles	50
4.2.1	Potentiometers: wiper position	52
4.2.2	Generic sensors anchors	53
4.2.3	Resistive components customization	53
4.2.3.1	Geometry.	53
4.2.3.2	Thickness.	54
4.2.3.3	Arrows.	54
4.3	Capacitors and inductors: dynamical bipoles	54
4.3.1	Capacitors	54
4.3.2	Capacitive sensors anchors	55
4.3.3	Capacitors customizations	56
4.3.4	Inductors	56
4.3.5	Inductors customizations	57
4.3.5.1	Chokes	58
4.3.6	Inductors anchors	58
4.3.6.1	Taps.	58
4.3.6.2	Core anchors.	58
4.3.6.3	Dot anchors.	59
4.4	Diodes and such	59
4.4.1	Tripole-like diodes	62
4.4.2	Thyristors anchors and customization	65
4.4.3	Diode customizations	66
4.4.3.1	Optical devices arrows	66
4.5	Sources and generators	67
4.5.1	Batteries	67

4.5.2	Stationary sources	68
4.5.3	Sinusoidal sources	68
4.5.4	Controlled sources	69
4.5.5	Noise sources	70
4.5.6	Special sources	71
4.5.7	Nullator and norator	72
4.5.8	DC sources	73
4.5.9	Sources customizations	73
4.5.9.1	Size.	73
4.5.9.2	Waveform symbols.	74
4.5.9.3	Polarity symbols.	74
4.5.9.4	Three-phase symbols.	74
4.5.10	Source borders	75
4.6	Instruments	75
4.6.1	Instruments customizations	76
4.6.1.1	Oscilloscope waveform.	77
4.6.2	Rotation-invariant elements	77
4.6.3	Instruments as node elements	78
4.6.4	Measuring voltage and currents, multiple ways	79
4.7	Mechanical Analogy	81
4.7.1	Mechanical elements customizations	82
4.8	Miscellaneous bipoles	82
4.8.1	Miscellaneous element customization	84
4.9	Multiple wires (buses)	84
4.10	Crossings	85
4.10.1	Crossing customization	86
4.11	Arrows	86
4.11.1	Arrows size	87
4.11.2	Generic Tunable Arrows	87
4.12	Terminal shapes	88
4.13	Connectors	88
4.13.1	BNC connector/terminal	89
4.13.2	IEC 60617 socket-plug connectors	89
4.14	Block diagram components	91
4.14.1	Blocks anchors	95
4.14.2	Blocks customization	97
4.14.2.1	Multi ports	98
4.14.2.2	Labels and custom two-port boxes	98
4.14.2.3	Box option	98
4.14.2.4	Dash optional parts	98
4.14.2.5	Dashing the DC symbol in blocks.	99

4.15	Transistors	99
4.15.1	Standard bipolar transistors	99
4.15.2	Multi-terminal bipolar transistors	100
4.15.3	Field-effect transistors	101
4.15.4	Transistor texts (labels)	105
4.15.5	Transistors customization	106
4.15.5.1	Size.	106
4.15.5.2	Arrows.	106
4.15.5.3	Circles.	107
4.15.5.4	Body diodes and similar things.	107
4.15.5.5	HEMT customization.	108
4.15.5.6	Schottky transistors.	109
4.15.5.7	Ferroelectric transistors	109
4.15.5.8	IGBT outer base.	109
4.15.5.9	UJT transistors.	110
4.15.5.10	Base/Gate terminal.	110
4.15.5.11	Bulk terminals.	111
4.15.5.12	Simplified symbols for depletion-mode MOSFETs	112
4.15.5.13	Gate/Base gap coloring.	113
4.15.6	Multiple terminal transistors customization	113
4.15.7	Transistor circle customization	114
4.15.7.1	Position and size.	114
4.15.7.2	Line and color.	114
4.15.7.3	Partially drawn circle borders	115
4.15.8	Transistor bodydiode customization	116
4.15.9	Transistors anchors	117
4.15.10	Transistor paths	120
4.16	Electronic Tubes	121
4.16.1	Tubes customization	123
4.16.2	Tubes anchors	124
4.16.3	Partially drawn tube borders	124
4.16.4	Other tubes-like components	125
4.16.4.1	Dynode customization.	126
4.17	RF components	126
4.17.1	RF elements customization	128
4.17.2	Microstrip customization	128
4.18	Electro-Mechanical Devices	129
4.18.1	Electro-Mechanical Devices anchors	130
4.19	Double bipoles (transformers)	130
4.19.1	Transformer anchors	132
4.19.2	Transformers customization	133

4.19.3	Styling transformer's coils independently	135
4.19.4	Generic double bipoles	136
4.20	Amplifiers	137
4.20.1	Amplifiers anchors	138
4.20.2	Amplifiers customization	140
4.20.2.1	Input polarity.	140
4.20.2.2	Input and output pins symbols.	141
4.20.2.3	Input and output pins length.	141
4.20.2.4	Main amplifier label.	142
4.20.2.5	European-style amplifier customization.	142
4.20.3	Designing your own amplifier	144
4.21	Switches, buttons and jumpers	144
4.21.1	Traditional switches	144
4.21.2	Cute switches	145
4.21.2.1	Cute switches anchors	146
4.21.2.2	Cute switches customization	147
4.21.3	Proximity switches	147
4.21.4	Rotary switches	148
4.21.4.1	Rotary switch anchors	149
4.21.4.2	Rotary switch customization	150
4.21.5	Switch arrows	150
4.21.5.1	Rotary switch arrows.	151
4.21.6	Jumpers	151
4.21.6.1	Simple jumpers.	151
4.21.6.2	Two-ways (three-pins) jumpers.	152
4.21.7	Solder jumpers.	153
4.22	Logic gates	154
4.22.1	American Logic gates	154
4.22.2	IEEE logic gates	155
4.22.3	European Logic gates	156
4.22.4	Path-style logic ports	157
4.22.5	American ports usage	158
4.22.5.1	American logic port customization	158
4.22.5.2	American logic port anchors	160
4.22.6	IEEE logic gates usage.	161
4.22.6.1	Stacking and aligning IEEE standard gates.	163
4.22.6.2	IEEE standard ports customization	164
4.22.6.3	IEEE standard ports anchors	165
4.22.6.4	Transmission gate symbols.	165
4.22.7	European logic port usage	166
4.22.7.1	European logic port customization	166

4.22.7.2	European logic port anchors	166
4.23	Flip-flops	167
4.23.1	Custom flip-flops	169
4.23.2	Flip-flops anchors	169
4.23.3	Flip-flops customization	170
4.24	Multiplexer and de-multiplexer	171
4.24.1	Mux-Demux: design your own shape	173
4.24.2	Mux-Demux customization	174
4.24.3	Mux-Demux anchors	174
4.24.4	Adding wedge or circular inversion markers	175
4.24.5	Mux-Demux special usage	176
4.25	Chips (integrated circuits)	177
4.25.1	DIP and QFP chips customization	177
4.25.2	Chips anchors	179
4.25.3	Chips rotation	179
4.25.4	Chip special usage	179
4.26	Seven segment displays	180
4.26.1	Seven segments anchors	181
4.26.2	Seven segments customization	181
5	Labels, voltages and currents	182
5.1	Labels and Annotations	182
5.1.1	Label and annotation position	183
5.1.1.1	Adjust label and annotation position.	183
5.1.2	Special symbols in labels and annotations.	183
5.1.3	Labels and annotation orientation.	184
5.1.4	Stacked (two lines) labels.	185
5.2	Currents and voltages	186
5.2.1	Common properties of voltages and currents	189
5.2.2	Special treatment for generators	191
5.3	Currents	193
5.4	Flows	194
5.5	Voltages	195
5.5.1	European style	195
5.5.2	Straight European style	197
5.5.3	American style	198
5.5.4	Raised American style	198
5.5.5	Voltage position	199
5.5.6	American voltages customization	200
5.5.7	Combining different styles	201
5.6	Changing the style of labels, voltages, and other text ornaments	201

5.7	Accessing labels text nodes	202
5.8	Advanced voltages, currents and flows	203
5.8.1	Activating the anchors	204
5.8.2	Auxiliary information	206
5.8.3	Fixed voltage arrows: an example of advanced voltage usage	206
5.9	Integration with <code>siunitx</code>	208
6	Using bipoles in circuits	209
6.1	Nodes (also called poles)	209
6.1.1	Transparent poles	211
6.2	Mirroring and Inverting	211
6.3	Putting them together	212
6.4	Line joins between Path Components	212
7	Colors	214
7.1	Shape colors	215
7.2	Fill colors	217
7.2.1	Background colors different from white	218
8	FAQ: Frequently asked questions	220
8.1	Using named nodes in circuits	220
8.2	Using dashed (or colored) wires in circuits	221
8.3	Errors when externalizing pictures	222
8.4	Labels, voltages and currents woes	222
8.5	Global scaling and rotating	223
8.6	Tunable components	223
9	Defining new components	224
9.1	Suggested setup	224
9.2	Path-style component	225
9.3	Node-style component	228
9.3.1	The internal hook system	229
9.3.2	Finishing your work	229
10	Examples	230
10.1	A red diode	230
10.2	Using the (experimental) <code>siunitx</code> syntax	231
10.3	Photodiodes	232
10.4	A Sallen-Key cell	232
10.5	Mixing circuits and graphs	233
10.6	RF circuit	234
10.7	A styled low noise input stage	235
10.8	An example with the <code>compatibility</code> option	236
10.9	3-phases block schematic	237
11	Changelog and Release Notes	238
	Index of the components	253

1 Introduction

*Lorenzo and Mirella, 57 years ago, started a trip
that eventually lead to a lot of things — among
them, CircuiTikZ v1.0.
In loving memory — R.G., 2020-02-04*

1.1 About

CircuiTikZ 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 CircuiTikZ slowed down, since L^AT_EX 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 CircuiTikZ 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 © 2007–2023 by Massimo Redaelli, 2013–2023 by Stefan Erhardt, 2015–2023 by Stefan Lindner, and 2018–2023 by Romano Giannetti. This package is author-maintained. Permission is granted to copy, distribute and/or modify this software under the terms of the L^AT_EX 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

L ^A T _E X	ConT _E Xt ¹
<code>\usepackage{circuitikz}</code>	<code>\usemodule[circuitikz]</code>

TikZ will be automatically loaded; additionally, the TikZ libraries `calc`, `arrows.meta`, `bending`, and `fpu` are loaded (the last one is used only on demand).

CircuiTikZ commands are just TikZ commands, so a minimum usage example would be:



There is really no support for Plain TeX — the maintainers are willing to consider patches if somebody is interested.

¹ConT_EXt support was added mostly thanks to Mojca Miklavc and Aditya Mahajan. **Please notice** that since ConT_EXt switched to the new `lmtx` engine (March 2023), there can be problems to compile TikZ which some version; in May 2023 the problem has been fixed. Please check [this issue](#) for details.

1.4 Installing a new version of the package.

The stable version of the package should come with your L^AT_EX distribution. Downloading the files from CTAN and installing them locally is, unfortunately, a distribution-dependent task and sometimes 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 CircuiTikZ 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 `circuitikzgit.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 sometimes, things happen.

1.5 Requirements

- `tikz`, version $\geq 3.1.5b$ (it *should* work with any version from 3.0 and up, but better use a newer one);
- `xstring`, not older than 2009/03/13;
- `siunitx`, if using `siunitx` option (better v2 or newer).

A similar approach for ConT_EXt is available, with the file `t-circuitikzgit.tex`; the compatibility in this case is not guaranteed, but provided on a *best effort* base.

This manual has been typeset with CircuiTikZ 1.6.3-unreleased-91346a3 (2023/06/02) on TikZ 3.1.5b (2020/01/08).

1.6 Incompatible packages

TikZ’s own `circuit` library, which was based on CircuiTikZ, (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 CircuiTikZ `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 236.

Anyway, the compatibility code is a *best effort* task and only very lightly tested — the authors advice is to choose one or the other, without mixing them.

Another thing to take into account is that any TikZ figure (and CircuiTikZ ones qualify) **will** have problems if you use the `babel` package with a language that changes active characters (most of them). The solution is normally to add the line `\usetikzlibrary{babel}` in your preamble, after loading TikZ or CircuiTikZ. This will normally solve the problem; some languages also require using `\deactivatequoting` or the option `shorthands=off` for `babel`. Please check the documentation of TikZ or this question [on T_EX stackexchange site](https://tex.stackexchange.com/questions/111111).

Finally, the TikZ library `bending` is loaded by the package, and its effects (the bending of the arrows on curved paths) will also affect the rest of your drawings.

1.7 Known bugs and limitation

CircuitikZ 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 will it work with angle-changing scaling (when `xscale` is different from `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 creates a lot of problems with compatibility between CircuitikZ and the new `pic TikZ` feature, so basically don't put components into `pics`.

Also, notice that several components will interact in a funny way with global path options. Depending on the specific component, some parameters are inherited by the internal shape, and some others are reset. This is not easy to fix in general. We want some options to go through — fill color, dashed pattern for example — and some others to stay only in the outer path; and if the background shape needs some option for drawing the internal shape, like for example a rounded corner, it *must* reset the external option. So there is no perfect solution, although since v1.5.0 the shapes have been “robustified”, so that by default arced corners and arrows parameters will *not* be propagated into the shape. Arrows with `to[]` components don't work, anyway, so basically avoid this situation.



```

1 \begin{circuitikz}[]
2   \draw (0,3) to[R] ++(3,0) node[npn, anchor=B]{};
3   % arrows will not work and give strange results
4   % so basically do not use them!
5   \draw[<->] (0,1.5) to[R] ++(3,0) node[npn, anchor=B]{};
6   \draw[shorten <=10pt] (0,0) to[R] ++(3,0)
7     node[npn, anchor=B]{};
8 \end{circuitikz}

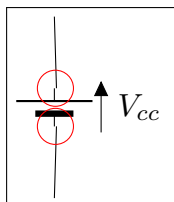
```

Lastly, voltage styles interacts in strange ways with general (such as `american`, `european` style), in the sense that sometimes the order in which you enact them is important. That should be arguably fixed, but it will change (read break) a lot of existing code, so it'll stay; more information and workarounds in section 5.5.7.

As a final notice, if you want to use the `externalize` library, do not use the `circuitikz` environment: use `tikzpicture` (which is really the same thing, see the FAQ 8.3).

1.8 Scale factor inaccuracies

Sometimes, when using fractional scaling factors and big values for the coordinates, the basic layer inaccuracies from \TeX can bite you, producing results like the following one:



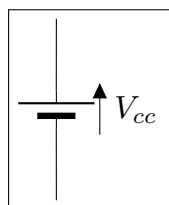
```

1 \begin{circuitikz}[scale=1.2, transform shape,
2   ]
3   \draw (60,1) to [battery2, v_=$V_{cc}$, name=B] ++(0,2);
4   \node[draw,red,circle,inner sep=4pt] at (B.left) {};
5   \node[draw,red,circle,inner sep=4pt] at (B.right) {};
6 \end{circuitikz}

```

A general solution for this problem is difficult to find; probably the best approach is to use a `scalebox` command to scale the circuit instead of relying on internal scaling.

Nevertheless, [Schrödinger's cat](#) found a solution which has been ported to CircuiTikZ: you can use the key `use fpu reciprocal` which will patch a standard low-level math routine with a more precise one.



```
1 \begin{circuitikz}[scale=1.2, transform shape,
2   use fpu reciprocal,
3 ]
4 \draw (60,1) to [battery2, v_=$V_{cc}$] ++(0,2);
5 \end{circuitikz}
```

The `use fpu reciprocal` key seems to have no side effects, but given that it is patching an internal interface of TikZ it can break any time, so it is advisable to use it only if and when needed.

1.9 Incompatibilities between versions

Here, we will provide a list of incompatibilities between different versions of CircuiTikZ. We will try to hold this list short, but sometimes it is easier to break with old syntax than include a lot of switches and compatibility layers. In general, changes that would invalidate a circuit (changes of polarity of components and so on) are almost always protected by a flag; the same is not true for purely aesthetic changes. If unsure, you can check the version in your local installation by using the macro `\pgfcircversion{}`.

- Since version 1.6.2 `siunitx` will **not** work anymore with ConTeXt (it was a very poor simulation layer, anyway); it has been disabled in upstream ConTeXt, in favor of [its own units module](#).
- Version 1.6.0 has a big rewrite of the block's code. In principle the changes are backward-compatible, but there were several bugs (wrong anchors, errors with rotations, and so on) that have been fixed in the process.
- Since v1.5.1² color management (see section 7) and the details of how the shapes are drawn and protected by the external drawing options has changed. There should be no substantial changes to the circuits, though.
- The TikZ fix for `to[...] +(x,y)` behavior (see 3.1.3) uncovered a bug in the positioning of the labels in CircuiTikZ that had been present since v0.8. So you **must** upgrade to v1.4.1 or better if you have TikZ newer than 3.1.8 (and you want/need to use the `+(x,y)` syntax).
- There have been changes in (internal) parameters for capacitors in v1.4.1; now to change them you should use the style interface (see 4.3.3).
- CircuiTikZ v1.4.0 introduced the rollback system for the package when using LaTeX; that (at least in principle) should be completely backward-compatible.
- The path construction in v1.4.0 has been changed a bit (again). The change shouldn't break any circuit and will correct a behavior that should have been fixed with the v1.2.1 change (see below).

²Do not use v1.5.0, it's buggy.

- Version 1.3.6 fixes several problems with the stacked labels; the most important change is that now the bracing of arguments is respected as in version 1.3.0 for the other labels. The special treatment in stacked labels (and only in stacked labels!) for the (still experimental³) `siunitx` compact syntax `<...>` has been removed: it was completely buggy before, and silently ignored, now will throw an error.
- Version 1.3.3 fixes the direction of the arrows in tunable elements; before this version, they were more or less random, now the arrow goes from bottom left to top right. You have the option to go back to the old behavior with `\ctikzset{bipoles/fix tunable direction=false}`. As a compensation for the fuss, now the arrows are configurable. To learn more, see the FAQ: [8.6](#).
- Version 1.3.1 removes the warning if you do not specify a voltage direction.
- Version 1.3.0 fixes the buggy stripping of braces from labels and annotations (see [5.1.2](#)).
- After 1.2.7 a big code reorganization (which had the collateral effect of fixing some bugs) has been made; no changes should be visible, but a fallback point at 1.2.7 has been added.
- You **must** upgrade to v1.2.7 or newer if you use a TikZ 3.1.8 or 3.1.8a (but better upgrade both packages to the current version). You can check the TikZ version installed using the macro `\pgfversion`.
- After v1.2.1: **Important:** the routine that implements the `to[...]` component positioning has been rewritten. That should enhance the line joins in paths, and it's safer, but it can potentially change some old behavior.

One of the changes is that the previous routine did the wrong thing if you used `(node)` `to[...]` (you should use an anchor or a coordinate, not a node there — like `(node.anchor)` `to[...]`).

The other one was that in the structure `... to[...] node[pos=something] (coord)` the value of `pos` was completely wrong (even if you don't use `pos` explicitly, remember it's `pos=0.5` by default).

Additionally, the old code disrupted the TikZ path-fill mechanism, so that you could get away with using the `fill` option on paths and having just the components filled, not the path. That was incorrect, although sometime it was handy (sorry).

See the FAQ at section [8.1](#) for more information.

- After v1.2.0: voltage arrows, symbols and label positions are calculated with a rewritten routine. There should be little change, *unless* you touched internal values...
- After v1.1.3: from version 1.1.0 to version 1.1.2, the inverted Schmitt buffer in IEEE style ports was called `inv schmitt` (with an additional space). The correct name is `invschmitt port` (the same as the legacy american port).
- After v1.1.2: the position of `american` voltages for the `open` bipoles changed (you can revert to the old behavior, see section [5.5.5](#)).
- After v0.9.7: the position of the text of transistor nodes has changed; see section [4.15.4](#).
- 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 the similarly named parameters in `nor`, `xor`, and `xnor` ports.

³and, really, not advised...

- After v0.9.0: voltage and current directions/signs (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 5.2). The default case is still the same as v0.8.3, to avoid potentially wrong circuits, but you would be better off with one of the new voltage directions (**EFvoltages** or **RPvoltages**) for newer circuits.
- 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 behavior at mirrored bipoles has changed, this fixes the voltage drawing, but perhaps you will 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, there exists an option to use the old behavior.

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 **L^AT_EX**, the distribution has embedded several important old versions: 0.4, 0.6, 0.7, 0.8.3, 0.9.3, 0.9.6, 1.0, 1.1.2, 1.2.7, and 1.4.6. To switch to use them, since v1.4.0 you simply use the [new LaTeX kernel rollback system](#), changing your **\usepackage** invocation to something like:

```
\usepackage[]{circuitikz}[v0.8.3] % or v0.4, v0.6, ...
```

You can also specify a date instead of a version number: if you write

```
\usepackage[]{circuitikz}[=2020/02/05]
```

the rollback system will load the version that was current on February 5th, 2020 (in this case it will be v1.0 which was released the day before).

If for whatever reasons your kernel is older, you can still use the old method of loading the *package-version* package; for example:

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

which is an inferior solution because it can fool any package you use that depend on **circuitikz**.

Either way, you have to take care of the options that may have changed between versions (and sometime styles, if you use them).

- if you are using ConT_EXt, only versions 0.8.3, 0.9.3, 0.9.6, 1.0, 1.1.2, 1.2.7, and 1.4.6 are packaged; you can use it with

```
\usemodule{circuitikz-0.8.3}
```

1.10 Feedback

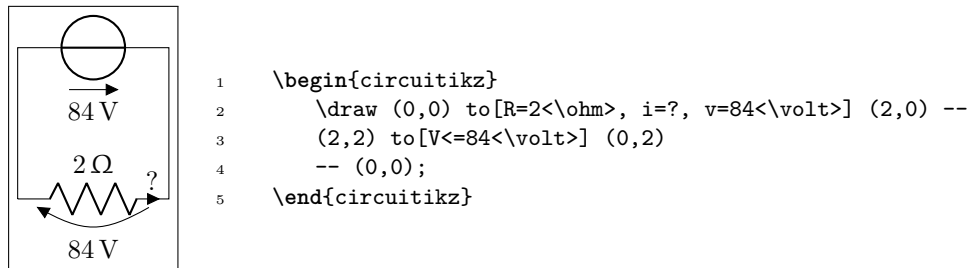
The easiest way to contact the authors is via the official GitHub repository: <https://github.com/circuitikz/circuitikz/issues>. For general help question, a lot of nice people are quite active on <https://tex.stackexchange.com/questions/tagged/circuitikz> — be sure to read the help pages for the site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

1.11 Package options

Circuit people are very opinionated about their symbols. In order to satisfy the individual taste you can set a bunch of package options.

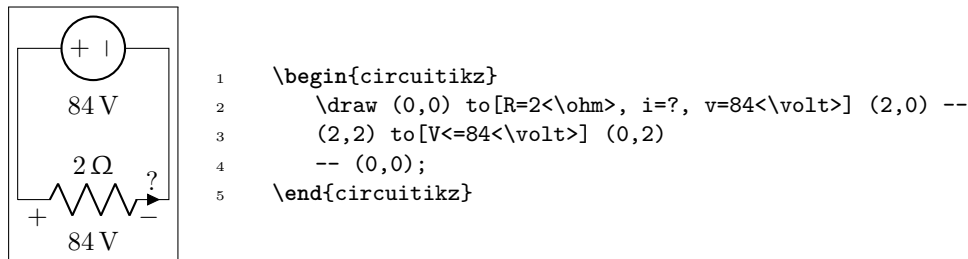
There are arguably way too many options in `CircuitikZ`, as you can see in the following list. Since version 1.0, it is recommended to just use the basic ones — voltage directions (you **should** specify one of them), `siunitx` (only for \LaTeX), the global style (`american` or `european`) and use styles (see 3.3) for the remaining options.

The standard options are set by historical reason, and reflect the preferences of the author that introduced them. For example you get this:



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

\LaTeX	Con \TeX t
<code>\usepackage[american]{circuitikz}</code>	<code>\usemodule[circuitikz][american]</code>



However, most of the global package options are not available in Con \TeX t; in that case you can always use the appropriate `\tikzset{}` or `\ctikzset{}` command after loading the package.

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 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;
- `europoanports`: uses rectangular logic ports, as per european standards;
- `americangfsurgearrester`: uses round gas filled surge arresters, as per american standards;
- `europoangfsurgearrester`: 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}` (not supported in `ConTeXt`; it has been disabled in upstream `ConTeXt`, in favor of [its own units module](#));
- `nosunitx`: 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 does not get filled;
- `lazymos`: draws lazy nmos and pmos transistors. Chip designers with huge circuits prefer this notation;
- `legacytransistorstext`: the text of transistor nodes is typeset near the collector;
- `nolegacytransistorstext` or `centertransistorstext`: the text of transistor nodes is typeset near the center of the component;
- `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 possible to load `CircuitikZ` and `TikZ` circuit library together.
- **Voltage directions**: until v0.8.3, there was an error in the coherence between american and european voltages styles (see section 5.2) 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 labeling for batteries;
 - **nooldvoltagedirection**: The standard from 0.5 onward, utilizes the (German?) standard of voltage arrows in the direction of electric fields (without fixing batteries);
 - **RPvoltages** (meaning Rising Potential voltages): the arrow is in the 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 the direction of the electric field, like in **nooldvoltagedirection**, but batteries are fixed;

If none of these options are given, the package will default to **nooldvoltagedirection**. The behavior is also selectable circuit by circuit with the `voltage dir` style.

- **betterproportions**⁴: nicer proportions of transistors in comparison to resistors; notice that this option is superseded by `styles` and it's kept just for compatibility, do not use it in new projects;

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, centertransistorstext]`.

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

⁴May change in the future!

2 Tutorials

Before even starting with CircuiTikZ you should be sure to have understood the basics of TikZ. It is *highly recommended* that you read and go through *at least* the following parts of the TikZ manual:

- “Tutorial: A Picture for Karl’s Students” (around page 30);
- “Specifying Coordinates” (around page 131)
- “Nodes and their shapes” (around page 220)

...but obviously a good knowledge of TikZ will help you a lot. Remember, a circuit drawn with CircuiTikZ is nothing more than a `tikzpicture` with an (albeit powerful and extended) set of shapes and commodity macros.

Said that, to draw a circuit, you have to load the CircuiTikZ package; this can be done with

```
1 \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 CircuiTikZ: 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 CircuiTikZ is 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 ones 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:



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

The symbol for the current source might surprise some; this is actually the european-style symbol, and the type of symbol chosen reflects the default options of the package (see section 1.11). 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.



```
1 \begin{circuitikz}[american]
2   \draw (0,0) to[isource, l=$I_0$] (0,3) -- (2,3)
3     to[R=$R_1$] (2,0) -- (0,0);
4   \draw (2,3) -- (4,3) to[R=$R_2$]
5     (4,0) -- (2,0);
6 \end{circuitikz}
```

You can use a single path or multiple paths when drawing your circuit, it's just a question of style (but be aware that closing paths perfectly could be non-trivial, see section 6.4), and you can use standard TikZ lines (–, |– or similar) for the wires. Nonetheless, sometime using the CircuitikZ specific `short` component for the wires can be useful, because then we can add labels and poles to them, as for example in the following circuit, where we add a current (with the key `i=...`, see section 5.3) and a connection dot (with the special shortcut `-*` which adds a `circ` node at the end of the connection, see sections 4.12 and 6.1).



```
1 \begin{circuitikz}[american]
2   \draw (0,0) to[isource, l=I_0$] (0,3)
3     to[short, -*, i=I_0$] (2,3)
4     to[R=$R_1$, i=i_1$] (2,0) -- (0,0);
5   \draw (2,3) -- (4,3)
6     to[R=$R_2$, i=i_2$]
7     (4,0) to[short, -*] (2,0);
8 \end{circuitikz}
```

One of the problems with this circuit is that we would like to have the current labels in a different position, such as for example on the upper side of the resistors, so that Kirchhoff's Current Law at the node is better shown to students. No problem; as you can see in section 5.2 you can use the position specifiers `<>_` after the key `i`:



```
1 \begin{circuitikz}[american]
2   \draw (0,0) to[isource, l=I_0$] (0,3)
3     to[short, -*, i=I_0$] (2,3)
4     to[R=$R_1$, i>_=$i_1$] (2,0) -- (0,0);
5   \draw (2,3) -- (4,3)
6     to[R=$R_2$, i>_=$i_2$]
7     (4,0) to[short, -*] (2,0);
8 \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...



```
1 \begin{circuitikz}[american, voltage shift=0.5]
2   \draw (0,0)
3     to[isource, l=I_0$, v=V_0$] (0,3)
4     to[short, -*, i=I_0$] (2,3)
5     to[R=$R_1$, i>_=$i_1$] (2,0) -- (0,0);
6   \draw (2,3) -- (4,3)
7     to[R=$R_2$, i>_=$i_2$]
8     (4,0) to[short, -*] (2,0);
9 \end{circuitikz}
```

Et voilà! Remember that this is still L^AT_EX, 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 forces European style and flows instead of currents, you just change a couple of things and you have what seems a completely different diagram:



```

1 \begin{circuitikz}[european, voltage shift=0.5]
2   \draw (0,0)
3     to[isourceC, l=$I_0$, v=$V_0$] (0,3)
4     to[short, -*, f=$I_0$] (2,3)
5     to[R=$R_1$, f>_=$i_1$] (2,0) -- (0,0);
6   \draw (2,3) -- (4,3)
7     to[R=$R_2$, f>_=$i_2$]
8     (4,0) to[short, -*] (2,0);
9 \end{circuitikz}

```

And finally, this is still TikZ, so that you can freely mix other graphics element to the circuit.



```

1 \begin{circuitikz}[american, voltage shift=0.5]
2   \draw (0,0)
3     to[source, l=$I_0$, v=$V_0$] (0,3)
4     to[short, -*, f=$I_0$] (2,3)
5     to[R=$R_1$, f>_=$i_1$] (2,0) -- (0,0);
6   \draw (2,3) -- (4,3)
7     to[R=$R_2$, f>_=$i_2$]
8     (4,0) to[short, -*] (2,0);
9   \draw[red, thick] (0.6,2.1) rectangle (4.2,3.8)
10
11   node[pos=0.5, above]{KCL};
12 \end{circuitikz}

```

2.2 A non-inverting op-amp amplifier

Let's now try to draw a non-inverting amplifier based on op-amps; the canonical implementation can be, for example, [this one from “electronics tutorials”](#). Obviously, the style and form of drawing a circuit is often a matter of personal tastes and, maybe even more important, of the details you are focusing on; drawing a non-inverting amplifier will be different if you are drawing it to *explain how it works* or if you are simply using it in a more complex circuit, assuming its operation well known by the reader. Anyway, the final objective is to have a circuit like the one on the right, drawn so that it is easy to reuse.



We have to start the drawing from a generic point. Given that the idea is to have a reusable block, instead of positioning the op-amp and build around it, we will start from the input “pole”:



```
1 \begin{circuitikz}[]
2   \draw (0,0) node[above]{$v_i$} to[short, o-] ++(1,0)
3   node[op amp, noinv input up, anchor=+] (OA1){\texttt{OA1}}
4   ;
5 \end{circuitikz}
```

In this snippet, notice that the only absolute coordinate is the first one; that will enable us to “copy and paste” the circuit in several places, or create a macro for it. We position a text node above it, and then draw a wire with a pole to a relative (1,0) coordinate: in other words, we *move* 1 unit to the right drawing a short-circuit, which is the same as a wire. The usage of `to[short...]` simplifies the position of the pole, but notice that we could have also written:

```
\draw (0,0) node[above]{$v_i$} node[ocirc]{} -- ++(1,0) ...
```

with the same result.

The second step is to position the op-amp. We can check the manual and see the component's description (section 4.20):



where we notice the type of the component (it is a node-type component, so we have to use `node` to position it) and the available “anchors”: points we can use to position the shape or to connect to. Not all the anchors are *explicitly* printed in the description box; you should read further in the manual and you'll see a “*component anchors*” (4.20.1) section with the relevant information.

Anyway, the op-amp must be connected with the + anchor to our input wire, so we say `anchor=+` in the option lists; this shifts the whole element so that the named anchor will lie at the current position of the path. Moreover, normally the shape has the inverting input on the bottom side, and we want it the other way around, so we use also `noinv input up` in the keys defining the node. We could also have flipped the shape with `yscale=-1`, but in this case we would need to consider the effects on anchors and on the text; see section 3.2.1.

Now we can draw the resistors; let's start with R_1 . We will draw it going down vertically from the `-` anchor — we have named the node `OA` so that will be `OA.-`. We will need to connect the R_2 also, so we do the following:

- draw a wire going down, and mark a point where we want the feedback resistor to connect;
- then draw R_1 and finally
- draw the ground node.



```

1 \begin{circuitikz}[scale=0.8, transform shape]
2   \draw (0,0) node[above]{$v_i$} to[short, o-] ++(1,0)
3     node[op amp, noinv input up, anchor=+] (OA){\texttt{OA1}}
4     (OA.-) -- ++(0,-1) coordinate(FB)
5     to[R=$R_1$] ++(0,-2) node[ground]{}
6   ;
7 \end{circuitikz}

```

We are only missing the feedback resistor now. We will use *orthogonal coordinates*, writing:

```
\draw (FB) to[R=$R_2$] (FB -| OA.out) -- (OA.out)
```

The meaning is the following:

- move the current point to the coordinate named `FB`;
- put a resistor, with label R_2 , from here **to**...
- the coordinates, which are at the intersection of a horizontal line through `FB` and a vertical line through `OA.out`: the `-|` coordinate operation is quite mnemonic;
- then continue drawing to `OA.out`.

You can use a separate `\draw` command or just continue the path you were writing; the choice is just personal preference, but be warned that it can affect the drawing of poles (see section 6.1 about this if you notice strange things).

Finally, we add the output and a couple of nodes:



```

1 \begin{circuitikz}[scale=0.8, transform shape]
2   \draw (0,0) node[above]{$v_i$} to[short, o-] ++(1,0)
3     node[op amp, noinv input up, anchor=+] (OA){\texttt{OA1}}
4     (OA.-) -- ++(0,-1) coordinate(FB)
5     to[R=$R_1$] ++(0,-2) node[ground]{}
6     (FB) to[R=$R_2$, *-] (FB -| OA.out) -- (OA.out)
7     to [short, *-o] ++(1,0) node[above]{$v_o$}
8   ;
9 \end{circuitikz}

```

The last step to obtain the final look is to add a bit of styling. We want the op-amp filled with a light cyan color, and we prefer to have the label aligned with the left side of the device:



```

1 \ctikzset{amplifiers/fill=cyan!20, component text=left}
2 \begin{circuitikz}[scale=0.8, transform shape]
3   \draw (0,0) node[above]{$v_i$} to[short, o-] ++(1,0)
4     node[op amp, noinv input up, anchor=+] (OA1){\texttt{OA1}}
5     (OA1.-) -- ++(0,-1) coordinate(FB)
6     to[R=$R_1$] ++(0,-2) node[ground]{}
7     (FB) to[R=$R_2$, *-] (FB -| OA1.out) -- (OA1.out)
8     to [short, *-o] ++(1,0) node[above]{$v_o$}
9   ;
10 \end{circuitikz}

```

The `\ctikzset` command choosing the style is better placed in the preamble, though. Style should be coherent for all the document body, so avoiding stating it for every circuit is normally the best strategy.

2.2.1 Reusing the circuit: the easy way

The easiest way to reuse the circuit is to put it in a macro. This is a very flexible way of doing it; the only drawback is that the only easy way to position it is using the first coordinate: you will not be able to move the component using “anchors”; that is more complex and will need the use of subcircuits (but you will lose parameters...see section 3.4).

Defining a macro for our amplifier could be as easy as this:

```

1 \newcommand\myNIA[4]{%1: name of this amplifier, %2 start coordinate, %3 R1, %4 R2
2   \draw #2 coordinate(#1-in) to[short] ++(1,0)
3   node[op amp, noinv input up, anchor=+] (#1-OA){\texttt{#1}}
4   (#1-OA.-) -- ++(0,-1) coordinate(#1-FB)
5   to[R=#3] ++(0,-2) node[ground]{}
6   (#1-FB) to[R=#4, *-] (#1-FB -| #1-OA.out) -- (#1-OA.out)
7   to [short, *-] ++(1,0) coordinate(#1-out)
8   ;
9 }

```

We remove the open poles (it's better to draw them at the end to avoid artifacts) and then we make the names of the coordinates and of the nodes unique, by prepending a parameter that we will provide at every invocation. Then we remove the labels (for simplicity here) and add a couple of coordinates that we will be able to use from the outside when building our circuit.

And we can use it like in the following:



```

1 \begin{circuitikz}[scale=0.7, transform shape]
2   \myNIA{OA1}{(0,0)}{$R_1$}{$R_2$}
3   % start drawing from the output of OA1
4   \myNIA{OA2}{(OA1-out)}{$R_3$}{$R_4$}
5   \node [ocirc] at (OA1-in) {};
6   \node [above] at (OA1-in) {$v_i$};
7   \node [ocirc] at (OA2-out) {};
8   \node [above] at (OA2-out) {$v_o$};
9   \draw (OA1-out) -| (OA2-in);
10 \end{circuitikz}

```

2.3 A transistor-based amplifier

The idea is to draw a two-stage amplifier for a lesson, or exercise, on the different qualities of BJT and MOSFET transistors.

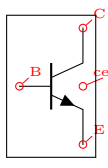
Please notice that this section uses the “new” position for transistor labels, enabled since version 0.9.7. You should refer to older manuals to see how to do the same with older versions; basically the transistor’s names were output using an additional `node{}` command.

Also 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 to use relative coordinates and named nodes as much as possible, so that changes in the circuit are easily done by changing just a few numbers that select relative positions and by using symmetries. Crucially, this kind of approach makes each block reusable in other diagrams by just changing one coordinate.

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

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

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



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

After the circuit is drawn, simply commenting out the second `\let` command 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}`:



```
1 \begin{circuitikz}[american,]
2 \ctikzset{tripoles/mos style/arrows}
3 \def\killdepth#1{{\raisebox{0pt}{\height}{0pt}}{#1}}
4 \path (0,0) -- (2,0); % bounding box
5 \draw (0,0) node[nmos] (Q1){\killdepth{Q1}};
6 \end{circuitikz}
```

I had to do draw an invisible line to take into account the text for Q1 — the text is not taken into account in calculating the bounding box. This is because the “geographical” anchors (`north`, `north west`, ...) are defined for the symbol only. In a complex circuit, this is rarely a problem.

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 of the symbol. Using the following setting (see section 4.15.5) will move the arrows to the start or end of the corresponding pin.

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

The tricky thing about `\killdepth{}` macro is the finicky details. Without the `\killdepth` macro, the labels of different transistors will be adjusted so that the vertical 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):



```

1 \begin{circuitikz}[american,]
2 \draw (0,0) node[nmos] (Q1){q1} ++(2,0)
3   node[nmos] (M1){m1};
4 \draw [red] (Q1.center) ++(0,-0.7ex) -- ++(3,0);
5 \draw (0,-2) node[nmos] (Q1){\killdepth{q1}} ++(2,0)
6   node[nmos] (M1){\killdepth{m1}};
7 \draw [red] (Q1.center) ++(0,-0.7ex) -- ++(3,0);
8 \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).



```

1 \begin{circuitikz}[american,]
2   \draw (0,0) node[nmos,] (Q1){\killdepth{Q1}};
3   \draw (Q1.S) to[R, l2^=$R_S$ and \SI{5}{k\ohm}]
4     ++(0,-3) node[vee] (VEE){$V_{EE}=\SI{-10}{V}$};
5   \draw (Q1.D) to[R, l2^=$R_D$ and \SI{10}{k\ohm}]
6     ++(0,3) node[vcc] (VCC){$V_{CC}=\SI{10}{V}$};
7   \draw (Q1.S) to[short] ++(2,0) to[C=$C_1$]
8     ++(0,-1.5) node[ground] (GND){};
9   % show the named coordinates!
10  \path (GND) \coord(GND)
11    (VCC) \coord(VCC)
12    (VEE) \coord(VEE);
13 \end{circuitikz}

```

After that, let's add the input part. I will use a named node here, referring 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 vertical one of (**GND**), lining it up with the ground of the capacitor C_1 (you can think it as “the point aligned vertically with **in** and horizontally with **GND**”).



```

1 \begin{circuitikz}[american, scale=0.7, transform
  shape]
2 \draw (0,0) node[nmos,](Q1){\killdepth{Q1}};
3 \draw (Q1.S) to[R, l2^=$R_S$ and \SI{5}{k\ohm}]
4   ++(0,-3) node[vee](VEE){$V_{EE}=\SI{-10}{V}$};
5 \draw (Q1.D) to[R, l2_=$R_D$ and \SI{10}{k\ohm}]
6   ++(0,3) node[vcc](VCC){$V_{CC}=\SI{10}{V}$};
7 \draw (Q1.S) to[short] ++(2,0) to[C=$C_1$]
8   ++(0,-1.5) node[ground](GND){};
9 \draw (Q1.G) to[short] ++(-1,0)
10   \coord (in) to[R, l2^=$R_G$ and \SI{1}{M\ohm}]
11   (in |- GND) node[ground]{};
12 \draw (in) to[C, l_=$C_2$,*-o]
13   ++(-1.5,0) node[left](vi1){$v_i=v_{i1}$};
14 \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:

```

1 % this is for the blue brackets under the circuit
2 \tikzset{blockdef/.style={%
3   {Straight Barb[harpoon, reversed, right, length=0.2cm]}--{Straight Barb[harpoon,
4     reversed, left, length=0.2cm]},
5   blue,
6 }}
7 \def\killdepth#1{\raisebox{0pt}{\height}[0pt]{#1}}
8 \def\coord(#1){coordinate(#1)}
9 \def\coord(#1){coordinate(#1) node[circle, red, draw, inner sep=1pt,
10   pin={red, overlay, inner sep=0.5pt, font=\tiny, pin distance=0.1cm,
11     % we reset the arrow in pin edge to avoid carrying over the path one!
12     pin edge={red, overlay,-}45:#1}]{#1-node}{}}
13 \begin{circuitikz}[american, ]
14   \draw (0,0) node[nmos,](Q1){\killdepth{Q1}};
15   \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
16   \draw (Q1.S) to[short] ++(2,0) to[C=$C_1$] ++(0,-1.5) node[ground](GND){};
17   \draw (Q1.G) to[short] ++(-1,0) \coord (in) to[R, l2^=$R_G$ and \SI{1}{M\ohm}] (in |-
18     GND) node[ground]{};
19   \draw (in) to[C, l_=$C_2$,*-o] ++(-1.5,0) node[left](vi1){$v_i=v_{i1}$};
20   \draw (Q1.D) to[R, l2_=$R_D$ and \SI{10}{k\ohm}] ++(0,3) node[vcc](VCC){$V_{CC}=\SI{10}{V}$};
21   \draw (Q1.D) to[short, -o] ++(1,0) node[right](vo1){$v_{o1}$};
22   %
23   \path (vo1) -- ++(2,0) \coord(bjt);
24   %
25   \draw (bjt) node[npn, anchor=B](Q2){\killdepth{Q2}};
26   \draw (Q2.B) to[short, -o] ++(-0.5,0) node[left](vi2){$v_{i2}$};
27   \draw (Q2.E) to[R, l2^=$R_E$ and \SI{9.3}{k\ohm}] (Q2.E |- VEE) node[vee]{};
28   \draw (Q2.E) to[short, -o] ++(1,0) node[right](vo2){$v_{o2}$};
29   \draw (Q2.C) to[short] (Q2.C |- VCC) node[vcc]{};
30   %
31   \path (vo2) ++(1.5,0) \coord(load);
32   \draw (load) to[C=$C_3$] ++(1,0) \coord(tmp) to[R=$R_L$] (tmp |- GND) node[ground]{};

```

```

31 \draw [densely dashed] (vo2) -- (load);
32 %
33 \draw [densely dashed] (vo1) -- (vi2);
34 %
35 \draw [blockdef] (vi1|-VEE) ++(0,-2) \coord(tmp)
36 -- node[midway, fill=white]{bloque 1} (vo1|- tmp);
37 \draw [blockdef] (vi2|-VEE) ++(0,-2) \coord(tmp)
38 -- node[midway, fill=white]{bloque 2} (vo2|- tmp);
39
40 \end{circuitikz}

```



You can see that after having found the place where we want to put the BJT transistor (line 21), we use the option `anchor=B` so that the base anchor will be put at the coordinate `bjt`.

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

```

1 \begin{circuitikz}[american, scale=0.8] % this will scale only the coordinates
2 \ctikzset{resistors/scale=0.7, capacitors/scale=0.6}
3 ...
4 \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).



bloque 1

bloque 2

2.4 A logic circuit

Let's suppose we want to reproduce the circuit on the right⁵, maybe as part of a more complex one.

Looking at the circuit to draw, I see that there is a basic block: the flip-flop with the added three-port circuit to its left, marked with the red dashed rectangle. The main distance to respect here is that we want the two ANDs in line with the flip-flop inputs, so I'll start with the flip-flop and then add the rest of the block.

The shapes are very similar to the IEEE logic gates (see section 4.22.2); after a first check, the standard size of the port is a bit too big with respect to the flip-flop, so I scale them down a bit.

```
1 \ctikzset{
2     logic ports=ieee,
3     logic ports/scale=0.7,
4 }
```



I want a reusable block, so I will start from a coordinate and then use only relative, defining coordinates along the way.

The first thing is to define a suitable flip-flop. The standard SR (see 4.23) is *almost* what we need, but not exactly the same. So let's define a new one:

```
1 \tikzset{sr-ff/.style={flipflop, flipflop def={
2     t1=S, t2=CP, t3=R, t4={\ctikztextnot{Q}},
3     t6=Q, nd=1}},
4 }
```



Now we can add the “and” gates. For example, we can add the gates to the right like this:



```
1 \begin{circuitikz}[]
2     \draw (0,0) node[sr-ff](FF){} (FF.bup)
3     node[above]{SR-FF};
4     \draw (FF.pin 1) -- ++(-1,0) node[and port,
5         anchor=out](AND1){}
6         (FF.pin 3) -- ++(-1,0) node[and port,
7         anchor=out](AND2){};
8 \end{circuitikz}
```

You can notice a pair of things here: first of all, the use of the `anchor=out` in the port, to tell TikZ that we want the node moved so that the out anchor is the reference one. The second one is that we have repeated the absolute shift (the `++(-1, 0)`) twice. This is a bad practice; it is much better to have the “free” parameters of a schematic just stated once, so that we can change them in just one point.

You can of course use a macro, like `\newcommand{\andshift}{(-1,0)}` but it is much more elegant to do something like this:

⁵It seems a quite popular one on tex.stackexchange.com...



```

1 \begin{circuitikz}[]
2   \draw (0,0) node[sr-ff](FF){} (FF.bup)
3     node[above]{SR-FF};
4   \draw (FF.pin 1) -- ++(-1,0) node[and port,
5     anchor=out](AND1){}
6     (FF.pin 3) -- (FF.pin 3 -| AND1.out)
7     node[and port, anchor=out](AND2){};
8 \end{circuitikz}

```

In this snippet, the coordinate (FF.pin 3 -| AND1.out) is the TikZ way to say “the point which is horizontally straight from FF.pin 3 and vertically from AND1.out”. That way one can change the number -1 to move both AND ports nearer or farther away.

Now we can add the not port. Since version 1.1.3 you can use a path-style not port, so you can just say: this:



```

1 \begin{circuitikz}[scale=0.8, transform shape]
2 \draw (0,0) node[sr-ff](FF){} (FF.bup)
3   node[above]{SR-FF} (FF.pin 1) -- ++(-1,0)
4   node[and port, anchor=out](AND1){}
5   (FF.pin 3) -- (FF.pin 3 -| AND1.out)
6   node[and port, anchor=out](AND2){}
7   (AND1.in 1) to[short, -*] ++(-1,0) coordinate(in)
8   to[inline not] (in |- AND2.in 2) -- (AND2.in 2);
9 \end{circuitikz}

```

In earlier versions, you should have found the center point between the two terminal, position the “not” shape and then connect it, like for example (this code must stay into the \draw command):

```

1 % let's position the NOT in the center
2 % this is using the calc tikz library
3 ($(in)!0.5!(in |- AND2.in 2)$) node[not port, rotate=-90](NOT){}
4 % and connect it
5 (in) -- (NOT.in) (NOT.out) |- (AND2.in 2)

```

Now we have the basic block; we have to use it twice, so one of the possible ways to do it is to prepare a command. We will change the names of the nodes and the coordinates to be different for any “call” of the block (another option is to use a pic; but this is more straightforward).

```

1 \newcommand*\myblock}[1]{% Add #1- to the node and coord names
2   node[sr-ff](#1-FF){} (#1-FF.bup) node[above]{SR-FF}
3   (#1-FF.pin 1) -- ++(-1,0) node[and port, anchor=out](#1-AND1){}
4   (#1-FF.pin 3) -- (#1-FF.pin 3 -| #1-AND1.out)
5   node[and port, anchor=out](#1-AND2){}
6   (#1-AND1.in 1) to[short, -*] ++(-1,0) coordinate(#1-in)
7   to[inline not] (#1-in |- #1-AND2.in 2) -- (#1-AND2.in 2)
8 }

```

So now we can draw two of our blocks:

```
1 \draw (0,0) \myblock{A};
2 \draw (0,-4) \myblock{B};
```

Part of the anchors and coordinates that we have accessible are marked in red in the diagram at the side.

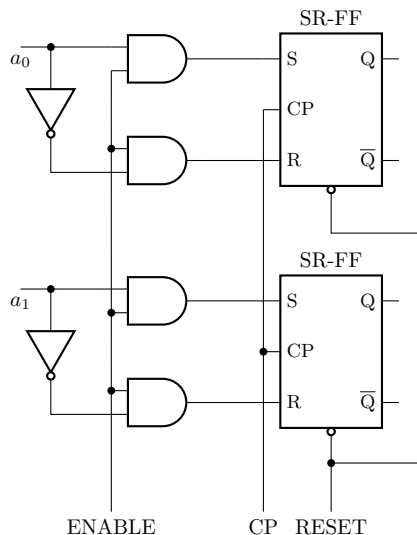
Now we have to just connect the relevant parts and add the labels. The names of the inputs are quite easy:

```
1 \draw (A-in) -- ++(-0.5, 0) node[
    below]{$a_0$};
2 \draw (B-in) -- ++(-0.5, 0) node[
    below]{$a_1$};
```

And finally:

```
1 \draw (A-AND1.in 2) to[short, -*] (A-AND2.in 1)
2   to[short, -*] (B-AND1.in 2) to[short, -*] (B-AND2.in 1)
3   -- ++(0, -2) coordinate(down) node[below]{ENABLE};
4 \draw (A-FF.pin 2) to[short, -*] (B-FF.pin 2)
5   -- (B-FF.pin 2 |- down) node[below]{CP};
6 \draw (B-FF.down) to[short, -*] ++(0,-0.3) coordinate(dd);
7 \draw (A-FF.down) -- ++(0,-.5) -- ++(1.5,0) |- (dd)
8   -- (dd |- down) node[below]{RESET};
```

Will create the final diagram:



3 The components: usage

Components in CircuiTikZ come in two forms: path-style, to be used in a `to[component,... path specifications`, and node-style, which will be instantiated by a `node[component,...] specification`. All the shapes defined by CircuiTikZ are `pgf` nodes, so they are usable in both `pgf` and `TikZ`.

3.1 Path-style components

The path-style components are used as shown below:

```

1 \begin{circuitikz}
2 \draw (0,0) to[#1=#2, options] (2,0);
3 \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 4.15.10.

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-shape 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 using the syntax `name=myname`.



```

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

```

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



```

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

```


3.1.2 Border anchors

Bipoles have also installed generic border anchors — that means, anchors that start at an angle. For complexity reason, these are for most of the components simply a generic enclosing rectangle (even for most of the round ones!⁶). They interact in a non-trivial way with the `mirror` and `invert` keys, so it's best not to use them directly.



You can notice that the border anchors are a bit spaced out (this is useful because those anchors are used to position labels and annotations). You can override this if you need to reach exactly the border (whatever could that mean depends on the component) by using the key `bipoles/border margin`, which is a number that states how much the enclosing border is stretched out (default value is 1.1). For example, setting `\ctikzset{bipoles/border margin=1}` will make the border anchor coincide with the geographical shape:



The above diagram has been obtained with the code:

```

1 \def\showbordersfornode#1{%
2 \begin{circuitikz}[baseline, scale=0.8, transform shape]
3   \node[#1shape, name=bip] at(0,0) {};
4   \foreach \a in {0,30,...,359} \draw[red] (bip.\a) -- ++(\a:0.7)
5     node[font=\tiny, fill=white, inner sep=0.5pt]{\a};
6   \foreach \a in {15,45,...,359} \draw[red] (bip.\a) -- ++(\a:0.4);
7   \node [font=\ttfamily\small, black, below] at (bip.-90)
8     {\detokenize\expandafter{#1}};
9 \end{circuitikz}}
10 \ctikzset{bipoles/border margin=1}
11 \showbordersfornode{generic} \showbordersfornode{resistor}
12 \showbordersfornode{fulldiode} \showbordersfornode{vsource}
13 \showbordersfornode{capacitivesens}

```

3.1.3 Relative coordinates

As noticed by user [septatrix](#), although full relative coordinates after a component work as expected when using `++(x,y)`-style coordinates, often there are problems when using the `+(x,y)`-style coordinates (which are supposed to set a temporary relative coordinate and then going back to the starting point).

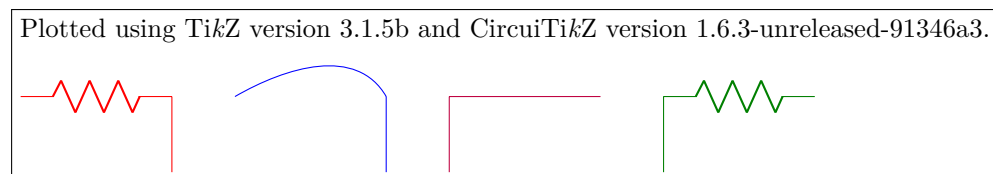
These kind of coordinate have in practice little use for the building of circuits, so have been only (very) lightly tested; avoid them if you can — the behavior will depend not only on the CircuiTiKZ version, but also on the TikZ layer underneath.

⁶This is needed for the correct label/voltage etc. placement, and it's too much work to change it.

This behavior, although not optimal, was standard in to operation in plain TikZ before version 3.1.8; it was changed by Henri Menke in later versions. Notice that the change revealed a problem in CircuiTikZ that should hopefully be fixed in v1.4.1; for more details see [this issue on GitHub](#).

You can see from the example below (notice the blue curve using a spline line). If all the vertical lines are at the left, the manual has been compiled with a new CircuiTikZ and TikZ. Otherwise, the red and/or blue curve will have the vertical line at the right (which in principle is wrong).

In the last (green) example, you can see a workaround using local path and the key `current point is local` that will work for older (and do not create problems in newer) versions.



```

1 Plotted using \TikZ\ version \pgfversion{} and CircuiTikZ\ version \pgfcircversion{}.
2
3 \begin{tikzpicture}
4   \draw[color=red] (0,0) to[R] +(2,0) +(0,0) -- ++(0,-1);
5 \end{tikzpicture}
6 \quad
7 \begin{tikzpicture}
8   \draw[color=blue] (0,0) to[out=30, in=120] +(2,0) +(0,0) -- ++(0,-1);
9 \end{tikzpicture}
10 \quad
11 \begin{tikzpicture}
12   \draw[color=purple] (0,0) to[] +(2,0) +(0,0) -- ++(0,-1);
13 \end{tikzpicture}
14 \quad
15 \begin{tikzpicture}
16   \draw[color=green!50!black] (0,0)
17     {[current point is local] to[R] +(2,0)} +(0,0) -- ++(0,-1);
18 \end{tikzpicture}

```

3.1.4 Customization

Pretty much all CircuiTikZ relies heavily on `pgfkeys` for value handling and configuration. Indeed, at the beginning of `circuitikz.sty` and in the file `pgfcirc.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.

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



```

1 \ctikzset{bipoles/length=1.4cm}
2 \begin{circuitikz}[scale=1.2]\draw
3   (0,0) node[anchor=east] {B}
4     to[short, o-*] (1,0)
5     to[R=20<\ohm>, *-] (1,2)
6     to[R=10<\ohm>, v=$v_x$] (3,2) -- (4,2)
7     to[cI=$\frac{\si{\siemens}}{5}$ v_x$, *-] (4,0) -- (3,0)
8     to[R=5<\ohm>, *-] (3,2)
9   (3,0) -- (1,0)
10  (1,2) to[short, -o] (0,2) node[anchor=east]{A}
11;\end{circuitikz}

```



```

1 \ctikzset{bipoles/length=.8cm}
2 \begin{circuitikz}[scale=1.2]\draw
3   (0,0) node[anchor=east] {B}
4     to[short, o-*] (1,0)
5     to[R=20<\ohm>, *-] (1,2)
6     to[R=10<\ohm>, v=$v_x$] (3,2) -- (4,2)
7     to[cI=$\frac{\si{\siemens}}{5}$ v_x$, *-] (4,0) -- (3,0)
8     to[R=5<\ohm>, *-] (3,2)
9   (3,0) -- (1,0)
10  (1,2) to[short, -o] (0,2) node[anchor=east]{A}
11;\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 the one in the second line below:



```

1 \begin{circuitikz}[
2   bigR/.style={R, bipoles/length=3cm}
3 ]
4 \draw (0,3) to [bigR, o-o] ++(4,0);
5 \draw (0,1.5) to [bigR, o-o] ++(4,0)
6   to[R, o-o] ++(2,0); % will fail here
7 \draw (0,0) to [R, o-o] ++(4,0);
8 \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:



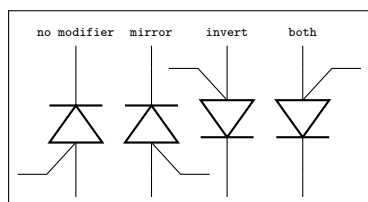
```

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

```

Never use `scale`, `xscale` or `yscale` in a path-style component (i.e., inside a `to[...]` command).

3.1.4.2 Mirroring and flipping path-style components To change the orientation of path-style components, *never* use `xscale=-1` nor `yscale=-1`. That will mess up the path completely. Use the `mirror` and `invert` options:



```

1 \begin{tikzpicture}[N/.style={
2   font=\tiny\ttfamily, above}]
3 \draw (0,0) to [put] ++(0,2)
4   node[N]{no modifier};
5 \draw (1,0) to [put, mirror] ++(0,2)
6   node[N]{mirror};
7 \draw (2,0) to [put, invert] ++(0,2)
8   node[N]{invert};
9 \draw (3,0) to [put, mirror, invert] ++(0,2)
10  node[N]{both};
11 \end{tikzpicture}

```

3.1.4.3 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.



```

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

```

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


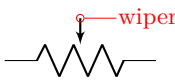
The shape of the components are adjustable with a lot of parameters; in this manual we will comment the main ones, but you can look into the source files specified above to find more.



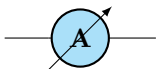
It is recommended to use the styling parameters to change the shapes; they are not so fine-grained (for example, you can change the width of resistor, not the height at the moment), but they are more stable and coherent across your circuit.

3.1.5 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.

Most path-style components can be used as a node-style components; to access them, normally you add a **shape** to the main name of component (for example, **diodeshape**). Sometimes though the “node name” is different, so it is specified in the description of each component.

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:

```

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

Explanation of the parameters:

#1: component name⁷ (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)

⁷For using bipoles as nodes, the name of the node is **#1shape**.

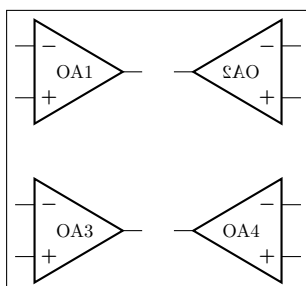
Notice: Nodes must have curly brackets at the end, even when empty. An optional anchor (#3) can be defined within round brackets to be addressed again later on. And please don't forget the semicolon to terminate the `\draw` command.

Also notice: If using the `\tikzexternalize` feature, as of TikZ 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 TikZ keys `xscale` and `yscale`. Notice that these parameters also affect text labels, so they need to be un-scaled by hand. Notice that you **do not** use `xscale` or `yscale` in a path-style component, see section 3.1.4.2 for that case.

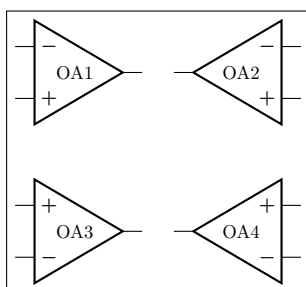


```

1 \begin{circuitikz}[scale=0.7, transform shape]
2   \draw (0,3) node[op amp]{OA1};
3   \draw (3,3) node[op amp, xscale=-1]{OA2};
4   \draw (0,0) node[op amp]{OA3};
5   \draw (3,0) node[op amp, xscale=-1]{%
6     \scalebox{-1}[1]{OA4}};
7 \end{circuitikz}

```

To simplify this task, CircuiTikZ has three helper macros — `\ctikzflipx{}`, `\ctikzflipy{}`, and `\ctikzflipxy{}`, that can be used to “un-rotate” the text of nodes drawn with, respectively, `xscale=-1`, `yscale=-1`, and `scale=-1` (which is equivalent to `xscale=-1`, `yscale=-1`).



```

1 \begin{circuitikz}[scale=0.7, transform shape]
2   \draw (0,3) node[op amp]{OA1};
3   \draw (3,3) node[op amp, xscale=-1]{\ctikzflipx{OA2}};
4   \draw (0,0) node[op amp, yscale=-1]{\ctikzflipy{OA3}};
5   \draw (3,0) node[op amp, scale=-1]{\ctikzflipxy{OA4}};
6 \end{circuitikz}

```

3.2.2 Anchors

Node components anchors vary a lot across the various kinds of components, so they will be described better after each category is presented in the manual. In general all components have geographic anchors (`north`, `north west`, ...), but most of the other anchors are very component-specific.

3.2.3 Descriptions

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

	Cute spdt down with arrow, type: node (<code>node[cute spdt down arrow]{}</code>). Class: switches .
	NPN, TYPE: NODE (<code>node[npn]{}</code>). Class: transistors .

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 (<code>node[plain amp]{}</code>). 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 (<code>node[rotaryswitch] (N){}</code>). 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 circuit⁸.

⁸This is a just an example, the circuit is not intended to be functional.

```

1 \begin{circuitikz}[american, cute inductors]
2   \node [op amp] (A1){\texttt{OA1}};
3   \draw (A1.-) to[short] ++(0,1) coordinate(tmp) to[R, l_=$R$] (tmp -| A1.out) to[short] (A1.out);
4   \draw (tmp) to[short] ++(0,1) coordinate(tmp) to[C=$C$] (tmp -| A1.out) to[short] (A1.out);
5   \draw (A1.+) to [battery2, invert] ++(0,-2.5) node[ground](GND){};
6   \draw (A1.-) to [L=$L$] ++(-2,0) coordinate(tmp) to[sV, l=$v_s$, fill=yellow] (tmp | -GND) node[ground]{};
7   \draw (A1.out) to[R=$R_s$] ++(2,0) coordinate(bb) to[I, l_=$I_B$, invert] ++(0,2) node[vcc](VCC){};
8   \draw (bb) to[D, l=$D$, *-] ++(0,-2) coordinate(bb1) to[R=$R_m$] ++(0,-2) node[vee](VEE){};
9   \draw (bb) ---++(1,0) node[npn, anchor=B](Q1){Q1};
10  \draw (bb1) ---++(1,0) node[pnp, anchor=B](Q2){Q2};
11  \draw (Q1.E) -- (Q2.E) ($ (Q1.E)!0.5!(Q2.E)$) to [short, *-o, name=S] ++(2.5,0)
12  node[right]{$v_{o_Q}$};
13  \draw (S.s) to[european resistor, l=$Z_L$, *-] (S.s|-GND) node[ground]{};
14  \draw (Q1.C) -- (Q1.C|-VCC) node[vcc]{$\SI{5}{V}$};
15  \draw (Q2.C) -- (Q2.C|-VEE) node[vee]{$\SI{-5}{V}$};
16 \end{circuitikz}

```

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



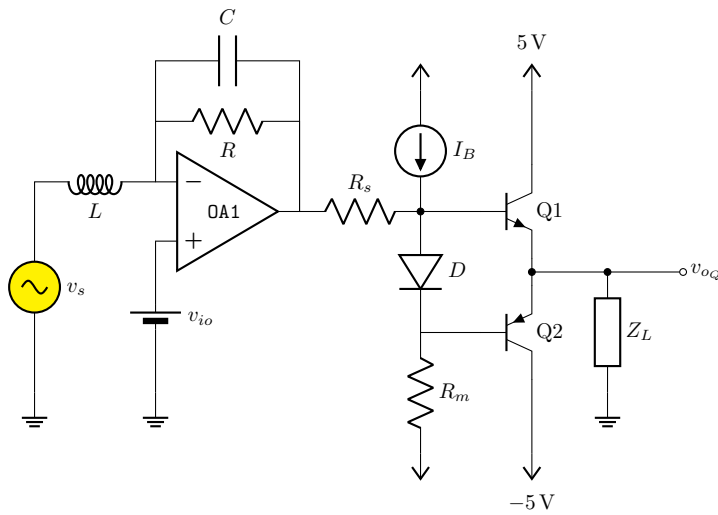
3.3.1 Relative size

Component size can be changed globally (see section 3.1.4.1), 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 3.1.4.1 enables your style for the circuit. For example, setting:

```

1 \ctikzset{resistors/scale=0.8, % smaller R
2   capacitors/scale=0.7,      % even smaller C
3   diodes/scale=0.6,          % small diodes
4   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:

```

1 \ctikzset{
2   amplifiers/fill=cyan,
3   sources/fill=green,
4   diodes/fill=red,
5   resistors/fill=violet,
6 }

```

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-colors 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 3.1.4.3); 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

```
1 \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 TikZ level, not CircuiTikZ — that let’s you use it in the top option of the circuit, like:

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

If you just want to use one style, you can load and activate it in one command with

```
\ctikzsetstyle{romano}
```

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



```
1 \begin{circuitikz}
2   \draw (0,0) node[op amp]{OA1};
3 \end{circuitikz}
4 \ctikzloadstyle{example}
5 \begin{circuitikz}[example circuit style]
6   \draw (0,0) node[op amp]{OA1};
7 \end{circuitikz}
```

The style `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 right 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 TikZ style with name `example circuit style`; basically it has to be something along this:

```
1 % example style for circuits
2 % Do not use LaTeX commands if you want it to be compatible with ConTeXt
3 % Do not add spurious spaces
4 \tikzset{example circuit style/.style={%
5   \circuitikzbasekey/.cd,%
6   amplifiers/fill=blue!20!white,
7 },% end .style
8 }% end \tikzset
9 %
10 \endinput
```

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

```

1 \ctikzloadstyle{legacy}% start from a known state
2 \tikzset{romano circuit style/.style={%
3   legacy circuit style, % load the legacy style
4   \circuitikzbasekey/.cd,%
5   % Resistors
6   resistors/scale=0.8,
7 [...]
8 }}

```

If you want to add a setting to your style file that has been recently added to the package (for example, the thyristor compact shapes added in 1.3.5), but you want your style file to be still compatible with older versions of CircuiTikZ, you can use the `.try` statement:

```

1 % Diodes
2 diodes/scale=0.6,
3 diodes/thickness=1.0,
4 thyristor style/.try=compact,

```

Or, in case of new values of existing “choice” keys, you can use the syntax:

```

1 % Logic ports
2 logic ports/ieee/.try,
3 % this way of setting the key does nothing if ieee option
4 % does not exist; logic ports/.try=ieee does not work
5 % if the key exists but the value is not defined
6 logic ports/scale=1.0,

```

3.4 Subcircuits

Starting from version 1.3.5, there is support for generating sub-circuits, or circuit blocks. The creation and use of subcircuits is somewhat limited, to keep them simple and easy to define and maintain.

A subcircuit is basically a path (and just one path!) of generic TikZ instructions, with a series of accessible coordinates that behave more or less like anchors in the “real” shapes. The basic limitation is that a subcircuit can be moved, replicated and placed around but it can’t be easily personalized. Even if scaling and rotation is in principle possible, it is not easily done. Nevertheless, they can be quite useful to build complex components and reusable blocks.

3.4.1 Subcircuit definition

To define a block you use the `\ctikzsubcircuitdef` macro; this macro has 3 arguments:

- the first argument is the name of the subcircuit, and it must form a valid TeX command name when prepended with a backslash: so just letters (no spaces, nor numbers, nor symbols like underscores, etc.);
- the second one is a comma-separated list of anchor names; here you can use whatever you can use for naming a coordinate or a node (so it’s much more relaxed than the first one);

- finally, the commands that will draw the circuit. You must suppose you are in a `\draw` command, with the start coordinate already set-up. You can (and should) use `#1` as the name of the current node, and you *must* define the coordinates of all the anchors listed before as `coordinate(#1-anchorname)`. You should **not** finish the path here and use **only relative coordinates or named ones**.

Let's see that with an example:

```

1 \ctikzsubcircuitdef{optovishay}{in 1, out 1, in 2, out 2, center}{%
2   % reference anchor is -center
3   coordinate(#1-center)
4   (#1-center) +(-1.2,-1) rectangle +(1.2,1)
5   (#1-center) ++(-1.2,0.8) coordinate (#1-in 1)
6   (#1-center) ++(-1.2,-0.8) coordinate (#1-in 2)
7   (#1-center) ++(1.2,0.8) coordinate (#1-out 1)
8   (#1-center) ++(1.2,-0.8) coordinate (#1-out 2)
9   (#1-center) ++(0,1) coordinate (#1-up)
10  (#1-in 1) -- ++(0.5,0) coordinate(#1-tmp)
11      to[leD*, diodes/scale=0.6, led arrows from cathode]
12      (#1-tmp|- #1-in 2) -- (#1-in 2)
13  (#1-out 1) -- ++(-0.5,0) coordinate(#1-tmp)
14      to[pD*, diodes/scale=0.4, mirror] ++(0,-0.5)
15      edge[densely dashed] ++(0,-0.533) ++(0,-0.566)
16      to[pD*, diodes/scale=0.4,mirror] (#1-tmp|- #1-out 2) -- (#1-out 2)
17  % leave the position of the path at the center
18  (#1-center)
19 }
```

Our element is a symbol for an optocoupler; in this case is the symbol used for one cell of the double [Vishay vo1263 device](#).

The name of the subcircuit is `optovishay` — notice we can use only letters here, upper or lowercase, and nothing more. Then we have a series of anchor names; here we can use letters, numbers, spaces and some symbol — but avoid the dot (.) and the hyphen (-). Additionally, the anchor named `subckt@reference` is reserved and shouldn't be used. If you use spaces, be on the safe side and *never* use two or more consecutive spaces.

After that, you have to draw your subcircuit as if you were into a `\draw` command, starting from a generic point. In this case, we decide to draw the circuit around this generic point so that it will result to be the center of the block; so as a first thing, we “mark” the position of the center anchor, with `coordinate(#1-center)`. The `#1` will be substituted with the specific name of the subcircuit's instance later — so if you then call one instance of the optocoupler `opto1`, that coordinate will be called `opto1-center`.

We continue by defining all our anchors (there is no need to do that at the start, but it's handy because then you can use them). You **must** define all the anchors!

Important: all the coordinates used must be either relative, or named in the form `#1-something`; absolute coordinate will not work when instantiating the block. The block is thought to be used inside a path specification, so the idea is not to end the path — that means that changing line styles or colors is at best difficult. You can still use `edges`, though (see 8.2).

After that, we draw our circuit; in this case a LED and a couple of smaller photodiodes will do. We also define a coordinate `-up` (you can define more coordinates, in addition to the anchors, or name elements with `name=#1-something` for later access) for adding text.

3.4.2 Using the subcircuit

To use the subcircuit, an additional step is needed. Somewhere you have to *activate* it. This is needed to calculate the relative positions of anchors using the current set of style parameters. The normal place is to activate it just before usage; to do that you use the command `\ctikzsubcircuitactivate` with the name of the subcircuit. That will define a new command, `\nameofthesubcircuit` that you can use then in your paths.

So to check your subcircuit while defining it you can use this simple snippet:



3.4.2.1 Scaling, flipping and rotating subcircuits To scale and rotate a subcircuit you have to include it into a *scope* with the appropriate `scale` and `rotation` commands. Notice that, as in general in CircuiTiKZ, global scales that affect rotation works only if `transform shape` is issued (see 1.7); nesting `transform shape` normally works, but it has been really lightly tested.



3.4.3 Parameters in subcircuits

There are no additional parameters definable for subcircuit shapes; this is a bit of a pity, because sometimes they could be useful, especially for labels of objects. Given the need to use `transform shape` to translate and rotate them, though, it is better not to add invariant-direction things (like text) into the subcircuit, unless you are sure you will just translate them. One possibility is to use additional macros and anchors for positioning, like in the following example.

Suppose you have defined

```

1 \ctikzsubcircuitdef{divider}{in, out}{%
2   coordinate (#1-in) to[R, l=~, name=#1-rh, -*] ++(2,0)
3   coordinate(#1-tmp) to[R, l=~, name=#1-rl] ++(0,-2)
4   node[tlground]{} (#1-tmp) ---++(0.5,0) coordinate(#1-out)
5 }

```

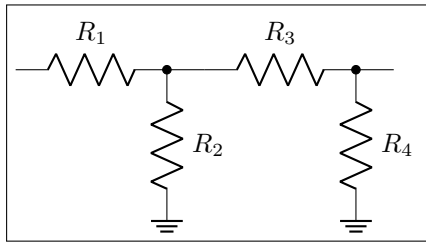
then you can additionally define:

```

1 \newcommand{\mydiv}[4]{
2   \divider{#1}{#2} (#1-rh.n) node[above]{#3}
3   (#1-rl.n) node[right]{#4} (#1-out)
4 }

```

And finally do:



```

1 \ctikzsubcircuitactivate{divider}
2 \begin{tikzpicture}
3   \draw (0,0) \mydiv{a}{in}{R_1}{R_2};
4   \draw (a-out) -- \mydiv{b}{in}{R_3}{R_4};
5 \end{tikzpicture}

```


4 The components: list

This section is dedicated to the full list of available components.

4.1 Grounds and supply voltages

Ground symbols and power supplies — they have two different classes for styling.

4.1.1 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 (<code>node[ground]{}).</code> Class: grounds .
	Tailless ground, type: node (<code>node[tlground]{}).</code> Class: grounds .
	Reference ground, type: node (<code>node[rground]{}).</code> Class: grounds .
	Signal ground, type: node, fillable (<code>node[sground]{}).</code> Class: grounds .
	Thicker tailless reference ground, type: node (<code>node[tground]{}).</code> Class: grounds .
	Noiseless ground, type: node (<code>node[nground]{}).</code> Class: grounds .
	Protective ground, type: node, fillable (<code>node[pground]{}).</code> Class: grounds .
	Chassis ground ⁹ , type: node (<code>node[cground]{}).</code> Class: grounds .
	European style ground, type: node (<code>node[eground]{}).</code> Class: grounds .
	European style ground, version 2 ¹⁰ , type: node (<code>node[eground2]{}).</code> Class: grounds .

4.1.1.1 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”:



⁹These last three were contributed by Luigi «Liverpool»

¹⁰These last two were contributed by @fotesan

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

4.1.2 Power supplies

↑	VCC/VDD, type: node (<code>node[vcc]{}</code>). Class: <code>power supplies</code> .
↓	VEE/VSS, type: node (<code>node[vee]{}</code>). Class: <code>power supplies</code> .

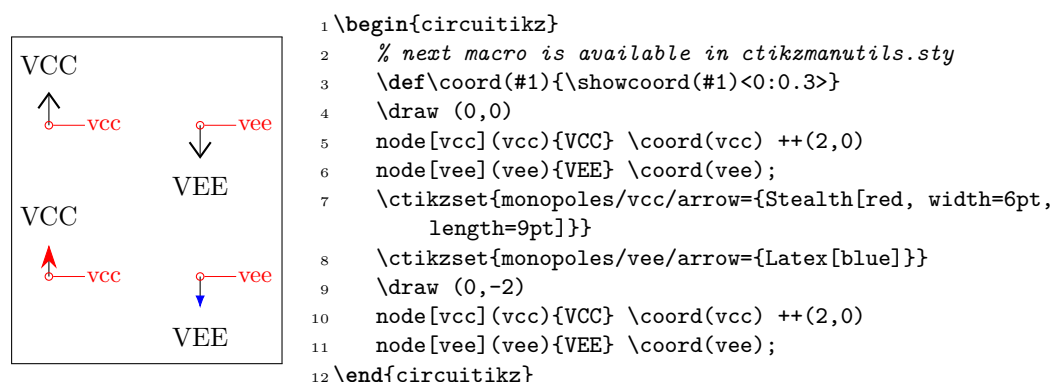
The power supplies are normally drawn with the arrows shown in the list above.

4.1.2.1 Power supply anchors They are similar to ground anchors, and the geographical anchors are correct only for the default arrow.



4.1.2.2 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 TikZ 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!



However, arrows in TikZ 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 the arrows to behave like the legacy symbols (which are shapes), *only in the arrow definitions*, you can use the special length parameter `\scaledwidth`¹¹ in the arrow definition, which correspond to the width of the legacy `vcc` or `vee`. Compare the effects on the following circuit.

¹¹Thanks to @Schrödinger's cat on [TeX stackexchange site](https://tex.stackexchange.com)



```

1 \ctikzset{%
2   monopoles/vcc/arrow={Triangle[width=0.8*\scaledwidth, length=\scaledwidth]},
3   monopoles/vee/arrow={Triangle[width=6pt, length=8pt]},
4 }
5 \begin{circuitikz}[baseline=(vo.center)]
6   \node [ocirc](TW) at (0,0) {};
7   \draw (TW.east) -- ++(1,0) node[midway, above]{$v_i$} node[op amp, anchor=-](A1){};
8   \draw (A1.up) -- ++(0, 0.3) node[vcc]{\SI{+10}{V}};
9   \draw (A1.down) -- ++(0,-0.3) node[vee]{\SI{-10}{V}};
10  \draw (A1.+) -- ++(-0.5,0) to[battery2, invert, l_=\SI{2}{V}] ++(0,-1) node[ground
11    ]{};
12  \draw (A1.out) to[short, -o] ++(0.5,0) node[above](vo){$v_o$};
13 \end{circuitikz} \quad
14 \begin{circuitikz}[baseline=(vo.center), scale=0.6, transform shape]
15   \node [ocirc](TW) at (0,0) {};
16   \draw (TW.east) -- ++(1,0) node[midway, above]{$v_i$} node[op amp, anchor=-](A1){};
17   \draw (A1.up) -- ++(0, 0.3) node[vcc]{\SI{+10}{V}};
18   \draw (A1.down) -- ++(0,-0.3) node[vee]{\SI{-10}{V}};
19   \draw (A1.+) -- ++(-0.5,0) to[battery2, invert, l_=\SI{2}{V}] ++(0,-1) node[ground
20     ]{};
21   \draw (A1.out) to[short, -o] ++(0.5,0) node[above](vo){$v_o$};
22 \end{circuitikz}

```

4.2 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.
	xgeneric: Crossed generic (symmetric) bipole, type: path-style, fillable, nodename: xgenericshape. Class: resistors.
	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.

Both **shortshape** and **openshape** are not really supposed to be used; they are dummy shapes used as placeholders for the path-drawing routines.

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: Resistive sensor, type: path-style, nodename: resistivesensshape. Aliases: american resistive sensor. Class: resistors.
	ldR: Ligh-Dependent resistor, type: path-style, fillable, nodename: ldresistorshape. Aliases: american light dependent resistor. Class: resistors.

If instead **europaeansresistors** 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.

	ldR : Ligth-Dependent resistor, type: path-style, fillable, nodename: ldgenericshape. Aliases: european light dependent resistor. 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.

4.2.1 Potentiometers: wiper position

Since version 0.9.5, you can control the position of the wiper in potentiometers using the key **wiper pos**, which is a number in the range [0,1]. The default middle position is **wiper pos=0.5**.

	<pre> 1 \begin{circuitikz}[american] 2 \ctikzset{resistors/width=1.5, resistors/zigs=9} 3 \draw (0,0) to[pR, name=A] ++(0,-4); 4 \draw (1.5,0) to[pR, wiper pos=0.3, name=B] ++(0,-4); 5 \ctikzset{european resistors} 6 \draw (3,0) to[pR, wiper pos=0.8, name=C] ++(0,-4); 7 \foreach \i in {A, B, C} 8 \node[right] at (\i.wiper) {\i}; 9 \end{circuitikz} </pre>
---	--

Since version 1.6.0, potentiometers and variable resistors have extra anchors¹², to allow this kind of circuit (that seems to be common in some region):

¹²Thanks to a suggestion by [Dr. Matthias Jung on GitHub](#)



```

1 \begin{circuitikz}[european]
2   \draw (0,0) to[battery2, l=E] ++(0,4.5)
3     -- ++(2,0) coordinate(tmp)
4       to[vR, l2_=$P_1$ and \SI{10}{\kohm}, mirror,
5         invert, name=P]
6         (0,0-|tmp) -- (0,0);
7   \draw (0,0-|tmp) -- ++(1.5,0)
8     to[R=$R_1$, -*] ++(0,2) coordinate(p)
9     |- (P.wiper);
10  \draw (p) to[rmeterwa, t=V] (tmp-|p) -- (tmp);
11 \end{circuitikz}

```

4.2.2 Generic sensors anchors

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



```

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

```

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

4.2.3 Resistive components customization

4.2.3.1 Geometry. 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).



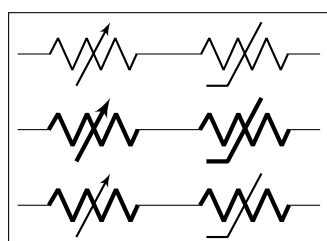
```

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

```

4.2.3.2 Thickness. The line thickness of the resistive components is governed by the class thickness; you can change it assigning a value to the key `resistors/thickness` (default `none`, that means `bipoles/thickness` is used, and that defaults to 2.0; the value is relative to the base line thickness).

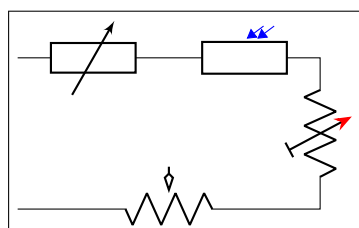
We can call *modifiers* the elements that are added to the basic shape to express some characteristics of the component; for example the arrows for the variable resistors or the bar for the sensors. Normally the thickness of these elements are the same as the one chosen for the component¹³. You can change their thickness with the class key `modifier thickness` which is relative to the main component thickness.



```
1 \begin{circuitikz}[american]
2   \draw (0,2) to[vR] ++(2,0) to[sR] ++(2,0);
3   \ctikzset{resistors/thickness=4}
4   \draw (0,1) to[vR] ++(2,0) to[sR] ++(2,0);
5   \ctikzset{resistors/modifier thickness=0.5}
6   \draw (0,0) to[vR] ++(2,0) to[sR] ++(2,0);
7 \end{circuitikz}
```

4.2.3.3 Arrows. You can change the arrow tips used in tunable resistors (`vR`, `tgeneric`) with the key `tunable end arrow` and in potentiometers with the key `wiper end arrow` (by default the key is the word “default” to obtain the default arrow, which is `latexslim` for both). Also you can change the start arrow with the corresponding `tunable start arrow` or `wiper start arrow` (the default value “default” is equivalent to `{}` for both, which means no arrow).

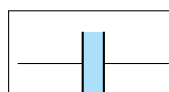
You can change that globally or locally, as ever. The tip specification is the one you can find in the TikZ manual (“Arrow Tip Specifications”). For the photoresistor and the two “flavors” of the light-dependent resistor (`ldR`, `american` or `european`), the style of the arrows follow the `opto` commands as in the photodiodes and phototransistor: see 4.4.3.1.



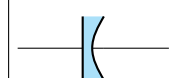
```
1 \begin{tikzpicture}[american]
2   % globally all the potentiometers
3   \ctikzset{wiper end arrow={Kite[open]},
4             opto arrows/color=blue, opto end arrow={Triangle}}
5   \draw (0,0) to[tgeneric] ++(2,0) to[phR] ++(2,0)
6   % set locally on this variable resistor
7       to[vR, tunable end arrow={Stealth[red]},
8       tunable start arrow={Bar}, invert] ++(0,-2)
9       to[pR, mirror] ++(-4,0);
10 \end{tikzpicture}
```

4.3 Capacitors and inductors: dynamical bipoles

4.3.1 Capacitors



capacitor: Capacitor, type: `path-style`, fillable, nodename: `capacitorshape`. Aliases: `C`. Class: `capacitors`.



curved capacitor: Curved (polarized) capacitor, type: `path-style`, fillable, nodename: `ccapacitorshape`. Aliases: `cC`. Class: `capacitors`.

¹³Due to a bug in versions before 1.3.4, that didn't happen for thermistors

	ecapacitor : Electrolytic capacitor, type: path-style, fillable, nodename: ecapacitorshape. Aliases: eC, elko. Class: capacitors.
	variable capacitor : Variable capacitor, type: path-style, fillable, nodename: vcapacitorshape. Aliases: vC. Class: capacitors.
	capacitive sensor : Capacitive sensor, type: path-style, fillable, nodename: capacitivesensshape. Aliases: sC. Class: capacitors.
	piezoelectric : Piezoelectric Element, type: path-style, fillable, nodename: piezoelectricshape. Aliases: PZ. Class: capacitors.
	cpe : Constant Phase Element, type: path-style, fillable, nodename: cpeshape. Aliases: cpe. Class: capacitors.
	feC : Ferroelectric capacitor ¹⁴ , type: path-style, fillable, nodename: ferrocapshape. Aliases: ferrocap. Class: capacitors.

Capacitors are fillable since v1.4.1; this is normally just a stylistic option but in the case of ferroelectric capacitors that could be used to show the state of the hysteresis of the component.



There is also the (deprecated¹⁵ — its polarity is not coherent with the rest of the components) `polar capacitor`; please do not use it.

4.3.2 Capacitive sensors anchors

For capacitive sensors, you have the same anchors than in the case of resistive sensors, see section 4.2.2.

¹⁴suggested by [Mayeul Cantan](#)

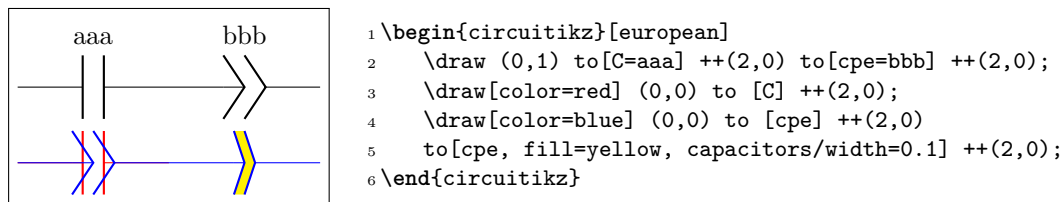
¹⁵Thanks to [Anshul Singhv](#) for noticing.

4.3.3 Capacitors customizations

You can change the scale of the capacitors by setting the key `capacitors/scale` to something different from the default 1.0. For thickness, you can use the same keys (applied to the `capacitors` class) as for resistors in 4.2.3.2.

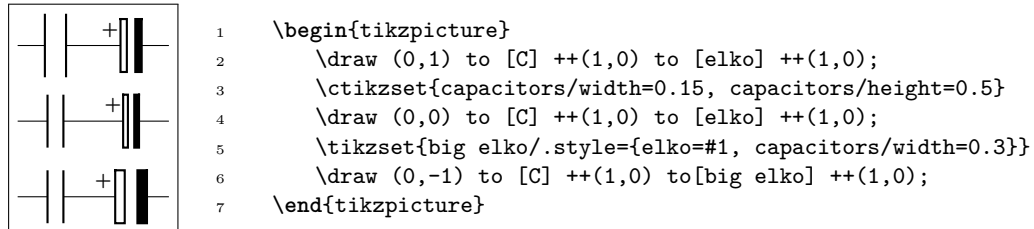
Variable capacitors arrow tips follow the settings of resistors, see section 4.2.3.3.

The relative size of the capacitors is a bit of a mixed bag, because each one has historically different internal parameters that makes maintaining coherence quite difficult. In v1.4.1 this has changed and now you can use styling options to change the way the capacitors look. The main parameter you can set is `capacitors/width` (default 0.2), which controls the standard distance between plates. That will change all the components (notice that `piezoelectric` and `cpe` default width is twice the size of a standard capacitor — although this is not evident for the `cpe` given its shape.)




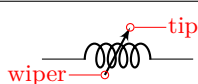
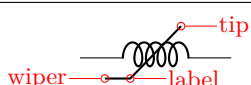
The `capacitors/height` key is available also to set the height of the capacitor; the default is 0.6 for most of the capacitors, but 0.5 for electrolytic ones and 0.7 for piezoelectric. When used, it will set all of them at the same value, which is a good thing.

If you want that only a specific kind of capacitor has a different value for a key, you can always use a style which will have a local scope, as in the following example.



4.3.4 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: <code>path-style</code> , nodename: <code>cuteinductorshape</code> . Aliases: <code>cute inductor</code> . Class: <code>inductors</code> .
	vL: Variable inductor, type: <code>path-style</code> , nodename: <code>vcuteinductorshape</code> . Aliases: <code>variable cute inductor</code> . Class: <code>inductors</code> .
	sL: Inductive sensor, type: <code>path-style</code> , nodename: <code>scuteinductorshape</code> . Aliases: <code>cute inductive sensor</code> . Class: <code>inductors</code> .

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

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

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

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

For historical reasons, *chokes* come only in the *cute*. You can use the `core west` and `core east` anchors (see 4.3.6.2) to build your own core lines for the other inductors.

	cute choke: Choke, type: <code>path-style</code> , nodename: <code>cutechokeshape</code> . Class: <code>inductors</code> .
---	---

4.3.5 Inductors customizations

You can change the scale of the inductors by setting the key `inductors/scale` to something different from the default 1.0. For thickness, you can use the same keys (applied to the `inductors` class) as for resistors in 4.2.3.2.

Variable inductors arrow tips follow the settings of resistors, see section 4.2.3.3.

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). **Notice** that the minimum number of `coils` is 1 for american inductors, and 2 for cute ones.



```

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

```

4.3.5.1 Chokes can have single and double lines, and can have the line thickness adjusted (the value is relative to the thickness of the inductor). In general, you should use the anchors (see 4.3.6.2) to add core lines to inductors.



```

1 \begin{circuitikz}[american]
2     \draw (0,0) to[cute choke] ++(3,0);
3     \draw (0,-1) to[cute choke, twolineschoke] ++(3,0);
4
5     \ctikzset{bipoles/cutechoke/ctick=2, twolineschoke}
6
7     \draw (0,-2) to[cute choke] ++(3,0);
8     \draw (0,-3) to[cute choke, onelinechoke] ++(3,0);
9 \end{circuitikz}

```

4.3.6 Inductors anchors

For inductive sensors, you have the same anchors than in the case of resistive sensors, see section 4.2.2.

4.3.6.1 Taps. Inductors have an additional anchor, called `midtap`, that connects to the center of the coil “wire”. Notice that this anchor could be on one side or the other of the component, depending on the number of loops of the element; if you need a fixed position, you can use the geographical anchors.

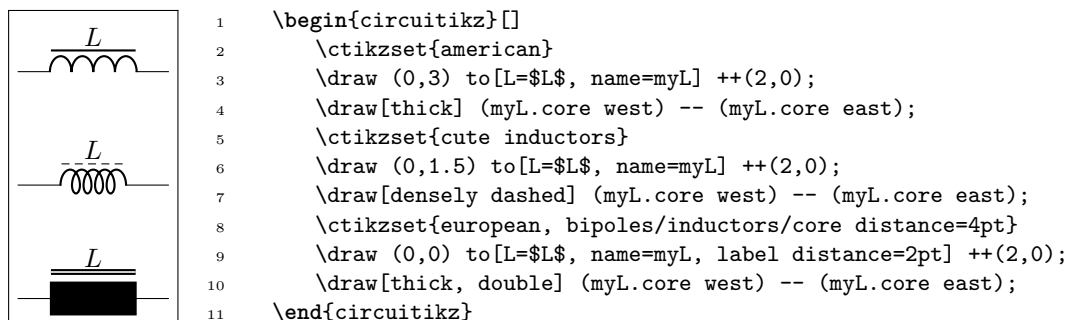


```

1 \begin{circuitikz}[
2     loops/.style={circuitikz/inductors/coils=#1}]
3 \ctikzset{cute inductors}
4 \draw (0,2) to[L, loops=5, name=A] ++(2,0)
5 to[L, loops=6, name=B] ++(2,0);
6 \ctikzset{american inductors}
7 \draw (0,0) to[L, loops=5, name=C] ++(2,0)
8 to[L, loops=6, name=D] ++(2,0);
9 \foreach \i in {A, B, C, D}
10 \node[circle, fill=red, inner sep=1pt] at (\i.midetap){};
11 \end{circuitikz}

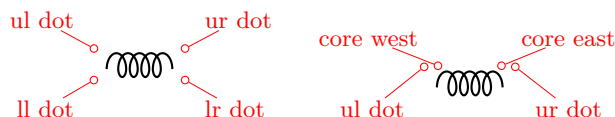
```

4.3.6.2 Core anchors. Inductors have additional anchors to add core lines (for historical reasons, there is a `cute choke` component also, but to use inductors in the chosen style you’d better use these anchors). The anchors are called `core west` and `core east` and they are positioned at a distance that you can tweak with the `\ctikzset` key `bipoles/inductors/core distance` (default 2pt).

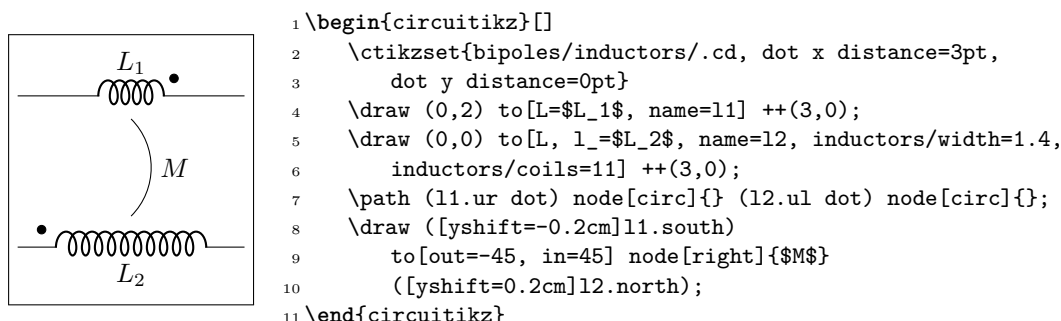


Notice that the core lines will **not** change the position of labels. You have to move them by hand if needed (or position them on the other side); see 5.1.1.1.

4.3.6.3 Dot anchors. Inductances also have “dot” anchors¹⁶, to help positioning dots when specifying mutual inductance signs. The anchors are name `lr dot` (for lower right dot), `ur dot` (upper right) and so on:



and as you can see, you have to be careful if you use dot anchors and core anchors together. You can change the position of the dots by changing the keys `bipoles/inductors/dot x distance` (default 4pt), which represent how much the dot extends outside the component width, and the corresponding `dot y distance` (default 1pt), which serves the same scope in the height direction.



Notice that the position of the dot anchors does not coincide with the position of the dots in the transformers (see section 4.19), because there they depend on the size of the complete double-bipole more than on the size of the inductances.

4.4 Diodes and such

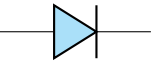
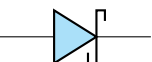
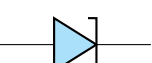

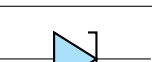
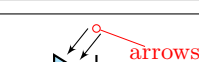
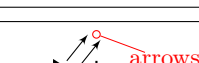
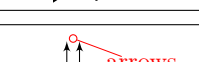

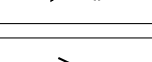
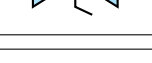
There are three basic styles for diodes: `empty` (fillable in color), `full` (completely filled with the draw color) and `stroke` (empty, but with a line across them).

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).

¹⁶proposed by Romano in a discussion by [GitHub user AndreaDiPietro92](#)

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!).

	<code>empty diode</code> : Empty diode, type: <code>path-style</code> , fillable, nodename: <code>emptydiodeshape</code> . Aliases: <code>Do</code> . Class: <code>diodes</code> .
	<code>empty Schottky diode</code> : Empty Schottky diode, type: <code>path-style</code> , fillable, nodename: <code>emptysdiodeshape</code> . Aliases: <code>sDo</code> . Class: <code>diodes</code> .
	<code>empty Zener diode</code> : Empty Zener diode, type: <code>path-style</code> , fillable, nodename: <code>emptyzdiodeshape</code> . Aliases: <code>zDo</code> . Class: <code>diodes</code> .
	<code>empty ZZener diode</code> : Empty ZZener diode, type: <code>path-style</code> , fillable, nodename: <code>emptyzzdiodeshape</code> . Aliases: <code>zzDo</code> . Class: <code>diodes</code> .
	<code>empty tunnel diode</code> : Empty tunnel diode, type: <code>path-style</code> , fillable, nodename: <code>emptytdiodeshape</code> . Aliases: <code>tDo</code> . Class: <code>diodes</code> .
	<code>empty photodiode</code> : Empty photodiode, type: <code>path-style</code> , fillable, nodename: <code>emptytpdiodeshape</code> . Aliases: <code>pDo</code> . Class: <code>diodes</code> .
	<code>empty led</code> : Empty led, type: <code>path-style</code> , fillable, nodename: <code>emptylediodeshape</code> . Aliases: <code>leDo</code> . Class: <code>diodes</code> .
	<code>empty laser diode</code> : Empty laser diode ¹⁷ , type: <code>path-style</code> , fillable, nodename: <code>emptylaserdiodeshape</code> . Aliases: <code>lasD</code> . Class: <code>diodes</code> .
	<code>empty varcap</code> : Empty varcap, type: <code>path-style</code> , fillable, nodename: <code>emptyvarcapshape</code> . Aliases: <code>VCo</code> . Class: <code>diodes</code> .
	<code>empty TVS diode</code> : Empty TVS diode, transorb ¹⁸ , type: <code>path-style</code> , fillable, nodename: <code>emptytvsvdiodeshape</code> . Aliases: <code>tvsvDo</code> . Class: <code>diodes</code> .
	<code>empty Shockley diode</code> : Empty Shockley diode ¹⁹ , type: <code>path-style</code> , fillable, nodename: <code>emptyshdiodeshape</code> . Aliases: <code>shDo</code> . Class: <code>diodes</code> .

¹⁷Added by André Alves in v1.4.4

¹⁸Transobs were suggested by <https://tex.stackexchange.com/q/642219/38080>

¹⁹Shockley diodes were suggested by <https://tex.stackexchange.com/q/646039/38080>

	empty bidirectionaldiode: Empty bidirectional-diode, 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: fulllediodeshape. Aliases: leD*. Class: diodes.
	full laser diode: Full laser diode, type: path-style, nodename: fulllaserdiodeshape. Aliases: lasD*. Class: diodes.
	full varcap: Full varcap, type: path-style, nodename: fullvarcapshape. Aliases: VC*. Class: diodes.
	full TVS diode: Full TVS diode, transorb, type: path-style, nodename: fulltvsvdiodeshape. Aliases: tvsvD*. Class: diodes.
	full Shockley diode: Full Shockley diode, type: path-style, nodename: fullshdiodeshape. Aliases: shD*. Class: diodes.
	full bidirectionaldiode: Full bidirectional-diode, 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 laser diode: Stroke laser diode, type: path-style, fillable, nodename: emptylaserdiodeshape. Aliases: lasD-. Class: diodes.
	stroke varcap: Stroke varcap, type: path-style, fillable, nodename: emptyvarcapshape. Aliases: VC-. Class: diodes.

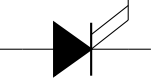
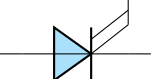
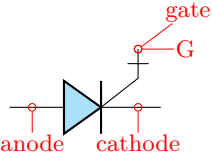
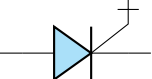
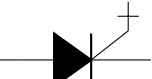
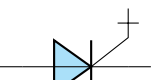
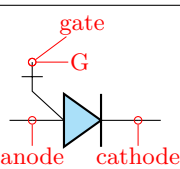
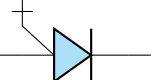
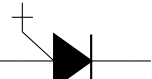
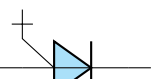
4.4.1 Tripole-like diodes

The following tripoles are entered with the usual command, of the form `to[Tr, ...]`. In the following list you can see the traditional, or `legacy`, shape of the Thyristors-type devices.

	full diode: Full diode, type: path-style, nodename: fulldiodeshape. Aliases: D*. Class: diodes.
	stroke diode: Stroke diode, type: path-style, fillable, nodename: emptydiodeshape. Aliases: D-. Class: diodes.
	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.
	put: Standard Programmable Unipolar Transistor ²⁰ (shape depends on package option), type: path-style, fillable, nodename: emptyputshape. Aliases: PUT. Class: diodes.
	empty put: Empty PUT, type: path-style, fillable, nodename: emptyputshape. Aliases: PUTo. Class: diodes.
	full put: Full PUT, type: path-style, nodename: fullputshape. Aliases: PUT*. Class: diodes.
	stroke put: Stroke PUT, type: path-style, fillable, nodename: emptyputshape. Aliases: PUT-. Class: diodes.
	gto: Standard GTO (shape depends on package option), type: path-style, fillable, nodename: emptygtoshape. Aliases: GTO. Class: diodes.
	empty gto: Empty GTO, type: path-style, fillable, nodename: emptygtoshape. Aliases: GTOo. Class: diodes.

²⁰This components, and the GTO family, has been suggested by [GitHub user JetherReis](#).

	full gto : Full GTO, type: path-style, nodename: fullgtoshape. Aliases: GTO*. Class: diodes.
	stroke gto : Stroke GTO, type: path-style, fillable, nodename: emptygtoshape. Aliases: GTO-. Class: diodes.
	gtobar : Standard GTO with bar-type gate (shape depends on package option), type: path-style, fillable, nodename: emptygtobarshape. Aliases: GTOb. Class: diodes.
	empty gtobar : Empty GTO, bar-type, type: path-style, fillable, nodename: emptygtobarshape. Aliases: GTOb-. Class: diodes.
	full gtobar : Full GTO, bar-type, type: path-style, nodename: fullgtobarshape. Aliases: GTOb*. Class: diodes.
	stroke gtobar : Stroke GTO, bar type, type: path-style, fillable, nodename: emptygtobarshape. Aliases: GTOb-. Class: diodes.
	agtobar : Standard GTO with bar-type gate on anode (shape depends on package option), type: path-style, fillable, nodename: emptyagtobarshape. Aliases: aGTOb. Class: diodes.
	empty agtobar : Empty GTO, bar-type on anode, type: path-style, fillable, nodename: emptyagtobarshape. Aliases: aGTOb-. Class: diodes.
	full agtobar : Full GTO, bar-type on anode, type: path-style, nodename: fullagtobarshape. Aliases: aGTOb*. Class: diodes.
	stroke agtobar : Stroke GTO, bar-type on anode, type: path-style, fillable, nodename: emptyagtobarshape. Aliases: aGTOb-. Class: diodes.

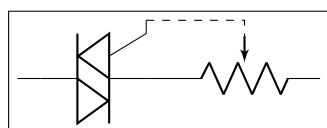
For basically stylistical reasons, there is a different, more compact, shape available for them, activated with the key `thyristor style=compact` (the default is `legacy`). All the devices above are present, we will show here just the automatic version for shortness.

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

	<p>thyristor: Standard thyristor (shape depends on package option), type: <code>path-style</code>, fillable, nodename: <code>emptythyristorshape</code>. Aliases: <code>Ty</code>. Class: <code>diodes</code>.</p>
	<p>put: Standard Programmable Unipolar Transistor (shape depends on package option), type: <code>path-style</code>, fillable, nodename: <code>emptyputshape</code>. Aliases: <code>PUT</code>. Class: <code>diodes</code>.</p>
	<p>gto: Standard gto (shape depends on package option), type: <code>path-style</code>, fillable, nodename: <code>emptygtoshape</code>. Aliases: <code>GTO</code>. Class: <code>diodes</code>.</p>
	<p>gtobar: Standard GTO with a bar symbol on the gate (shape depends on package option), type: <code>path-style</code>, fillable, nodename: <code>emptygtobarshape</code>. Aliases: <code>GTOb</code>. Class: <code>diodes</code>.</p>
	<p>agtobar: Standard GTO with bar-type gate on anode (shape depends on package option), type: <code>path-style</code>, fillable, nodename: <code>emptyagtobarshape</code>. Aliases: <code>aGTOb</code>. Class: <code>diodes</code>.</p>

4.4.2 Thyristors anchors and customization

When inserting a thyristor, 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:

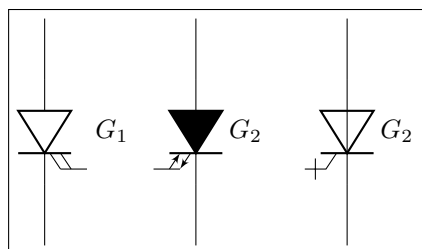


```

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

```

As commented above, you can change the shape of these devices (globally or locally) setting the key **thyristor style=compact** (the default is **legacy**). Additionally, normally the plain GTO symbols come without the arrows, but you can add them using a syntax similar to the one explained in section 4.2.3.3 using the arrow group **gto gate**.



```

1 \begin{circuitikz}[]
2   \ctikzset{thyristor style=compact}
3   \draw (0,0) to[GTO=$G_1$] ++(0,-3);
4   \ctikzset{gto gate end arrow=latexslim}
5   \draw (2,0) to[GTO*=$G_2$, mirror] ++(0,-3);
6   \draw (4,0) to[GTOb=$G_3$, mirror] ++(0,-3);
7 \end{circuitikz}

```

Notice that you can set both **gto gate end arrow** and **gto gate start arrow** — choosing just one of the two you can decide the “rotation” direction of the symbol. There is little space though, so don’t overdo it.

4.4.3 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.



```

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

```

4.4.3.1 Optical devices arrows You can change the direction of the LEDs and photodiodes' arrows by using the binary keys `led arrows from cathode` and `pd arrows to cathode` (the default are `led arrows from anode` and `pd arrows to anode`), as you can see in the following example.



```

1 \begin{circuitikz}
2   \ctikzset{led arrows from anode} % default
3   \ctikzset{pd arrows to anode} % default
4   \ctikzset{full diodes}
5   \draw (0,0) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
6   \ctikzset{stroke diodes}
7   \draw (0,-1) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
8   \ctikzset{empty diodes}
9   \draw (0,-2) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
10
11  \ctikzset{led arrows from cathode}
12  \ctikzset{pd arrows to cathode}
13  \ctikzset{full diodes}
14  \draw (0,-4) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
15  \ctikzset{stroke diodes}
16  \draw (0,-5) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
17  \ctikzset{empty diodes}
18  \draw (0,-6) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
19 \end{circuitikz}

```

Since version 1.5.5²¹, you can change the arrows used for LEDs, photodiodes and laser diodes with the generic arrows options shown in 4.2.3.3, using the name `opto`, like in the following (overdone) example. Normally you want just to change the `end arrow`...

As you can see, you can also have the option to globally change the color, relative thickness, and dash pattern by setting keys with the `\ctikzset` command (or, like in the following example, directly in the node instantiation) under the `opto arrows` hierarchy. The available keys are:

²¹Thanks to the idea by [Dr. Matthias Jung on GitHub](#).

parameter	default	description
<code>relative thickness</code>	1.0	multiply the class thickness
<code>color</code>	default	stroke color: default is the same as the component
<code>dash</code>	default	dash pattern: default means not to change the setting for the component; <code>none</code> means unbroken line; every other input is a dash pattern. ²²



4.5 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.

4.5.1 Batteries



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

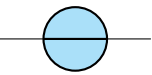
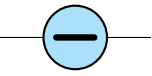
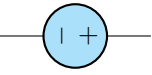
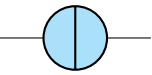
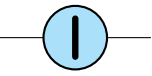
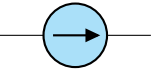


battery1: Single battery cell, type: path-style, nodename: battery1shape. Class: batteries.

²²Follows the syntax of the pattern sequence `\pgfsetdash` — see TikZ manual for details; phase is always zero. Basically you pass pairs of dash-length – blank-length dimensions, see the examples.

	battery2: Single battery cell, type: <code>path-style</code> , nodename: <code>battery2shape</code> . Class: <code>batteries</code> .
---	--

4.5.2 Stationary sources

	european voltage source: Voltage source (european style), type: <code>path-style</code> , fillable, nodename: <code>vsourceshape</code> . Aliases: <code>vsource</code> . Class: <code>sources</code> .
	cute european voltage source: Voltage source (cute european style), type: <code>path-style</code> , fillable, nodename: <code>vsourceshape</code> . Aliases: <code>vsourceshape</code> , <code>ceV</code> . Class: <code>sources</code> .
	american voltage source: Voltage source (american style), type: <code>path-style</code> , fillable, nodename: <code>vsourceshape</code> . Aliases: <code>vsourceshape</code> . Class: <code>sources</code> .
	european current source: Current source (european style), type: <code>path-style</code> , fillable, nodename: <code>isourceshape</code> . Aliases: <code>isource</code> . Class: <code>sources</code> .
	cute european current source: Current source (cute european style), type: <code>path-style</code> , fillable, nodename: <code>isourceshape</code> . Aliases: <code>isourceshape</code> , <code>ceI</code> . Class: <code>sources</code> .
	american current source: Current source (american style), type: <code>path-style</code> , fillable, nodename: <code>isourceshape</code> . Aliases: <code>isourceshape</code> . Class: <code>sources</code> .

If (default behavior) `européancurrents` 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 behavior) `europeanvoltages` option is active (or the style `[european voltages]` is used), the shorthands `voltage source`, `vsources`, 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`.

4.5.3 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.



```

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

```



4.5.4 Controlled sources

	 european controlled voltage source: Controlled voltage source (european style), type: path-style, fillable, nodename: cvsourceshape. Aliases: cvsourc. Class: csources.
	 cute european controlled voltage source: Voltage source (cute european style), type: path-style, fillable, nodename: cvsourcCshape. Aliases: cvsourcC, cceV. Class: csources.
	 american controlled voltage source: Controlled voltage source (american style), type: path-style, fillable, nodename: cvsourcAMshape. Aliases: cvsourcAM. Class: csources.
	 european controlled current source: Controlled current source (european style), type: path-style, fillable, nodename: cisourceshape. Aliases: cisourc. Class: csources.
	 cute european controlled current source: Current source (cute european style), type: path-style, fillable, nodename: cisourcCshape. Aliases: cisourcC, cceI. Class: csources.
	 american controlled current source: Controlled current source (american style), type: path-style, fillable, nodename: cisourcAMshape. Aliases: cisourcAM. Class: csources.
	 empty controlled source: Empty controlled source, type: path-style, fillable, nodename: ecsourceshape. Aliases: ecsourc. Class: csources.

If (default behaviur) `européancurrents` 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 behaviur) `européanvoltages` option is active (or the style `[european voltages]` is used), the shorthands `controlled voltage source`, `cvsources`, 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 4.5.3.

	controlled sinusoidal voltage source: Controlled sinusoidal voltage source, type: <code>path-style</code> , fillable, nodename: <code>cvsourcesinshape</code> . Aliases: <code>controlled vsourcesin</code> , <code>cvsourcesin</code> , <code>csV</code> . Class: <code>csources</code> .
	controlled sinusoidal current source: Controlled sinusoidal current source, type: <code>path-style</code> , fillable, nodename: <code>cisourcesinshape</code> . Aliases: <code>controlled isourcesin</code> , <code>cisourcesin</code> , <code>csI</code> . Class: <code>csources</code> .

4.5.5 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: <code>path-style</code> , nodename: <code>vsourcenshape</code> . Aliases: <code>vsourcen</code> , <code>nV</code> . Class: <code>sources</code> .
	noise current source: Sinusoidal current source, type: <code>path-style</code> , nodename: <code>isourcenshape</code> . Aliases: <code>isourcen</code> , <code>nI</code> . Class: <code>sources</code> .

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

		<code>1 \begin{circuitikz}</code>
		<code>2 \draw(0,0) to [nV, l=\$e_n\$] ++(2,0);</code>
		<code>3 \draw(0,-2) to [nI, l=\$i_n\$] ++(2,0);</code>
		<code>4 \begin{scope}[circuitikz/bipoles/noise sources/</code>
		<code>fillcolor=red!50]</code>
		<code>5 \draw(3,0) to [nV, l=\$e_n\$] ++(2,0);</code>
		<code>6 \draw(3,-2) to [nI, l=\$i_n\$] ++(2,0);</code>
		<code>7 \end{scope}</code>
		<code>8 \end{circuitikz}</code>

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



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



4.5.6 Special sources

	square voltage source: Square voltage source, type: path-style, fillable, nodename: vsourcesquashape. 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.
	pvsources: Photovoltaic-voltage source, type: path-style, fillable, nodename: pvsourceshape. Class: sources.
	pvmodule: Photovoltaic module source ²³ , type: path-style, fillable, nodename: pvmoduleshape. Class: sources.
	ioosources: Double Zero style current source, type: path-style, fillable, nodename: oosourceshape. Class: sources.

²³ Added by André Alves in v1.3.5

	voosource: Double Zero style voltage source, type: path-style, fillable, nodename: oosourceshape. Class: sources.
	oosourcetrans: transformer source ²⁴ , type: path-style, fillable, nodename: oosourcetransshape. Class: sources.
	ooosource: transformer with three windings ²⁵ , type: path-style, fillable, nodename: ooosourceshape. Class: sources.

The transformer shapes vector group options can be specified for the primary (**prim=*value***), the secondary (**sec=*value***) and tertiary (**tert=*value***) three-phase vector groups: the value can be one of delta, wye and zig.

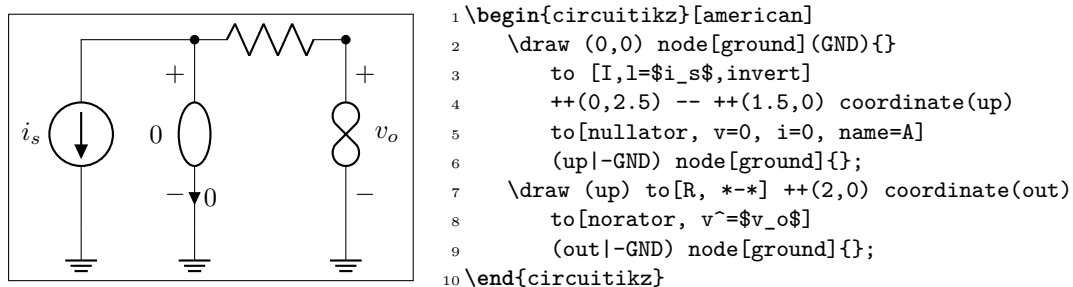


4.5.7 Nullator and norator

These are special elements used in some approaches to model ideal amplifiers²⁶.

	nullator: Nullator element (virtual short circuit; forces V and I to zero), type: path-style, fillable, nodename: nullatorshape. Class: sources.
	norator: Norator element (admits any combination of V and I), type: path-style, fillable, nodename: noratorshape. Class: sources.

They are in the **sources** class, but they are not treated like sources in the labeling sense (they have both **bipoles/is voltage=false** and **bipoles/is current=false**, see 5.2.2).



²⁴The **oosourcetrans** and **ooosource** components have been added by [user @olffline on GitHub](#)

²⁵The component here is scaled up 1.5 times to better show the anchors.

²⁶See [the Wikipedia article](#); suggested by [user atticus-sullivan on GitHub](#).

The symbol shapes used here seems to be the most common in publications; if you prefer the shapes from the Wikipedia article, you can use the following definitions:



```

1 \tikzset{noratorW/.style={voosource,
2     bipoles/oosource/circlesize=0.5,
3     bipoles/oosource/circleoffset=0.5},
4     nullatorW/.style={esource, sources/scale=0.5}}
5 \begin{tikzpicture}
6     \draw (0,0) to[noratorW] ++(2,0) to[noratorW] ++(0, -2);
7 \end{tikzpicture}

```

4.5.8 DC sources



dcvsource: DC voltage source, type: `path-style`, fillable, nodename: `dcvsource`shape. Class: `sources`.



dcisource: DC current source, type: `path-style`, fillable, nodename: `dcisource`shape. 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).



```

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

```

4.5.9 Sources customizations

4.5.9.1 Size. 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.

Notice that the size of the double-circle sources (and of the triple-circle one) are tuned so that the full source occupy more or less the same horizontal space than one of the single-circle one; as a consequence, the circles are much smaller. If you want to have the same circle radius, you have to scale (locally!) those sources by one factor that is 1.5384 (1/0.65) for `oosource`, 1.6667 (1/0.6) for `oosourcetrans`, and 1.8182 (1/0.55) for `ooosource`.



```

1 \begin{circuitikz}
2     \draw[color=red] (0,0) to[esource] ++(3,0);
3     \draw (0,0) to[oosourcetrans, prim=delta, sec=wye,
4         sources/scale=1.667] ++(3,0);
5 \end{circuitikz}

```

4.5.9.2 Waveform symbols. Internal symbols of sinusoidal, triangular and square sources are drawn with the same line thickness as the component by default. You can modify this by setting the key `sources/symbols/thickness` for independent sources and the corresponding `csource/...` for dependent ones. The value used here is relative to the component (i.e. the circle) value.

Normally the symbol is oriented in the same direction as the line, and rotate rigidly with the component; you can change this orientation using the key `sources/symbols/rotate` or `csource/...`. The default value is 90 which correspond to the “line” direction (remember, path components are defined as horizontal ones). If instead of an angle value you use `auto`, the symbol will be rotated so that the waveform is always vertical, similar to what happens in instruments:



```

1 \begin{circuitikz}
2   \draw (0,1) to[sqV] ++(3,0)
3     to[sqV] ++(1,-1)
4     to[sqV] ++(0,-3);
5   \ctikzset{sources/symbol/rotate=auto}
6   \ctikzset{sources/symbol/rotate=auto, sources/symbol/
7     thickness=3}
8   \draw[color=red] (0,0) to[sqV] ++(3,0)
9     to[sqV] ++(0,-3)
10    to[sqV] (0,0);
11 \end{circuitikz}

```

4.5.9.3 Polarity symbols. The symbols drawn into the american voltage source²⁷ can be changed by using the `\ctikzset` keys `bipoles/vsourceam/inner plus` and `.../inner minus` (by default they are `+$` and `-$` respectively, in the current font), and move them nearer or farther away by twiddling `bipoles/vsourceam/margin` (default 0.7, less means nearer).

You can do the same with the american controlled voltage sources, substituting `cvsourc` to `vsource` (notice the initial “c”).



```

1 \begin{circuitikz}[american]
2   \ctikzset{bipoles/vsourceam/inner plus={\tiny $+$}}
3   \ctikzset{bipoles/vsourceam/inner minus={\tiny $-$}}
4   \draw (0,0) to[V, l_=$V$] ++(0,3)
5     to[R=\SI{5}{\ohm}] ++(3,0)
6     to[V, invert,
7       bipoles/vsourceam/inner plus={\color{red}\tiny $\oplus$},
8       bipoles/vsourceam/inner minus={\color{blue}\tiny $\ominus$},
9       bipoles/vsourceam/margin=0.5]
10    ++(0,-3) to[short, -*] (0,0) node[ground]{};
11 \end{circuitikz}

```

4.5.9.4 Three-phase symbols. The three-phase symbols `delta`, `wye`, and `zig` follows the line thickness exactly as the waveform ones (see above). Additionally, you can scale them up and down by changing the value of the keys `sources/symbol/delta scale`, `.../wye scale`, and `.../zig scale` (default 1).

²⁷Since version 1.1.0, thanks to the suggestions and discussion [in this TeX.SX question](#).



```

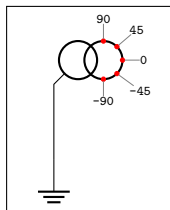
1 \begin{circuitikz}[scale=1.8, transform shape]
2   \tikzset{myoosource/.style={oosource,
3     prim=wye, sec=delta, tert=zig,
4     }}
5   \draw (0,2) to[myoosource] ++(2,0);
6   \ctikzset{%
7     sources/symbol/thickness=0.5,
8     sources/symbol/delta scale=1.2,
9     sources/symbol/wye scale=1.4,
10    sources/symbol/zig scale=1.3,
11  }
12   \draw (0,0) to[myoosource] ++(2,0);
13 \end{circuitikz}

```

4.5.10 Source borders

Unfortunately, the border of the sources is only easily accessed if some anchor is provided. The border anchors of the shapes are not tight on them (see section 3.1.2), which is not easily changeable, given that the algorithm that positions the labels depends on it.

On the other hand, TikZ powerful partway coordinate calculation (around section 13.5.3 of the manual) makes it possible to easily identify points on a circle if the center and one point of the circle are known, as you can see in the following example.

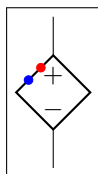


```

1 \begin{tikzpicture}[]
2   \path (0,0) to[oosourcetrans, name=T, ]++(2,0);
3   % Use partway modifiers to reach a point on the left circle
4   \draw ($(T.centerprim)!1!45:(T.left)$) -- ++(-135:0.2)
5     -- ++(0,-1) node[ground]{};
6   \begin{scope}[font=\tiny\ttfamily, pin distance=2mm, inner sep=0pt]
7     \foreach \a in {-90,-45,...,90}
8       \node [circ, scale=0.5, pin=\a:\a, color=red] at
9         ($(T.centersec)!1!\a:(T.right)$) {};
10  \end{scope}
11 \end{tikzpicture}

```

A similar approach can be used for dependent sources. Just remember that the anchors move (rotate) together with the component.

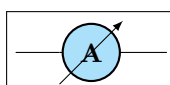


```

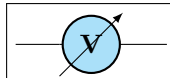
1 \begin{tikzpicture}[american]
2   \draw (0,0) to [cvsource, name=S] ++(0,2);
3   \node [circ,red] at ($(S.e)!0.3333!(S.n)$) {};
4   \node [circ,blue] at ($(S.e)!0.6666!(S.n)$) {};
5 \end{tikzpicture}

```

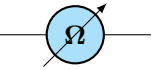
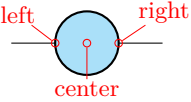
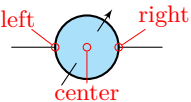
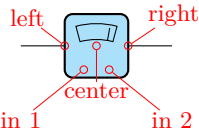
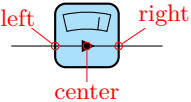
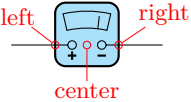
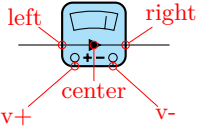
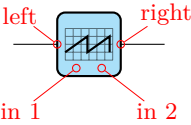
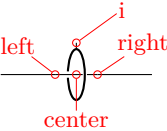
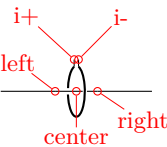
4.6 Instruments



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



voltmeter: Voltmeter, type: path-style, fillable, nodename: voltmetershape. Class: instruments.

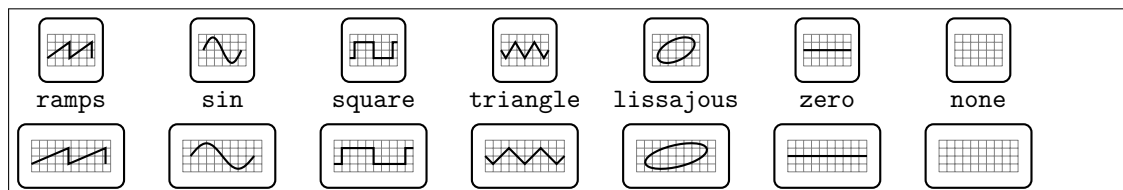
	ohmmeter: Ohmmeter, type: path-style, fillable, nodename: ohmmetershape. Class: instruments.
	rmeter: Round meter (use t=... for the symbol), type: path-style, fillable, nodename: rmetershape. Class: instruments.
	rmeterwa: Round meter with arrow (use t=... for the symbol), type: path-style, fillable, nodename: rmeterwashape. Class: instruments.
	smeter: Square meter (use t=... for the symbol), type: path-style, fillable, nodename: smetershape. Class: instruments.
	qprobe: QUCS-style current probe, type: path-style, fillable, nodename: qprobeshape. 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 ²⁸ , 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.

4.6.1 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.

²⁸Suggested by @nobl on GitHub

4.6.1.1 Oscilloscope waveform. You can change the waveform shown in the oscilloscope “screen”²⁹. To change it, you just set the key `bipoles/oscope/waveform` to one of the available shape. You have available the shapes in the following list (the default is `ramps`):



```

1 \begin{circuitikz}
2   \foreach [count=\i] \wvf in {ramps, sin, square, triangle, lissajous, zero, none} {
3     \ctikzset{bipoles/oscope/waveform=\wvf}
4     \draw ({2*\i},1.4) node[oscopeshape](0){}
5         ({2*\i},0.65) node[anchor=base]{\texttt{\wvf}};
6   }
7   \ctikzset{bipoles/oscope/width=1.0}
8   \foreach [count=\i] \wvf in {ramps, sin, square, triangle, lissajous, zero, none} {
9     \ctikzset{bipoles/oscope/waveform=\wvf}
10    \draw ({2*\i},0) node[oscopeshape]{};
11  }
12 \end{circuitikz}

```

If you want more or different shapes, you can define your owns, but you have to use low-level `pgf` commands (see part IX, “The Basic Layer”, in the PGF/TikZ manual). The code is executed into a `\pgfscope ...\endpgfscope` environment, and it must use the path built with a `\pgfusepath`. The coordinates have been scaled so that the external box of the scope is a rectangle between $(-1\text{cm}, -1\text{cm})$ and $(1\text{cm}, 1\text{cm})$; the oscilloscope grid is fixed and painted between $(-0.75\text{cm}, -0.5\text{cm})$ and $(0.75\text{cm}, 0.5\text{cm})$. If you stretch the scope with the `...width` or `...height` keys, the drawing will be stretched too.

```

1   \ctikzset{%
2     bipoles/oscope/waveform/mywave/.code={%
3       \pgfsetcolor{red}
4       \pgfpathmoveto{\pgfpoint{-0.75cm}{-0.5cm}}
5       \pgfpathlineto{\pgfpoint{0.75cm}{0.5cm}}
6       \pgfusepath{draw}
7       \pgfsetcolor{green}
8       \pgfpathmoveto{\pgfpoint{-0.75cm}{0.5cm}}
9       \pgfpathlineto{\pgfpoint{0.75cm}{-0.5cm}}
10      \pgfusepath{draw}
11    }}
12 \begin{circuitikz}
13   \ctikzset{bipoles/oscope/waveform=mywave}
14   \draw (0,0) node[oscopeshape]{};
15 \end{circuitikz}

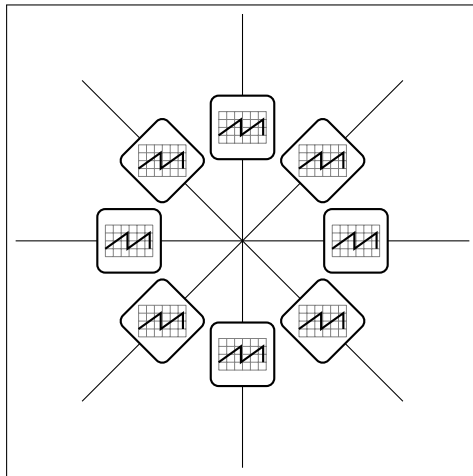
```



4.6.2 Rotation-invariant elements

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

²⁹Suggested by [Mario Tafur on TeX.SX](#)



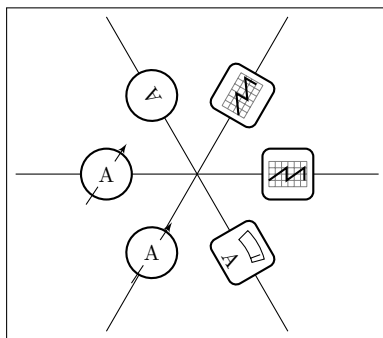
```

1 \begin{circuitikz}
2   \foreach \a in {0,45,...,350} {
3     \draw (0,0) to[oscope] (\a:3);
4   }
5 \end{circuitikz}

```

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

However, if you prefer that the `oscope`, `rmeter`, `smeter` and `rmeterwa` instruments rotate the text or the diagram, you can use the key or style `rotated instruments` (the default style is `straight instruments`).



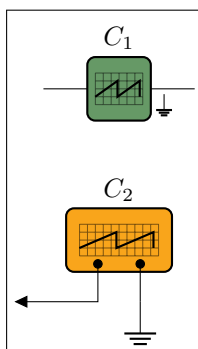
```

1 \begin{circuitikz}[scale=0.8, transform shape]
2 \ctikzset{rotated instruments} % new default
3 \draw (0,0) to[oscope] ++(0:3);
4 \draw (0,0) to[oscope] ++(60:3);
5 \draw (0,0) to[rmeter, t=A] ++(120:3);
6 % local override
7 \draw (0,0) to[rmeterwa, t=A, straight instruments]
8   ++(180:3);
9 \ctikzset{straight instruments} % back to default
10 \draw (0,0) to[rmeterwa, t=A] ++(240:3);
11 % local override
12 \draw (0,0) to[smeter, t=A, rotated instruments]
13   ++(300:3);
14 \end{circuitikz}

```

4.6.3 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` (which both default to 0.6).



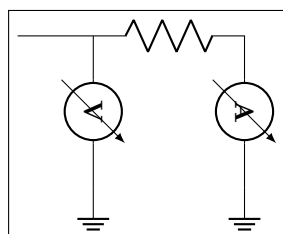
```

1 \begin{circuitikz}
2   \draw (0,1)
3     to[oscope=$C_1$, fill=green!20!gray, name=O1] ++(2,0);
4   \path (O1.right)
5     node[ground, scale=0.5, below right=4pt]{};
6   \ctikzset{bipoles/oscope/width=1.0}
7   \draw (1,-1)
8     node[oscopeshape, fill=yellow!20!orange] (O2){$C_2$};
9   \draw (O2.in 2) to[short, *-] ++(0,-0.5) node[ground]{};
10  \draw (O2.in 1) to[short, *-] ++(0,-0.5)
11    -- ++(-1,0) node[curarrow, xscale=-1]{};
12 \end{circuitikz}

```

4.6.4 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.

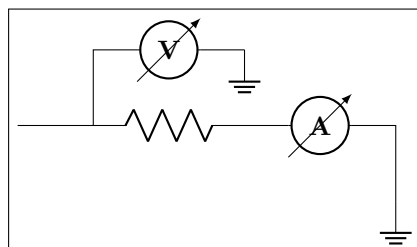


```

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

```

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

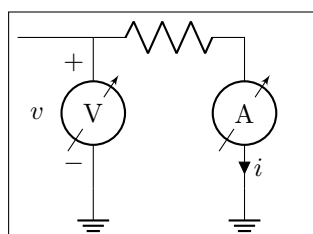


```

1 \begin{circuitikz}
2   \draw (0,0) -- ++(1,0) to[R] ++(2,0)
3     to [ammeter] ++(2,0) --
4     ++(0,-1) node[ground]{};
5   \draw (1,0) -- (1,1) to[voltmeter]
6     ++(2,0) node[ground]{};
7 \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).

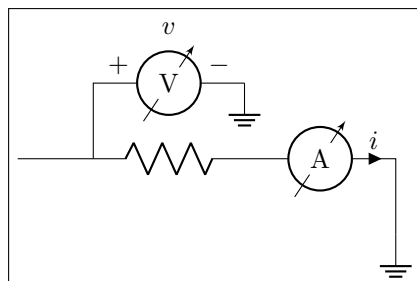


```

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

```

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

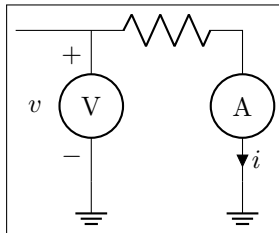


```

1 \begin{circuitikz}[american]
2   \draw (0,0) -- ++(1,0) to[R] ++(2,0)
3     to [rmeterwa, t=A, i=$i$] ++(2,0) --
4     ++(0,-1) node[ground]{};
5   \draw (1,0) -- (1,1) to[rmeterwa, t=V, v^=$v$]
6     ++(2,0) node[ground]{};
7 \end{circuitikz}

```

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

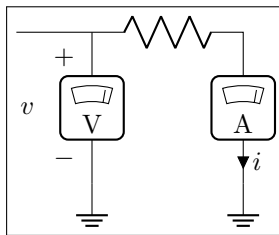


```

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

```

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

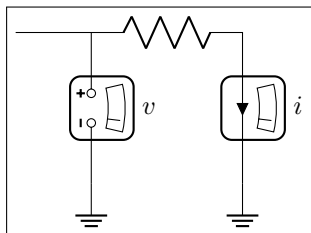


```

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

```

Another possibility is to use QUCS³⁰-style probes, which have the nice property of explicitly showing the type of connection (in series or parallel) of the meter:

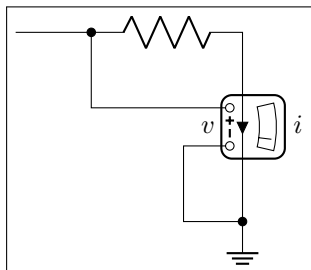


```

1 \begin{circuitikz}[american]
2   \draw (0,0) -- ++(1,0) to[R] ++(2,0)
3     to [qiprobe, l=$i$] ++(0,-2) node[ground]{};
4   \draw (1,0) to[qvprobe, l=$v$] ++(0,-2)
5     node[ground]{};
6 \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-` :

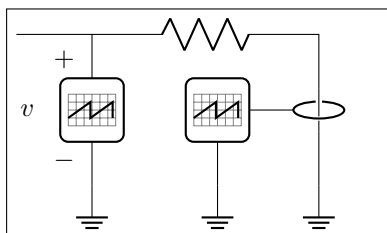


```

1 \begin{circuitikz}[american]
2   \draw (0,0) to[short,-*] ++(1,0) coordinate(b)
3     to [R] ++(2,0) to [qpprobe, l=$i$, a=$v$, name=P]
4     ++(0,-2.5) node[ground] (GND){};
5   \draw (P.v-) -| ++(-0.5,-1) coordinate(a)
6     to [short,-*] (a|GND);
7   \draw (P.v+) -| (b);
8 \end{circuitikz}

```

The final possibility is to use oscilloscopes. For example:



```

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

```

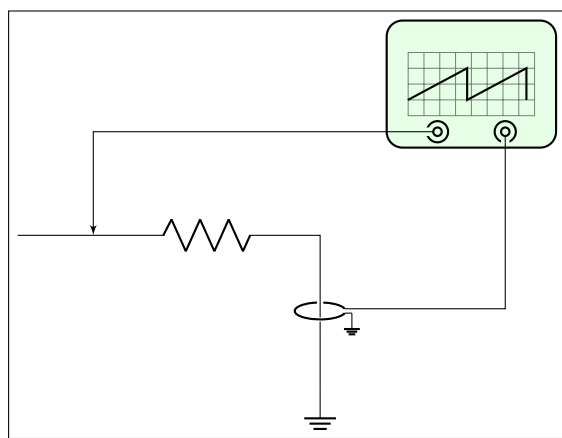
³⁰QUCS is an open source circuit simulator: <http://qucs.sourceforge.net/>

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

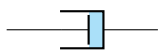
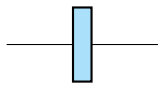

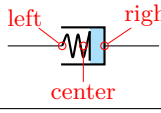

```

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

```



4.7 Mechanical Analogy

	damper: Mechanical Damping, type: path-style, fillable, nodename: dampershape. Class: mechanicals.
	inerter: Mechanical Inerter, type: path-style, fillable, nodename: inertershape. Class: mechanicals.
	spring: Mechanical Stiffness, type: path-style, nodename: springshape. Class: mechanicals.
	viscoe: Mechanical viscoelastic element ³¹ , type: path-style, fillable, nodename: viscoeshape. Class: mechanicals.
	mass: Mechanical Mass, type: path-style, fillable, nodename: massshape. Class: mechanicals.




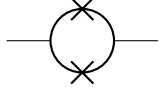

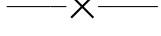
³¹Suggested by @Alex in <https://tex.stackexchange.com/q/484268/38080>

4.7.1 Mechanical elements customizations


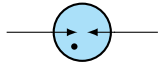
You can change the scale of all the mechanical elements by setting the key `mechanicals/scale` to something different from the default 1.0.

4.8 Miscellaneous bipoles

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

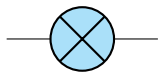
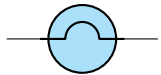
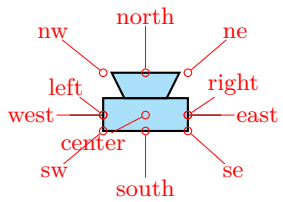
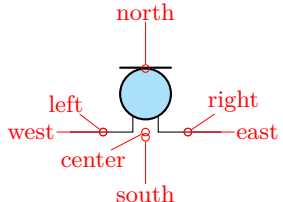
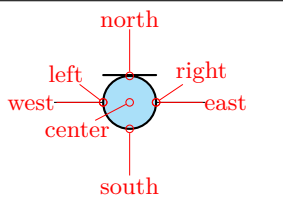
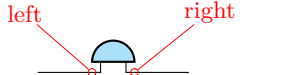
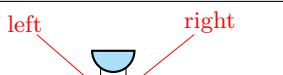
	thermocouple: Thermocouple, type: path-style, nodename: thermocoupleshape. Class: misc.
	fuse: Fuse, type: path-style, fillable, nodename: fuseshape. Class: misc.
	afuse: Asymmetric fuse, type: path-style, fillable, nodename: afuseshape. Aliases: asymmetric fuse. Class: misc.
	squid: Squid, type: path-style,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

You can tune how big is the gap in the `openbarrier` component by setting the key `bipoles/openbarrier/gap` (default value 0.5; 0 means no gap and 1 full gap).

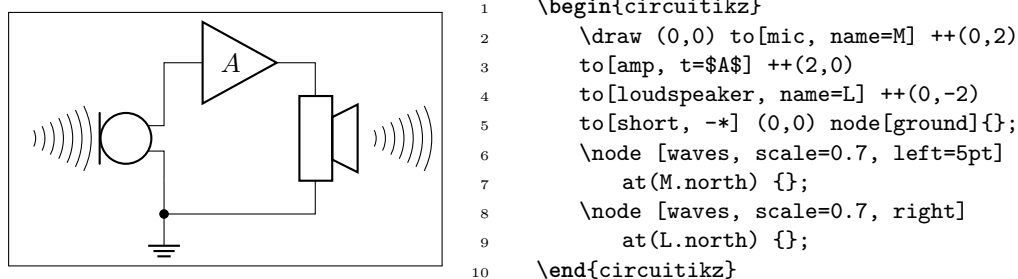
	european gas filled surge arrester: European gas filled surge arrester, type: path-style, fillable, nodename: european gas filled surge arrestershape. Class: misc.
	american gas filled surge arrester: American gas filled surge arrester, type: path-style, fillable, nodename: american gas filled surge 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 `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.
	tlmic: tail-less mic ³² , type: path-style, fillable, nodename: tlmicshape. Class: misc.
	buzzer: Buzzer ³³ , type: path-style, fillable, nodename: buzzershape. Class: misc.
	rbuzzer: Reversed buzzer, type: path-style, fillable, nodename: rbuzzershape. Class: misc.

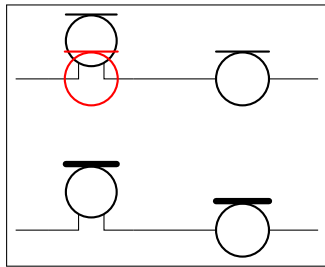
You can use microphones and loudspeakers with waves (see section 4.17) too:



You have two types of microphones; mic has protruding connection and tlmic (for tail-less microphone) is inline. This last one is handy for use as a separate shape (which is named tlmicshape). You can change the (relative) thickness of the straight bar using the key bipoles/mic/bar thickness (default 1).

³²Suggested by [Dr. Matthias Jung](#).

³³Buzzers were suggested by [user Michael.H on TeX.SX](#)



```

1 \begin{circuitikz}[]
2   \draw (0,2) to[mic, name=M] ++(2,0) to[tlmic] ++(2,0);
3   \node [color=red, tlmicshape](T) at (M.center) {};
4   \ctikzset{bipoles/mic/bar thickness=3}
5   \draw (0,0) to[mic] ++(2,0) to[tlmic] ++(2,0);
6 \end{circuitikz}

```

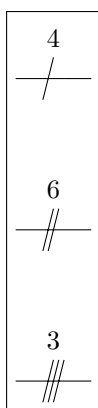
4.8.1 Miscellaneous element customization

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

4.9 Multiple wires (buses)

These are simple drawings to indicate multiple wires.

	multiwire : Single line multiple wires, type : path-style, nodename : multiwireshape. Aliases: multiwire. Class: default.
	bmultiwire : Double line multiple wires, type : path-style, nodename : bmultiwireshape. Aliases: bmultiwire. Class: default.
	tmultiwire : Triple line multiple wires ³⁴ , type : path-style, nodename : tmultiwireshape. Aliases: tmultiwire. Class: default.



```


1 \begin{circuitikz}
2   \draw (0,0) to[multiwire=4] ++(1,0);
3   \draw (0,-2) to[bmultiwire=6] ++(1,0);
4   \draw (0,-4) to[tmultiwire=3] ++(1,0);
5 \end{circuitikz}

```



³⁴added by olfline

4.10 Crossings

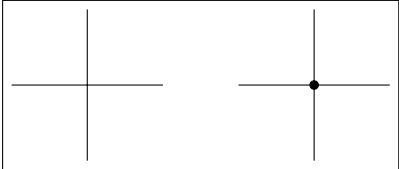
Path style:

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

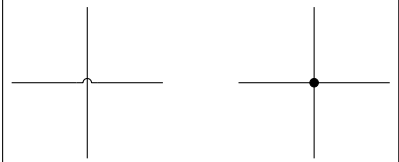
Node style:

	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.

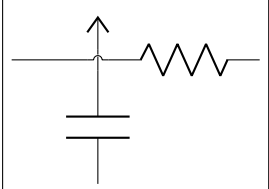
	<pre> 1 \begin{circuitikz}[] 2 \draw(1,-1) to[short] (1,1) 3 (0,0) to[short] (2,0); 4 \draw(4,-1) to[short] (4,1) 5 (3,0) to[short] (5,0) 6 (4,0) node[circ]{}; 7 \end{circuitikz} </pre>
--	---

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

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

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 sometimes a bit of trial and error is needed to position it.

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

	<pre> 1 \begin{circuitikz}[] 2 \node at (1,1)[jump crossing](X){}; 3 \draw (X.west) -- ++(-1,0); 4 \draw (X.east) to[R] ++(2,0); 5 \draw (X.north) node[vcc]{}; 6 \draw (X.south) to[C] ++(0,-1.5); 7 \end{circuitikz} </pre>
---	---

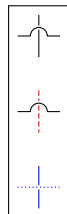
Notice that the **plain crossing** and the **jump crossing** have a small gap in the straight wire, to enhance the effect of crossing (as a kind of shadow).

4.10.1 Crossing customization

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

While the horizontal line will be drawn with the current path values, you can change the style of the vertical line³⁵ in a similar way to the one used for transistor's bodydiodes, by setting keys with the `\tikzset` command under the `crossing vertical` hierarchy. The available keys are:

parameter	default	description
<code>relative thickness</code>	1.0	multiply the default thickness (which is the same of the <code>choke</code> component).
<code>color</code>	default	stroke color: <code>default</code> is the same as the component.
<code>dash</code>	default	dash pattern: <code>default</code> means not to change the setting for the component; <code>none</code> means unbroken line; every other input is a dash pattern. ³⁶





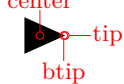
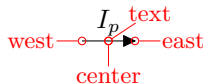
```

1 \begin{circuitikz}[every node/.append style={scale=2}]
2   \draw (0,2) node[jump crossing] (A){};
3   \begin{scope}
4     \tikzset{crossing vertical/.cd, color=red, dash={{2pt}{1pt}}}
5     \draw (0,1) node[jump crossing] (B){};
6   \end{scope}
7   \tikzset{crossing vertical/dash=none}
8   \draw[densely dotted, blue] (0,0) node[plain crossing] (B){};
9 \end{circuitikz}

```

4.11 Arrows

These are pseudo-arrows used in lot of places in the packages (for transistors, flows, currents, and so on). The first three arrows are magnified by a factor 3 in the boxes below; for the `trarrow`, the anchor `tip` is exactly on the tip and `btip` is slightly receded.

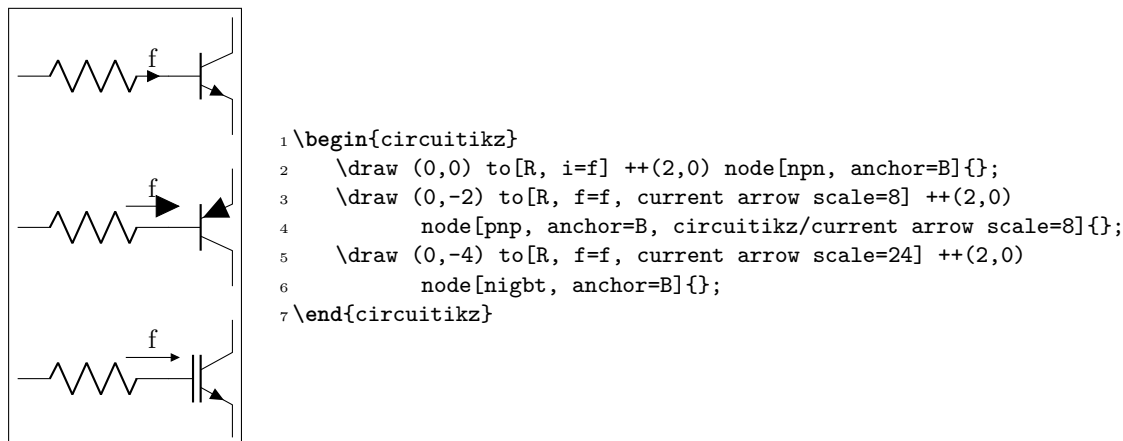
 <code>center</code>	Arrow for current and voltage, type: node (<code>node[currarrow]{}</code>). No class.
 <code>center</code>	Arrow that is anchored at its tip, useful for block diagrams., type: node (<code>node[inputarrow]{}</code>). No class.
 <code>center</code>	Arrow the same size of <code>currarrow</code> but only filled., type: node (<code>node[trarrow]{}</code>). No class.
 <code>center</code>	Arrow used for the flows, with a <code>text</code> anchor, type: node (<code>node[flowarrow]{\$I_p\$}</code>). No class.

³⁵Suggested by [user lkjell on GitHub](#), implemented in v1.6.2.

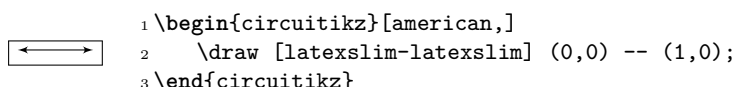
³⁶Follows the syntax of the pattern sequence `\pgfsetdash` — see *TikZ manual* for details; phase is always zero. Basically you pass pairs of dash-length – blank-length dimensions, see the examples.

4.11.1 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.



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



4.11.2 Generic Tunable Arrows

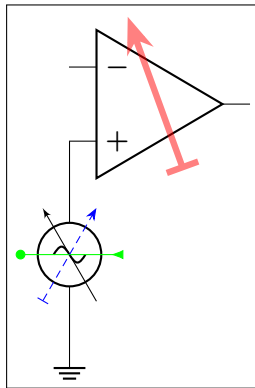
The basic passive components (resistors, capacitors and inductors) come with a “tunable version” (see for example 4.2.3.3) that conveys the information that their value is adjustable. For generic components you can obtain a similar effect with the extra macro `\ctikztunablearrow`, introduced in version 1.4.1. The macro should be called as:

`\ctikztunablearrow[extra options]{thickness}{length}{angle}{name}`

where *extra options* is an optional argument with generic TikZ keys, *thickness* is the relative thickness (referred to the current line width when the macro is invoked), *length* is the length of the arrow with respect to the diagonal size of the component, *angle* is the inclination with respect to the normal direction of the component³⁷, and finally *name* is the reference name of the bipole or node.

The arrows are the ones set with the keys `tunable start arrow` and `tunable end arrow` (to maintain coherency across the circuit), but you can override them in the *extra options* argument as shown in the following example.

³⁷which is the left-to-right direction of the component when shown in the component box in this manual.



```

1 \begin{circuitikz}
2   \draw (0,0) node[tlground]{} to[sV, name=A] ++(0,3)
3     node[op amp, anchor=+](B){};
4   \ctikztunablearrow{1}{1.2}{30}{A}
5   \ctikzset{tunable start arrow={Bar},
6     tunable end arrow={Stealth}}
7   \ctikztunablearrow[color=green,
8     {Latex[reversed]}-Circle}{1}{1.2}{90}{A}
9   \ctikztunablearrow[color=blue, densely dashed]{1}{1.2}{-30}{A}
10  \begin{scope}[transparency group, opacity=0.5]
11    \ctikztunablearrow[red, shorten <=3mm]{6}{0.8}{110}{B}
12  \end{scope}
13 \end{circuitikz}

```

Notice also the need to force a transparency group if you want a semitransparent arrow.

4.12 Terminal shapes

These are the so-called “bipole nodes” shapes, or poles (see section 6.1). These nodes are always filled; the “open” versions (starting with an o) are by default filled with the color specified by the key `open nodes fill` (by default `white`), but you can override locally it with the `fill` parameter.

•	Connected terminal, type: node (<code>node[circ]</code>). No class.
◦	Unconnected terminal, type: node (<code>node[ocirc]</code>). No class.
◆	Diamond-square terminal, type: node (<code>node[diamondpole]</code>). No class.
◇	Open diamond-square terminal, type: node (<code>node[odiamondpole]</code>). No class.
■	Square-shape terminal, type: node (<code>node[squarepole]</code>). No class.
□	Open square-shape terminal, type: node (<code>node[osquarepole]</code>). No class.

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

```

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

```

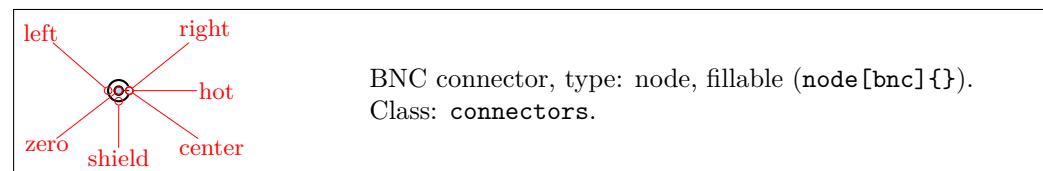


The size of the poles is controlled by the key `nodes width` (default 0.04, relative to the basic length). Be sure to see section 6.1 for more usage and configurability.

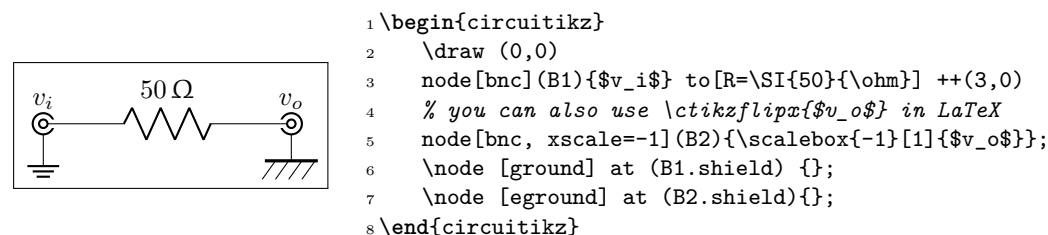
4.13 Connectors

Connectors have a class by themselves (`connectors`), so you can use the `scale`, `fill` and `thickness` properties as usual.

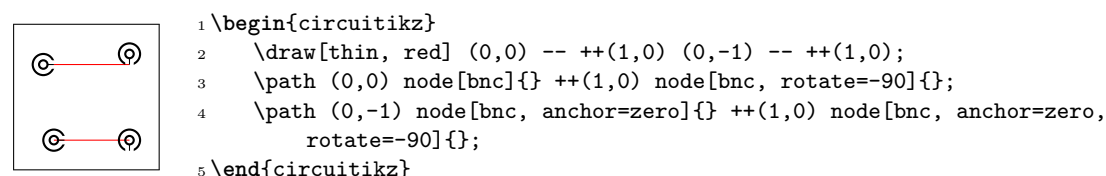
4.13.1 BNC connector/terminal



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):



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



4.13.2 IEC 60617 socket-plug connectors

Plug and socket connectors (modeled on the IEC60617 standard) are available³⁸ both in path-style form and, with separated but matching shapes for plug and socket, in node-style. There are two differently oriented shapes for each type to ease the construction of “split” connections (see the examples below). **Notice** that the elements in the following table are scaled by a factor of 1.5, to better show the position of the anchors.

	<p>iec connector: IEC 60617 connector, type: path-style, nodename: <code>ieconnshape</code>. Aliases: <code>ieconn</code>. Class: <code>connectors</code>.</p>
	<p>IEC 60617 connector, type: node (<code>node[ieconnshape]{\tiny text}</code>). Class: <code>connectors</code>.</p>
	<p>IEC 60617 female socket, left side, type: node (<code>node[iecsocketL]{\tiny text}</code>). Class: <code>connectors</code>.</p>

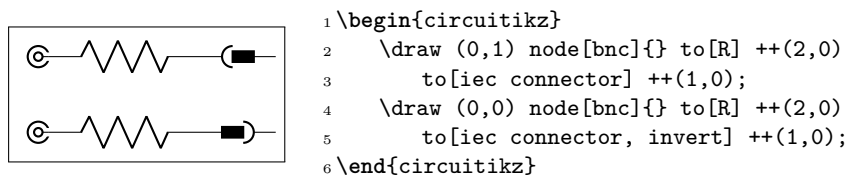
³⁸Since v1.5.0; thanks to Alexander Sauter for suggesting them and [helping in the design](#).

	IEC 60617 male plug, right side, type: node (node[iecpugR]{\tiny text}). Class: connectors.
	IEC 60617 male plug, left side, type: node (node[iecpugL]{\tiny text}). Class: connectors.
	IEC 60617 female socket, right side, type: node (node[iecsscketR]{\tiny text}). Class: connectors.

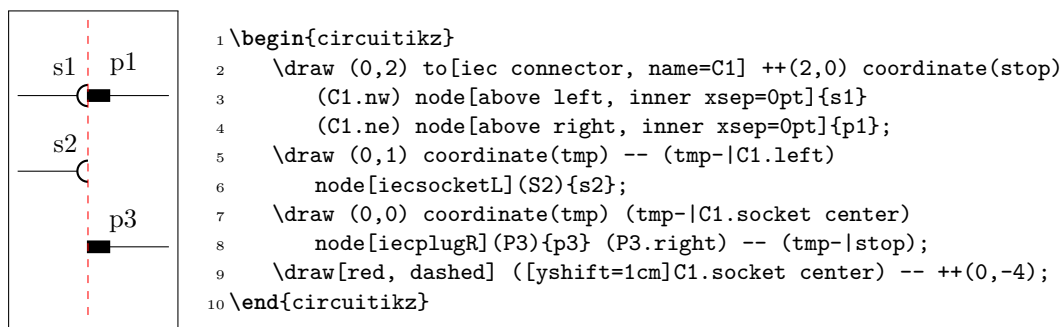
The **center** anchors (as well as the text position) of the split elements of the connectors are on the side of the component (similar to what happens with grounds and supply voltage arrows) to ease the most common use.

Also, the text for the plug nodes is raised to the same level of the text in the sockets, and it will ignore descendants, so that the two text lines up when the two components are put side by side.

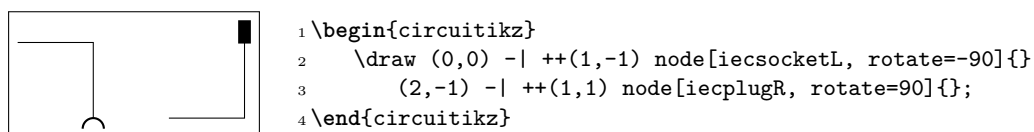
The **plug center** anchor always point to the center of the rectangular plug shape, and the **socket center** to the center of the semicircle in the sockets.



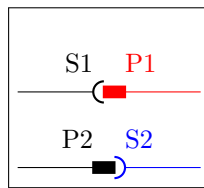
Aligning “disconnected” plugs and sockets is reasonably easy:



You can choose the best shape when rotating them, to simplify the positioning (shape rotates around the **center** anchor).



Choosing the proper left/right shape results in easily build “mixed” connectors; you can use the node text position properties to have lined-up labels, but remember that the text is outside the bounding box:

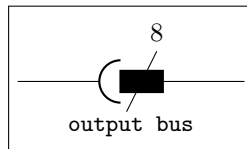


```

1 \begin{circuitikz}[]
2   \path (0,2); % for the bounding box, text is not accounted for
3   \draw (0,1)---++(1,0) node[iecssocketL](s1){S1};
4   \draw [color=red](s1.e) node[iecplugR](p1){P1} (p1.e)---++(1,0);
5   \draw (0,0) -- ++(1,0) node[iecplugL](p2){P2};
6   \draw [color=blue](p2.e) node[iecssocketR](s2){S2} (s2.e)---++(1,0);
7 \end{circuitikz}

```

You can use the `plug center` anchor to add the IEC “multiplier”:



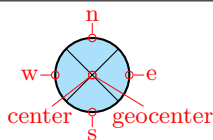
```

1 \begin{circuitikz}[]
2   \draw (0,0) to[iec connector, connectors/scale=2, name=A,
3     a={\small\ttfamily output bus}] ++(3,0);
4   \draw (A.plug center) ++(-.2,-.4) -- ++(.4,.8) node[above]{8};
5 \end{circuitikz}

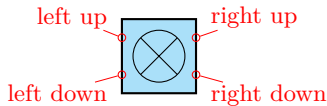
```

4.14 Block diagram components

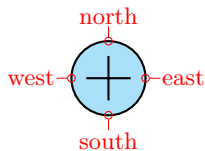
Contributed by Stefan Erhardt.



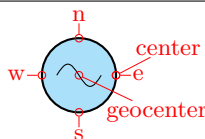
mixer, type: node, fillable (`node[mixer]{}`). Class: `blocks`.



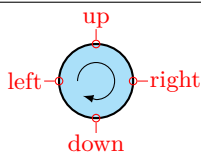
mixer, boxed, type: node, fillable (`node[mixer, boxed]{}`). Class: `blocks`.



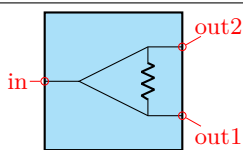
adder, type: node, fillable (`node[adder]{}`). Class: `blocks`.



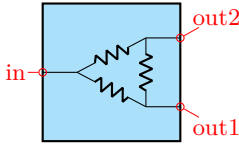
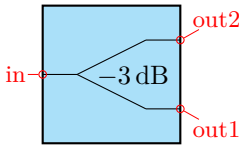
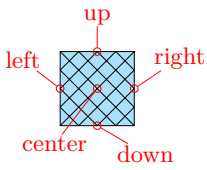
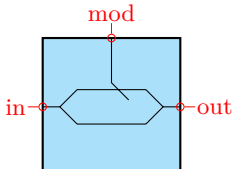
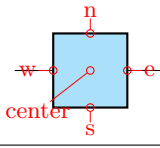
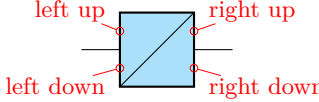

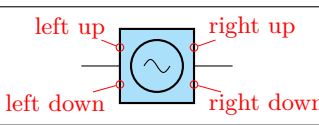
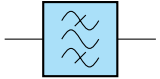

oscillator, type: node, fillable (`node[oscillator]{}`). Class: `blocks`.



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



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
















	resistive splitter ³⁹ , type: node, fillable (node[splitter]{}). Class: blocks.
	generic splitter ⁴⁰ , type: node, fillable (node[genericsplitter]{ $\text{\SI{-3}{\deci\bel}}$ }{}). Class: blocks.
	gridnode ⁴¹ , type: node, fillable (node[gridnode]{}). Class: blocks.
	Mach Zehnder Modulator ⁴² , type: node, fillable (node[mzm]{}). Class: blocks.
	twoport : generic two port (use t=... to specify text), type: path-style, fillable, nodename: twoportshape. Class: blocks.
	twoportsplit : generic two port split (use t1=... and t2=... to specify text), type: path-style, fillable, nodename: twoportsplitshape. Class: blocks.
	vco : vco, type: path-style, fillable, nodename: vcoshape. Class: blocks.
	vco,box : vco,box, type: path-style, fillable, nodename: vco,boxshape. Class: blocks.
	bandpass : bandpass, type: path-style, fillable, nodename: bandpassshape. Class: blocks.
	bandstop : bandstop, type: path-style, fillable, nodename: bandstopshape. Class: blocks.

³⁹added by matthuszagh

⁴⁰added by frankplow

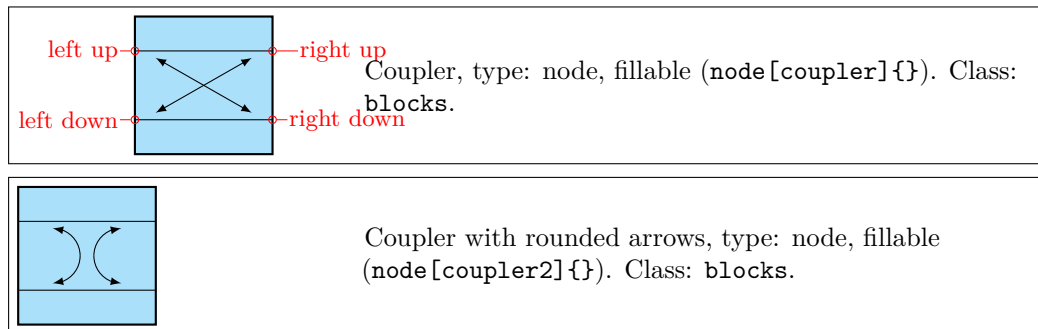
⁴¹added by olflne

⁴²added by dl1chb

	highpass: highpass, type: path-style, fillable, nodename: highpassshape. Class: blocks.
	lowpass: lowpass, type: path-style, fillable, nodename: lowpassshape. Class: blocks.
	allpass: allpass, type: path-style, fillable, nodename: allpassshape. Class: blocks.
	highpass2: simplified highpass (with only 2 waves), type: path-style, fillable, nodename: highpass2shape. Class: blocks.
	lowpass2: simplified lowpass (with only 2 waves), type: path-style, fillable, nodename: lowpass2shape. 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.

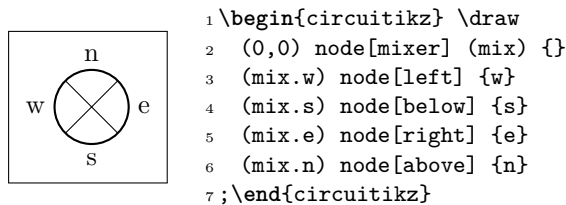
	phaseshifter : phase shifter, type: path-style, fillable, nodename: phaseshiftershape. Class: blocks.
	vphaseshifter : var. phase shifter, type: path-style, fillable, nodename: vphaseshiftershape. Class: blocks.
	detector : detector, type: path-style, fillable, nodename: detectorshape. Class: blocks.
	sdcdc : single wire DC/DC converter ⁴³ , type: path-style, fillable, nodename: sdcdcshape. Class: blocks.
	sacdc : single phase AC/DC converter, type: path-style, fillable, nodename: sacdcshape. Class: blocks.
	sdcac : single phase DC/AC converter, type: path-style, fillable, nodename: sdcacshape. Class: blocks.
	sacac : single phase AC/AC converter, type: path-style, fillable, nodename: sacacshape. Class: blocks.
	tacdc : three phases AC/DC converter, type: path-style, fillable, nodename: tacdcshape. Class: blocks.
	tdcac : three phases AC/DC converter, type: path-style, fillable, nodename: tdcacshape. Class: blocks.
	tacac : three phases AC/DC converter, type: path-style, fillable, nodename: tacacshape. Class: blocks.
	Generic fourport, type: node, fillable (node[fourport]{}). Class: blocks.

⁴³the converter blocks added by offline

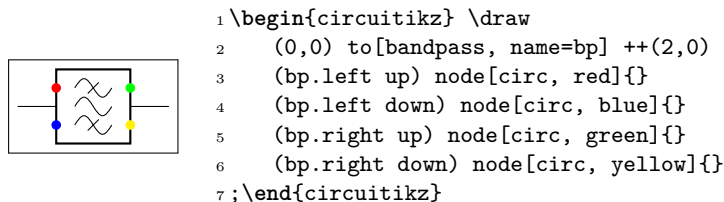


4.14.1 Blocks anchors

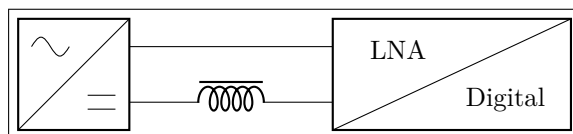
The ports of the mixer, adder, oscillator and circulator can be addressed with `west`, `south`, `east`, `north`; the equivalent `left`, `down`, `right`, `up`; or the shorter `w`, `s`, `e`, `n` ones:



In addition, since v1.6.0, most blocks also have the `left up`, `left down`, `right up` and `right down` anchors:



You can use those anchors to build “mixed-type” circuits, positioning the node-shapes:



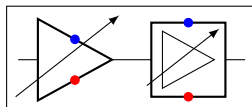
```

1 \begin{tikzpicture}[
2   big/.style={circuitikz/blocks/scale=1.5},
3   long/.style={circuitikz/bipoles/twoportsplit/width=1.5}]
4   \path (0,0) node[sacdcshape, big] (A){}
5     (5,0) node[twoportsplitshape, big, long, t1=LNA, t2=Digital] (B){};
6   \draw (A.right up) -- (B.left up) (A.right down) to[cute choke] (B.left down);
7 \end{tikzpicture}

```

Notice also from the previous example that the generic blocks (`twoport` and `twoportsplit`) can be made “longer” by setting different `width` and `height` (the other blocks are square, and just use the `width` key for both dimensions).

Also, for `amp` and `vamp`, the `up` and `down` anchors follow the shape when they are not boxed.

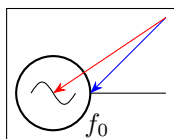


```

1 \begin{tikzpicture}
2   \draw (0,0) to[vamp, name=a] ++(1.5,0)
3     to [vamp, boxed, name=ab] ++(1.5,0);
4   \path (a.up) node[circ, blue]{} (ab.up) node[circ, blue]{};
5   \path (a.down) node[circ, red]{} (ab.down) node[circ, red]{};
6 \end{tikzpicture}

```

The `oscillator` has a displaced `center` anchor, to simplify the task of putting it at the left side of a circuit; it also has a special position for the node text. The four round elements (mixer, circulator, adder, and the oscillator) have a `geocenter` anchor which always corresponds to the center of the circle.

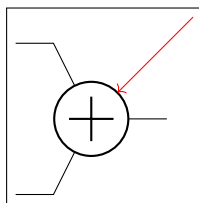


```

1 \begin{tikzpicture}[>=Stealth]
2   \draw (0,0) node[oscillator] (0) {\$f_0\$} -- ++(1,0);
3   \draw[blue, ->] (1,1) -- (0.center);
4   \draw[red, ->] (1,1) -- (0.geocenter);
5 \end{tikzpicture}

```

Moreover, they have proper border anchors since version 1.2.3 (and fixed for boxed elements in 1.5.0), so you can do things like this:

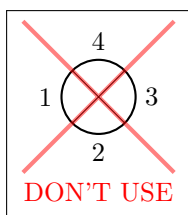


```

1 \begin{circuitikz}
2   \draw (0,0) node[adder] (mix) {}
3     (-1,1) -- ++(0.5,0) -- (mix)
4     (-1,-1) -- ++(0.5,0) -- (mix) -- ++(1,0);
5   \draw [red, <-] (mix.45) -- ++(1,1);
6 \end{circuitikz}

```

Those components have also **deprecated** anchors named 1, 2, 3, 4; they are better not used because they can conflict with the border anchor. They still work for backward compatibility, but could be removed in a future release.

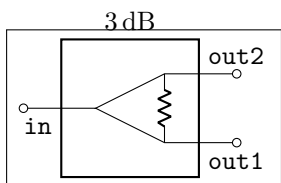


```

1 \begin{circuitikz} \draw
2   (0,0) node[mixer] (mix) {}
3   (mix.1) node[left] {1} (mix.2) node[below] {2}
4   (mix.3) node[right] {3} (mix.4) node[above] {4};
5 \draw [ultra thick, red, opacity=0.5]
6   (-1,-1) -- (1,1) (-1,1) -- (1,-1);
7 \node [red, below] at (0,-1) {DON'T USE};
8 \end{circuitikz}

```

The Wilkinson divider has (notice that the node text is outside the bounding box, similarly to what happens for transistors!):

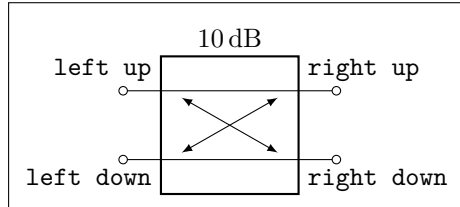


```

1 \begin{circuitikz} \draw
2   (0,0) node[wilkinson] (w) {\SI{3}{dB}}
3   (w.in) to[short,-o] ++(-0.5,0)
4   (w.out1) to[short,-o] ++(0.5,0)
5   (w.out2) to[short,-o] ++(0.5,0)
6   (w.in) node[below left] {\texttt{in}}
7   (w.out1) node[below right] {\texttt{out1}}
8   (w.out2) node[above right] {\texttt{out2}}
9   ;
10 \end{circuitikz}

```

The couplers have:

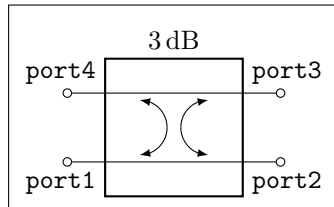


```

1 \begin{circuitikz} \draw (0,1.5) %bounding box
2 (0,0) node[coupler] (c) {\SI{10}{dB}}
3 (c.left down) to[short,-o] ++(-0.5,0)
4 (c.right down) to[short,-o] ++(0.5,0)
5 (c.right up) to[short,-o] ++(0.5,0)
6 (c.left up) to[short,-o] ++(-0.5,0)
7 (c.left down) node[below left] {\texttt{left
   down}}
8 (c.right down) node[below right] {\texttt{right
   down}}
9 (c.right up) node[above right] {\texttt{right
   up}}
10 (c.left up) node[above left] {\texttt{left up}}
11 ;
12 \end{circuitikz}

```

Or you can also use port1 to port4 if you prefer:

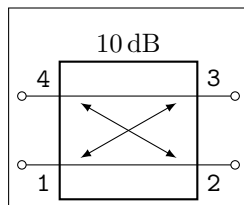


```

1 \begin{circuitikz} \draw (0,1.5) %bounding box
2 (0,0) node[coupler2] (c) {\SI{3}{dB}}
3 (c.port1) to[short,-o] ++(-0.5,0)
4 (c.port2) to[short,-o] ++(0.5,0)
5 (c.port3) to[short,-o] ++(0.5,0)
6 (c.port4) to[short,-o] ++(-0.5,0)
7 (c.port1) node[below left] {\texttt{port1}}
8 (c.port2) node[below right] {\texttt{port2}}
9 (c.port3) node[above right] {\texttt{port3}}
10 (c.port4) node[above left] {\texttt{port4}}
11 ;
12 \end{circuitikz}

```

Also they have the simpler 1, 2, 3, 4 anchors, and although they have no border anchors (for now), it is better not to use them.



```

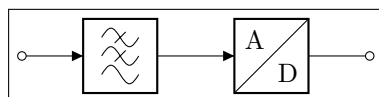
1 \begin{circuitikz} \draw(0,1.5) %bounding box
2 (0,0) node[coupler] (c) {\SI{10}{dB}}
3 (c.1) to[short,-o] ++(-0.5,0)
4 (c.2) to[short,-o] ++(0.5,0)
5 (c.3) to[short,-o] ++(0.5,0)
6 (c.4) to[short,-o] ++(-0.5,0)
7 (c.1) node[below left] {\texttt{1}}
8 (c.2) node[below right] {\texttt{2}}
9 (c.3) node[above right] {\texttt{3}}
10 (c.4) node[above left] {\texttt{4}}
11 ;
12 \end{circuitikz}

```

4.14.2 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.

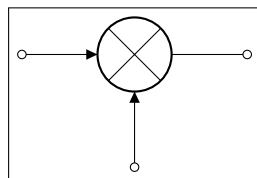


```

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

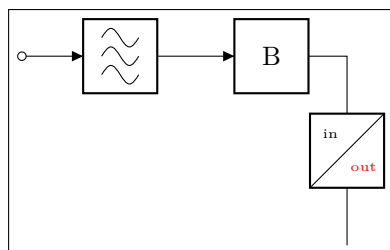
```

4.14.2.1 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:



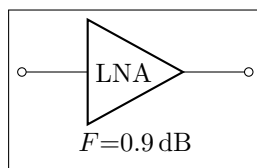
```
1 \begin{circuitikz} \draw
2 (0,0) node[mixer] (m) {}
3 (m.w) to[short,-o] ++(-1,0)
4 (m.s) to[short,-o] ++(0,-1)
5 (m.e) to[short,-o] ++(1,0)
6 (m.w) node[inputarrow] {}
7 (m.s) node[inputarrow,rotate=90] {};
8 \end{circuitikz}
```

4.14.2.2 Labels and custom two-port boxes You can use the keys `t`, `t1`, `t2` (shorthands for `text`, `text in`, `text out`) to fill the generic blocks:



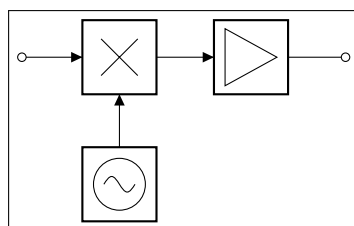
```
1 \begin{circuitikz} \draw
2 (0,0) to[short,o-] ++(0.3,0)
3 to[allpass,>] ++(2,0)
4 to[twoport,>,t={B}] ++(2,0)
5 to[twoportsplit,t1={\tiny in},
6 t2={\tiny\color{red}out}] ++(0,-2.5);
7 \end{circuitikz}
```

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



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

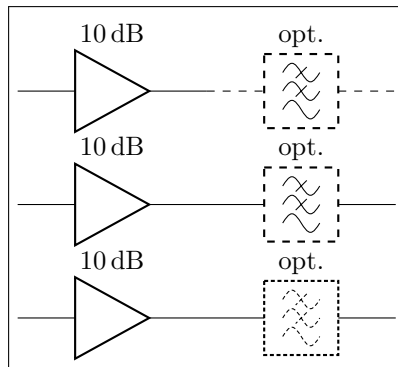
4.14.2.3 Box option Several devices have the possibility to add a box around them with the `box` or `boxed` option. The inner symbol scales down to fit inside the box. For the “circled” devices (mixer, adder, oscillator and circulator) the inner circle is normally drawn, unless you use the `box only` or `boxed only` option.⁴⁴



```
1 \begin{circuitikz} \draw
2 (0,0) node[mixer, box only, anchor=east] (m) {}
3 to[amp, boxed, >, -o] ++(2.5,0)
4 (m.west) node[inputarrow]{} to[short,-o] ++(-0.8,0)
5 (m.south) node[inputarrow,rotate=90]{} --
6 ++(0,-0.7) node[oscillator, box, anchor=north]{};
7 \end{circuitikz}
```

4.14.2.4 Dash optional parts To show that a device is optional, you can dash it. The inner symbol will be kept with solid lines, unless you set the key `inner blocks dashed` to true. Moreover, the key `dashed blocks pattern` (default `{\{1mm\}{1mm\}}`), be careful with the number of braces!.

⁴⁴Since 1.5.0, suggested by [GitHub user myzinsky](#)

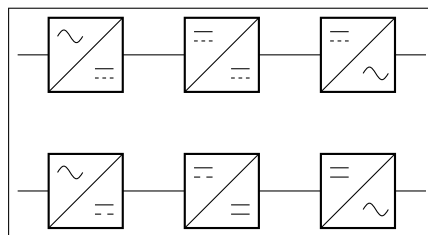


```

1 \begin{circuitikz}
2 \draw (0,1.5) to[amp,l=\SI{10}{dB}] ++(2.5,0);
3 \draw[dashed] (2.5,1.5) to[lowpass,l=opt.] ++(2.5,0);
4 % or just the block
5 \draw (0,0) to[amp,l=\SI{10}{dB}] ++(2.5,0)
6   to[lowpass,l=opt., dashed] ++(2.5,0);
7 % or everything
8 \ctikzset{inner blocks dashed,
9   dashed blocks pattern={\1.5pt}{1pt}},
10 }
11 \draw (0,-1.5) to[amp,l=\SI{10}{dB}] ++(2.5,0)
12   to[lowpass,l=opt., dashed] ++(2.5,0);
13 \end{circuitikz}

```

4.14.2.5 Dashing the DC symbol in blocks. The symbol for the DC side can be different across countries,⁴⁵ with different kind of dashing on the bottom line. Moreover, sometimes the dashing is used to convey different meanings (like rectified sinusoidal or stabilized DC). You can change the general style for the DC symbol using the key `blocks dc segments`; using 1 (default) will use a continuous line; using 2 you will have the international-styled symbol, and with 3 the English one (you can use higher numbers; the only restriction is that it must be strictly greater than 0). You can also change the input and output part separately with the keys `blocks dc in segments` and `blocks dc out segments` (see the following example). The inner blocks `dashed` option overrides these ones.



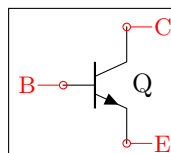
```

1 \begin{tikzpicture}[scale=0.9]
2   \ctikzset{blocks dc segments=3}
3   \draw (0,2) to[sacdc] ++(2,0) to[sdcac]
4     ++(2,0) to[sdcac] ++(2,0);
5   \ctikzset{blocks dc segments=1}
6   \draw (0,0) to[sacdc, blocks dc out segments=2]
7     ++(2,0) to[sdcac, blocks dc in segments=2]
8     ++(2,0) to[sdcac] ++(2,0);
9 \end{tikzpicture}

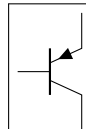
```

4.15 Transistors

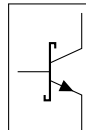
4.15.1 Standard bipolar transistors



nnp, type: node (node[nnp]{Q}). Class: transistors.

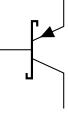
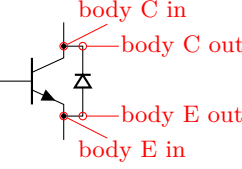
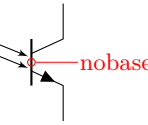
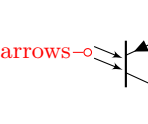
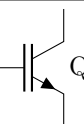
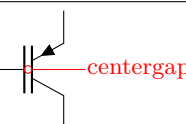
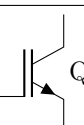
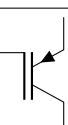
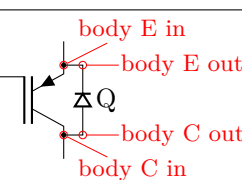


pnp, type: node (node[pnp]{}). Class: transistors.



schottky npn, type: node (node[nnp, schottky base]{}). Class: transistors.

⁴⁵Head-up from [user @dbstf on GitHub](#)

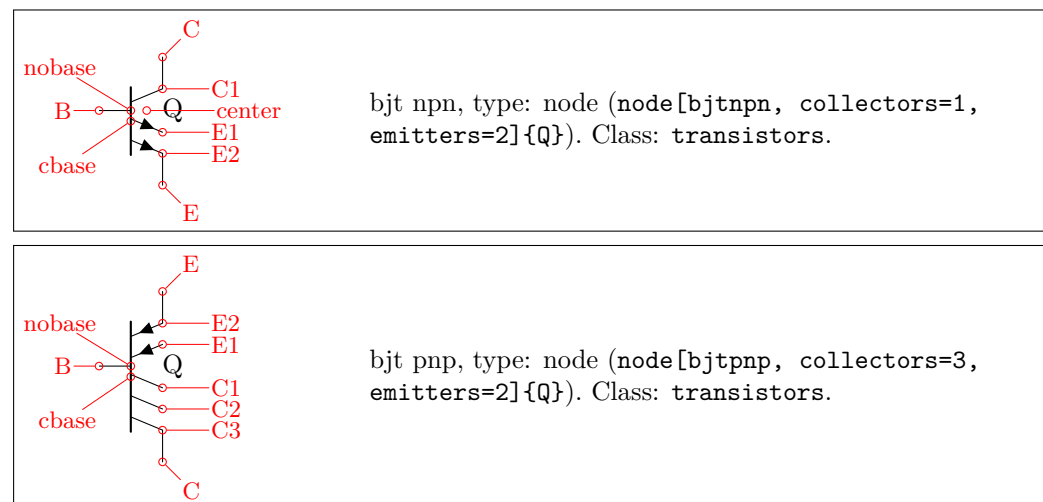
	schottky pnp, type: node (node[pnp, schottky base]{}). Class: transistors.
	npn, type: node (node[npn, bodydiode]{}). Class: transistors.
	photo npn, type: node (node[npn,photo]{}). Class: transistors.
	photo pnp, type: node (node[pnp,photo]{}). Class: transistors.
	nigt, type: node (node[nigt]{Q}). Class: transistors.
	pigbt, type: node (node[pigbt]{}). Class: transistors.
	Lnigt, type: node (node[Lnigt]{Q}). Class: transistors.
	Lpigbt, type: node (node[Lpigbt]{}). Class: transistors.
	Lpigbt, type: node (node[Lpigbt, bodydiode]{Q}). Class: transistors.

4.15.2 Multi-terminal bipolar transistors

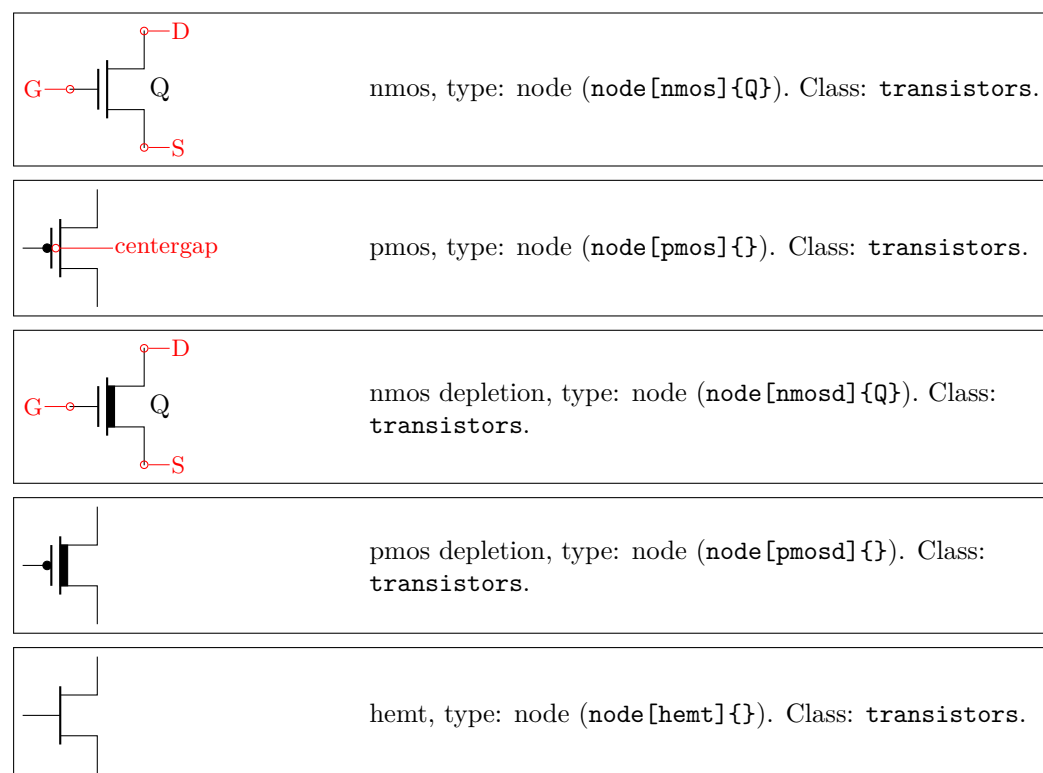
In addition to the standard BJTs transistors, since version 0.9.6 the `bjtnpn` and `bjtpnp` are also available; these are devices where you can have more collectors and emitters (on the other hand, they have no `photo` nor `bodydiode` options — they are silently ignored).

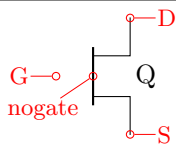
Basically they are the same as the normal `npn` and `pnp`, and they (by default) have similar sizes; the options `collectors` and `emitters` will change the number of the relative terminals. The base

terminal is connected midway from the collector and the emitter, *not* on the center of the base; a **cbase** anchor is available if you prefer to use it. The label of the component (the text) is set on the right side, vertically centered around the base terminal. They will accept the **schottky base** key.

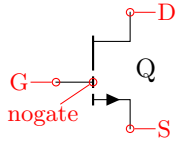


4.15.3 Field-effect transistors



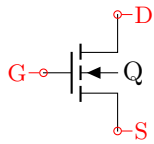


hemt without base terminal, type: node (node[hemt, nobase]{Q}). Class: transistors.

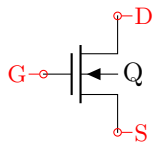


Gallium Nitride hemt (a “styled” hemt, see 4.15.5.5), type: node (node[GaN hemt]{Q}). Class: transistors.

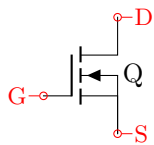
NFETs and PFETs have been incorporated based on code provided by Clemens Helfmeier and Theodor Borsche. Use the package options `fetsolderdot/nofetsolderdot` to enable/disable solderdot at some fet-transistors. Additionally, the `solderdot` option can be enabled/disabled for single transistors with the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



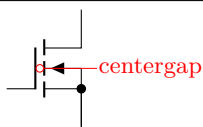
nfet, type: node (node[nfet]{Q}). Class: transistors.



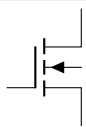
nfet depletion, type: node (node[nfetd]{Q}). Class: transistors.



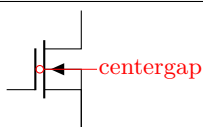
nigfete, type: node (node[nigfete]{Q}). Class: transistors.



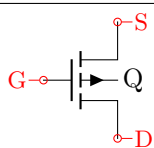
nigfete, type: node (node[nigfete,solderdot]{}). Class: transistors.



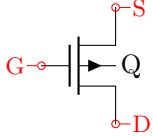
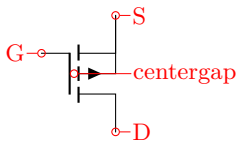
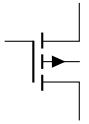
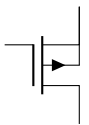
nigfetebulk, type: node (node[nigfetebulk]{}). Class: transistors.



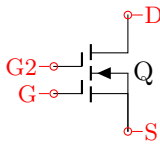
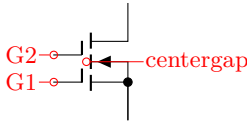
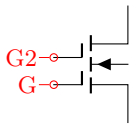
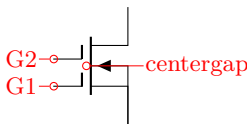
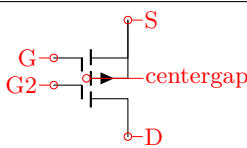
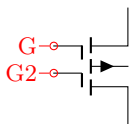
nigfetd, type: node (node[nigfetd]{}). Class: transistors.

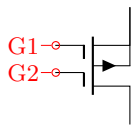


pfet, type: node (node[pfet]{Q}). Class: transistors.

	pfet depletion, type: node (node[pfetd]{Q}). Class: transistors.
	pigfete, type: node (node[pigfete]{}). Class: transistors.
	pigfetebulk, type: node (node[pigfetebulk]{}). Class: transistors.
	pigfetd, type: node (node[pigfetd]{}). Class: transistors.

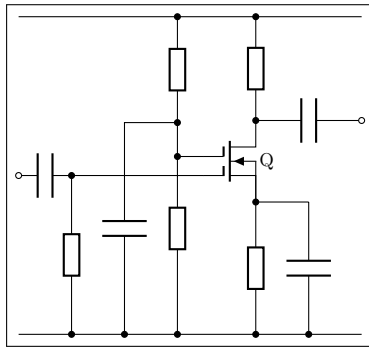
Since version 1.6.0, you can add the `doublegate` option to the `*igfet*` family of devices to have a double gate MOS — the additional gate is called `G2` or `gate2` (the plain `G` is where it will be without the `doublegate` option).

	nigfete, doublegate, type: node (node[nigfete, doublegate]{Q}). Class: transistors.
	nigfete, doublegate, type: node (node[nigfete,solderdot, doublegate]{}). Class: transistors.
	nigfetebulk, doublegate, type: node (node[nigfetebulk, doublegate]{}). Class: transistors.
	nigfetd, doublegate, type: node (node[nigfetd, doublegate]{}). Class: transistors.
	pigfete, doublegate, type: node (node[pigfete, doublegate]{}). Class: transistors.
	pigfetebulk, doublegate, type: node (node[pigfetebulk, doublegate]{}). Class: transistors.



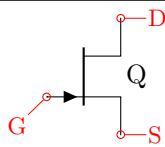
pigfetd, doublegate, type: node (node[pigfetd, doublegate]{}). Class: transistors.

You can use the double-gated transistor for example like this:⁴⁶

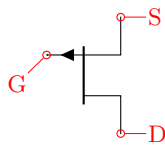


```
1 \begin{circuitikz}[european, scale=0.7, transform shape,
2   circuitikz/resistors/scale=0.7]
3   \draw (0,0) to[C, o-] ++(1,0) coordinate(in)
4   to[R, -] ++(0,-3) coordinate(GND)
5   ++(1,0) to[C, *-] ++(0,4) -- ++(1,0) coordinate(div)
6   to[R, *-] ++(0,2) coordinate(vdd)
7   (div) to[R, -] (div|-GND)
8   (in) -- ++(2.5,0) node[nigfetd, doublegate, anchor=G1] (Q){Q}
9   (Q.G2) to[short, -] (Q.G2|-div)
10  (Q.D) to[R, -] (Q.D|-vdd)
11  (Q.D) to[C, *-o] ++(2,0) coordinate(out)
12  (Q.S) to[R, *-] (Q.S|-GND) (Q.S) -- ++(1,0) coordinate(Sd)
13  (Sd) to[C, -] (Sd|-GND);
14  \draw (0,0|-GND) -- (out|-GND) (0,0|-vdd) -- (out|-vdd);
15 \end{circuitikz}
```

JFET are also available⁴⁷, both n-type and p-type.

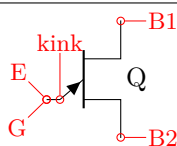


n-type JFET, type: node (node[njfet]{Q}). Class: transistors.

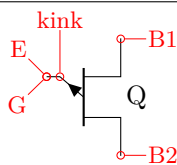


p-type JFET, type: node (node[pjfet]{}). Class: transistors.

UJT transistors⁴⁸ have a different anchor names although **most** of the others, like D and G, also work (the exception is E and **emitter**!). Notice that if used with **nobase**, the anchor E follows the wire, while G is fixed (as is kink).



n-type UJT, type: node (node[nujt]{Q}). Class: transistors.

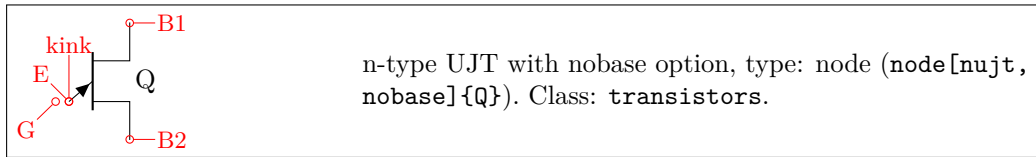


p-type UJT, type: node (node[pujt]{Q}). Class: transistors.

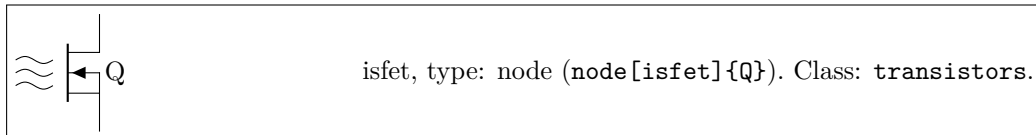
⁴⁶Found at [this page on electronics notes](#).

⁴⁷based on code provided by Danilo Piazzalunga

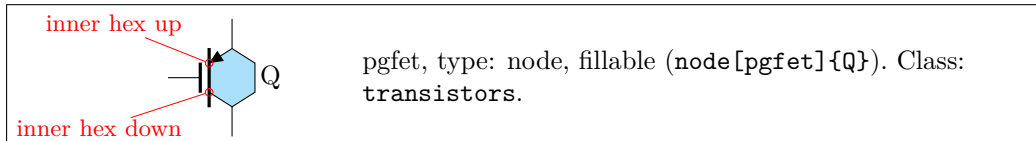
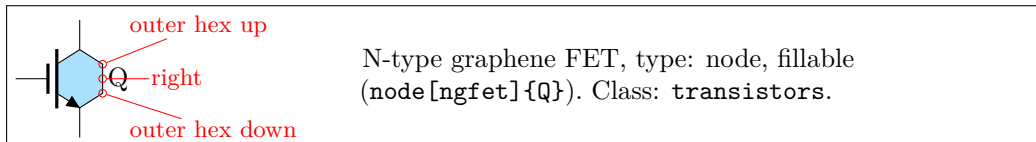
⁴⁸suggested by [user JetherReis on GitHub](#).



ISFET:



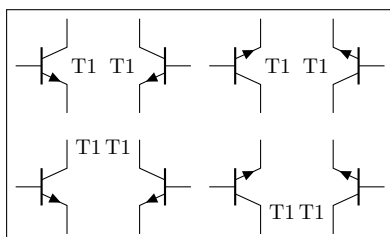
GRAPHENE FET have been added in version 1.3.2⁴⁹. They look better if you set `transistors/arrow pos=end` and `transistor/thickness=3` or higher for them, so they are plotted with this option here.



4.15.4 Transistor texts (labels)

In versions before 0.9.7, transistors text (the node text) was positioned near the collector terminal; since version 0.9.7 the default has been changed to a more natural position near the center of the device, similar to the multi-terminal transistors. You can revert to the old behavior locally with the key `legacy transistors text`, or globally by setting the package option `legacytransistorstext`.

Notice the use of the utility functions `\ctikzflip{x,y,xy}` as explained in section 3.2.1.



```

1 \begin{circuitikz}[scale=0.8, transform shape]
2   \draw (0,0) node [npn]{T1}
3     ++(1.2,0) node [npn, xscale=-1]{\ctikzflipx{T1}}
4     ++(2,0) node [npn, yscale=-1]{\ctikzflipy{T1}}
5     ++(1.2,0) node [npn, scale=-1]{\ctikzflipxy{T1}};
6   \ctikzset{legacy transistors text}
7   \draw (0,-2) node [npn]{T1}
8     ++(1.2,0) node [npn, xscale=-1]{\ctikzflipx{T1}}
9     ++(2,0) node [npn, yscale=-1]{\ctikzflipy{T1}}
10    ++(1.2,0) node [npn, scale=-1]{\ctikzflipxy{T1}};
11 \end{circuitikz}

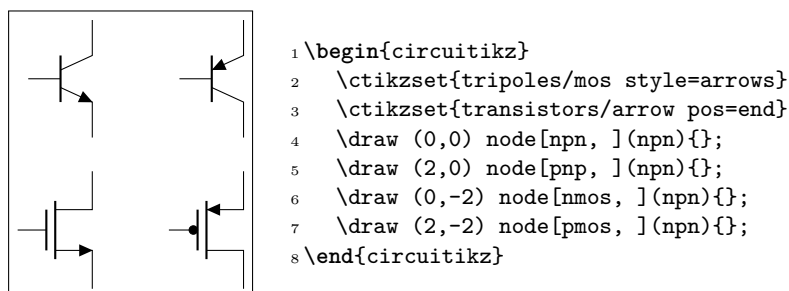
```

⁴⁹added by Romano Giannetti after a suggestion by Cees Keyer.

4.15.5 Transistors customization

4.15.5.1 Size. 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 `curarrow` (see section 4.11.1) and the dots on P-type transistors (if any) are the same as the nodes/poles (see section 6.1).

4.15.5.2 Arrows. 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`).



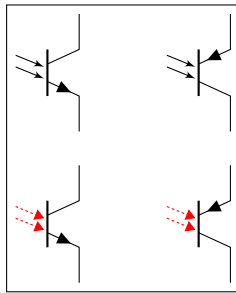
If the option `arrowmos` is used (or after the command `\ctikzset{tripoles/mos style/arrows}` is given), this is the output:

	nmos, type: node (node[nmos]{}). Class: transistors.
	pmos, type: node (node[pmos]{}). Class: transistors.
	nmos depletion, type: node (node[nmosd]{}). Class: transistors.
	pmos depletion, type: node (node[pmosd]{}). Class: transistors.

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

You can also change⁵⁰ the type of the arrow for the “light rays” of the phototransistors with the generic arrows options shown in 4.2.3.3, using the name `opto`, like in the following (overdone) example. Also the `opto arrows` styling options (see section 4.4.3.1).

⁵⁰Thanks to the idea by [Dr. Matthias Jung on GitHub](#).



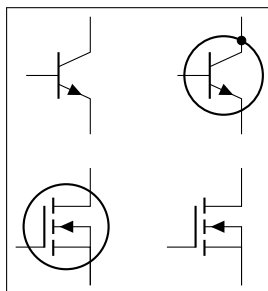
```

1 \begin{tikzpicture}
2   \draw (0,2) node[npn, photo]{} ++(2,0) node[pnp, photo]{};
3   \ctikzset{opto end arrow={Triangle[angle'=60]}}
4   \ctikzset{opto arrows/.cd, color=red, dash={{1pt}{1pt}}}
5   \draw (0,0) node[npn, photo]{} ++(2,0) node[pnp, photo]{};
6 \end{tikzpicture}

```

4.15.5.3 Circles. Since 1.2.6, you can add a circle⁵¹ to most of the transistor shapes — with the exception of multi-terminal (`bjtnpn` and `bjtpnp`, where it would be awkward anyway) and graphene FETs. The circle is intended in some case as the component's housing, and used to distinguish discrete components from integrated ones.

To add the circle to a single transistor, you use the `tr circle` keys in the node; if you want all of your transistors with a circle, you can set the property `tr circle` with a `\ctikzset` command (it will respect normal grouping, of course); in that case, you can use `tr circle=false` to locally disable them.



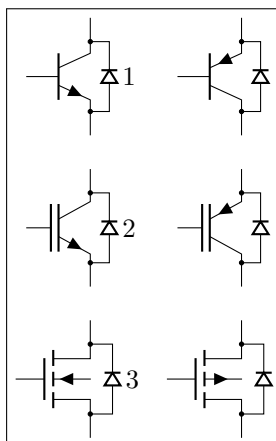
```

1 \begin{circuitikz}[]
2   \draw (0,2) node[npn]{} (2,2) node[npn, tr circle](Q){};
3   % collector connected to housing
4   \node [circ] at (Q.circle C){};
5   \ctikzset{tr circle=true} % or \ctikzset{tr circle} alone
6   \draw (0,0) node[nigtfet]{}
7         (2,0) node[nigtfet, tr circle=false]{};
8 \end{circuitikz}

```

You can tweak the appearance of transistor's circles and even draw it partially; see 4.15.7 for details.

4.15.5.4 Body diodes and similar things. For all transistors (minus `bjtnpn` and `bjtpnp`) a body diode (or freewheeling or flyback diode) can automatically be drawn. Just use the global option `bodydiode`, or for single transistors, the TikZ-option `bodydiode`. As you can see in the next example, the text for the diode is moved if a `bodydiode` is present (but beware, if you change a lot the relative dimension of components, it may become misplaced):



```

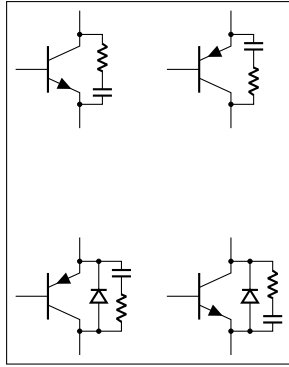
1 \begin{circuitikz}
2   \draw (0,0) node[npn,bodydiode](npn){1}
3         ++(2,0)node[pnp,bodydiode](pnp){};
4   \draw (0,-2) node[nigtbt,bodydiode](nigtbt){2}
5         ++(2,0)node[pigbt,bodydiode](pigbt){};
6   \draw (0,-4) node[nfet,bodydiode](nfet){3}
7         ++(2,0)node[pfet,bodydiode](pfet){};
8 \end{circuitikz}

```

⁵¹Suggested by Matthias Jung on [GitHub](#)

You can tweak the appearance of transistor's bodydiodes; see 4.15.8.

For more complex snubs or protections, you can use the `body ...` anchors to add more or different things to the transistors in addition (or instead) of the flyback diode.



```

1 \def\snubb#1#2{% add a snubber to a transistor
2   \draw (#1.body C #2) to[short, *, nodes width=0.02]
3   ++(0.3,0) coordinate(tmp) to [R, resistors/scale=0.3]
4   % 2/3 space for R, 1/3 for C
5   ($ (tmp)!0.66!(tmp|-#1.body E #2)$)
6   to [C, capacitors/scale=0.3] (tmp|-#1.body E #2)
7   to [short, -, nodes width=0.02] (#1.body E #2);
8 }
9 \begin{circuitikz}
10  \node[npn] (Q1) at(0,0) {};
11  \node[pnp] (Q2) at(2,0) {};
12  \node[pnp, bodydiode] (Q3) at(0,-3) {};
13  \node[npn, bodydiode] (Q4) at(2,-3) {};
14  \snubb{Q1}{in} \snubb{Q2}{in}
15  \snubb{Q3}{out} \snubb{Q4}{out}
16 \end{circuitikz}

```

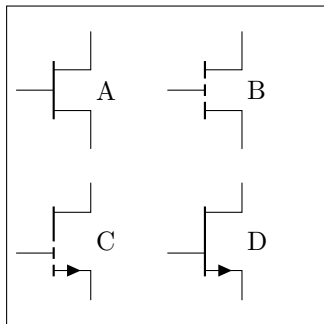
4.15.5.5 HEMT customization. Since v1.6.1 the shape of the `hemt` transistor can be customized.⁵² There are several keys under the hierarchy `tripoles/hemt` that can be used to change the appearance; some of them are common to other transistors and some are specific.

The main ones are `.../split gate` (boolean, default `false`) that will create a “split” gate, which sometimes is used to convey that the device is enhancement-type; `source arrow` (default 0), to add an arrow on the source terminal (1 for a right-facing one, -1 the other way around; the arrow will obey `arrow pos=end` if issued, but otherwise the position is fixed); `gate asym` (default 0.0) which displaces the gate asymmetrically. For example, the `GaN hemt` component is really a styled `hemt` (predefined), with the definition:

```

1 \tikzset{GaN hemt/.style={hemt,
2   circuitikz/tripoles/hemt/base height=0.6,% length of the "base" vertical bar
3   circuitikz/tripoles/hemt/gate height=0.5,% distance of the S/D terminals
4   circuitikz/tripoles/hemt/bodydiode conn=0.85,% attachment point of body diode
5   circuitikz/tripoles/hemt/gate asym=-0.1,% slightly down
6   circuitikz/tripoles/hemt/split gate=true,% split gate
7   circuitikz/tripoles/hemt/source arrow=1,% right-facing arrow
8 },
9 }

```



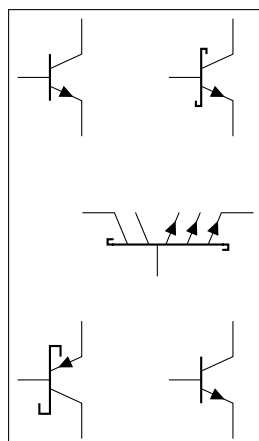
```

1 \begin{circuitikz}[]
2   \path (-1,-1) rectangle (3,3);% bounding box
3   \node [hemt] at (0,2) {A};
4   \node [hemt,
5     circuitikz/tripoles/hemt/split gate=true
6   ] at (2,2) {B};
7   \node [GaN hemt] at (0,0) {C};
8   \node [GaN hemt,
9     circuitikz/tripoles/hemt/split gate=false
10  ] at (2,0) {D};
11 \end{circuitikz}

```

⁵²After a suggestion from user @epsilon-phi on GitHub.

4.15.5.6 Schottky transistors. The Schottky transistors are generated by adding the `schottky base` key (there is also a `no schottky base` key that can be used if you use the other one as a default). You can change the size of the Schottky “hook” changing the parameter `tripoles/schottky base size` with `\ctikzset{}` (default 0.05; the unit is the standard resistor length, scaled if needed.)



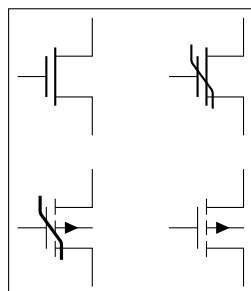
```

1 \begin{circuitikz}
2   \draw (0,4) node[npn]{}
3     ++(2,0) node[npn, schottky base]{};
4   \draw (1,2) node[bjtnpn, collectors=2, emitters=3,
5     schottky base, rotate=90]{};
6   \tikzset{schottky base}
7   \ctikzset{tripoles/schottky base size=0.1}
8   \draw (0,0) node[pnp]{}
9     ++(2,0) node[npn, no schottky base]{};
10 \end{circuitikz}

```

4.15.5.7 Ferroelectric transistors You can add the ferroelectric modifier⁵³ to the `*mos` and `*fet` transistor types. Similarly to the Schottky bipolar transistors, you activate it by adding the `ferroel gate` key (there is also a `no ferroel base` key that can be used if you use the other one as a default).

The mark will follow the `transistors` class thickness, but you can adjust it independently using the class parameter `modifier thickness` as in passive components — this value is relative to the class’ thickness.



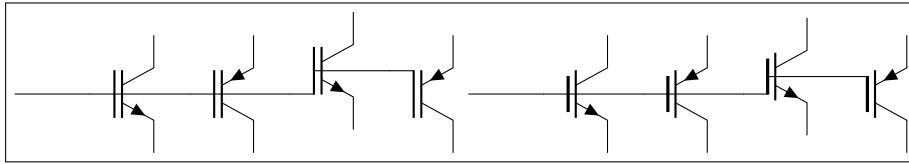
```

1 \begin{circuitikz}
2   \draw (0,2) node[nmos]{}
3     ++(2,0) node[nmos, ferroel gate]{};
4   \ctikzset{ferroel gate} % by default from now on
5   \ctikzset{transistors/.cd, % class properties
6     thickness=1, modifier thickness=3}
7   \draw (0,0) node[pfet]{}
8     ++(2,0) node[pfet, no ferroel gate]{};
9 \end{circuitikz}

```

4.15.5.8 IGBT outer base. Normally, in bipolar IGBTs the outer base is the same size (height) of the inner one, and of the same thickness (which will depend on the class thickness value). You can change this by setting (via `\ctikzset`) the keys `tripoles/igbt/outer base height` (default 0.4, the same as base height), and `tripoles/igbt/outer base thickness` (default 1.0), which will be relative to the class thickness.

⁵³suggested by [Mayeul Cantan](#)

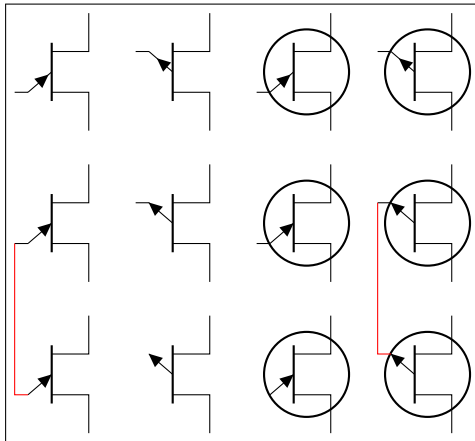


```

1 \begin{circuitikz}
2   \draw (0,0)
3     -- ++(1,0) node[nigbt, anchor=B] (B){} (B.nobase)
4     -- ++(1,0) node[pigbt, anchor=B] (B){} (B.nobase)
5     -- ++(1,0) node[Lnigbt, anchor=B] (B){} (B.nobase)
6     -- ++(1,0) node[Lpigbt, anchor=B] (B){} (B.nobase)
7   ;
8   \ctikzset{tripoles/igbt/outer base height=0.3}
9   \ctikzset{tripoles/igbt/outer base thickness=1.5}
10  \draw (6,0)
11    -- ++(1,0) node[nigbt, anchor=B] (B){} (B.nobase)
12    -- ++(1,0) node[pigbt, anchor=B] (B){} (B.nobase)
13    -- ++(1,0) node[Lnigbt, anchor=B] (B){} (B.nobase)
14    -- ++(1,0) node[Lpigbt, anchor=B] (B){} (B.nobase)
15  ;
16 \end{circuitikz}

```

4.15.5.9 UJT transistors. They look better if you use `transistors/arrow pos=end`, especially if you use them with `tr circle`. If you use the key `nobase` with UJTs, the horizontal part of the controlling terminal is not drawn; notice that this *will* move the E or emitter anchor, but not the generic ones like G.

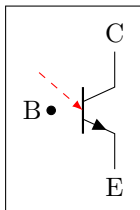


```

1 \begin{circuitikz}[scale=0.8]
2   \draw (0,5) node[nujt]{} ++(2,0) node[pujt]{}
3     ++(2,0) node[nujt, tr circle]{} ++(2,0)
4     node[pujt, tr circle]{};
5   \ctikzset{transistors/arrow pos=end}
6   \draw (0,2.5) node[nujt] (A){} ++(2,0) node[pujt]{}
7     ++(2,0) node[nujt, tr circle]{} ++(2,0)
8     node[pujt, tr circle] (C){};
9   \draw (0,0) node[nujt, nobase] (B){} ++(2,0)
10    node[pujt, nobase]{} ++(2,0)
11    node[nujt, tr circle, nobase]{} ++(2,0)
12    node[pujt, tr circle, nobase] (D){};
13   % "E" anchor follows the nobase option:
14   \draw[red] (A.E) |- (B.E) (C.E) |- (D.E);
15 \end{circuitikz}

```

4.15.5.10 Base/Gate terminal. 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:

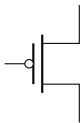
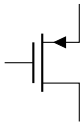


```

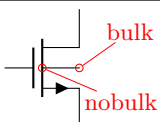
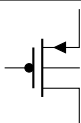
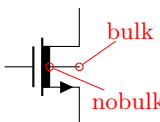
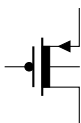
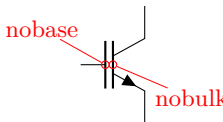
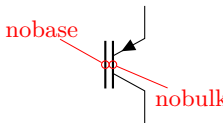
1 \begin{circuitikz}
2   \draw (2,0) node[npn,nobase] (npn){};
3   \draw (npn.E) node[below]{E};
4   \draw (npn.C) node[above]{C};
5   \draw (npn.B) node[circ]{} node[left]{B};
6   \draw[dashed,red,-latex] (1,0.5)--(npn.nobase);
7 \end{circuitikz}

```

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}`.

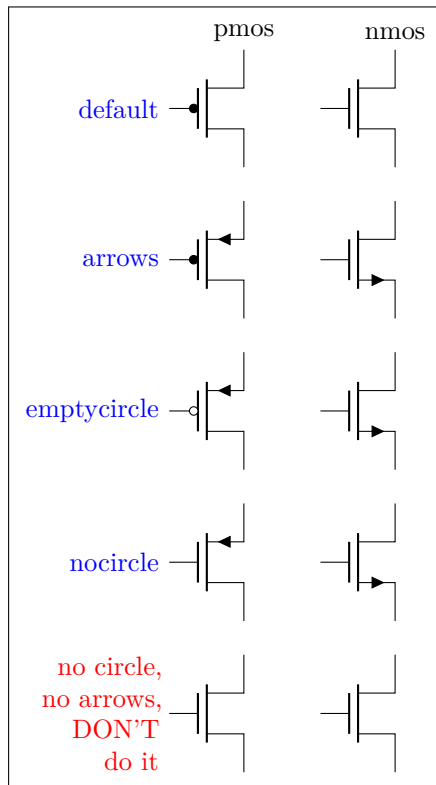
	<code>pmos, type: node (node[pmos,emptycircle]{}).</code> Class: <code>transistors</code> .
	<code>pmos, type: node (node[pmos,nocircle,arrowmos]{}).</code> Class: <code>transistors</code> .

4.15.5.11 Bulk terminals. You can add a bulk terminal⁵⁴ to `nmos` and `pmos` using the key `bulk` in the node (and `nobulk` if you set the bulk terminal by default); additional anchors `bulk` and `nobulk` are added (in the next example, `tripoles/mos style/arrows` is enacted, too):

	<code>pmos, type: node (node[nmos, bulk]{}).</code> Class: <code>transistors</code> .
	<code>pmos, type: node (node[pmos, bulk]{}).</code> Class: <code>transistors</code> .
	<code>nmos depletion, type: node (node[nmosd, bulk]{}).</code> Class: <code>transistors</code> .
	<code>pmos depletion, type: node (node[pmosd, bulk]{}).</code> Class: <code>transistors</code> .
	<code>night, type: node (node[night]{}).</code> Class: <code>transistors</code> .
	<code>pigbt with no base terminal⁵⁵, type: node (node[pigbt, nobase]{}).</code> Class: <code>transistors</code> .

⁵⁴Thanks to Burak Kelleci <kellecib@hotmail.com>.

⁵⁵Since v1.4.4, noticed by user [hinata](#) exc on Stack Exchange.



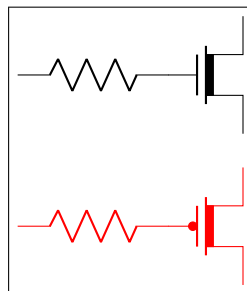
```

1 \begin{circuitikz}[
2   info/.style={left=1cm, blue, text width=5em,
3     align=right},]
4   \draw (0,1) node{pmos} (2,1) node{nmos};
5   \draw (0,0) node[info]{default} node[pmos]{}
6     (2,0) node[nmos]{};
7   \ctikzset{tripoles/mos style/arrows}
8   \draw (0,-2) node[info]{arrows} node[pmos]{}
9     (2,-2) node[nmos]{};
10  \ctikzset{tripoles/pmos style/emptycircle}
11  \draw (0,-4) node[info]{emptycircle} node[pmos]{}
12    (2,-4) node[nmos]{};
13  \ctikzset{tripoles/pmos style/nocircle}
14  \draw (0,-6) node[info]{nocircle} node[pmos]{}
15    (2,-6) node[nmos]{};
16  \ctikzset{tripoles/mos style/no arrows}
17  \draw (0,-8) node[info, red]{no circle, no
18    arrows, DON'T do it}
19    node[pmos]{} (2,-8) node[nmos]{};
20 \end{circuitikz}

```

4.15.5.12 Simplified symbols for depletion-mode MOSFETs . The `nmosd`, `pmosd` (simplified) symbols for depletion-mode MOSFET (introduced in 1.2.4) behave exactly like the normal (without the final `d`) ones.

By default, the thick bar (indicating the pre-formed channel) is filled with the same color as the drawing:

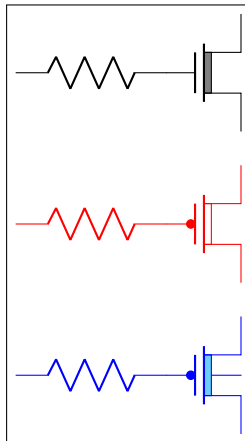


```

1 \begin{circuitikz}[ ]
2   \draw (0,2) to[R] ++(2,0) node[nmosd, anchor=G]{};
3   \draw[color=red] (0,0) to[R] ++(2,0) node[pmosd, anchor=G]{};
4 \end{circuitikz}

```

You can change this behavior by setting the key `tripoles/nmosd/depletion color` (default value `default`, which means “use the draw color”) to the color you want; using `none` will lead to an unfilled channel (note that in this case the color does not change automatically with the path!):

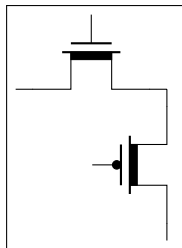


```

1 \begin{circuitikz}[ ]
2   \ctikzset{tripoles/nmosd/depletion color=gray}
3   \draw (0,2) to[R] ++(2,0) node[nmosd, anchor=G]{};
4   \ctikzset{tripoles/pmosd/depletion color=none}
5   \draw[color=red] (0,0) to[R] ++(2,0)
6     node[pmosd, anchor=G]{};
7   \ctikzset{tripoles/pmosd/depletion color=
8     {cyan!50!white}}
9   \draw[color=blue] (0,-2) to[R] ++(2,0)
10    node[pmosd, anchor=G, bulk]{};
11 \end{circuitikz}

```

Obviously you have the equivalent `tripoles/pmosd/depletion color` for type-P transistors. They also have path-style syntax, as the other transistors.

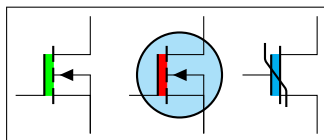


```

1 \begin{circuitikz}[ ]
2   \draw (0,0) to[Tnmosd] ++(2,0)
3     to[Tpmosd, invert] ++(0,-2)
4   ;
5 \end{circuitikz}

```

4.15.5.13 Gate/Base gap coloring. You can color the space representing the gate capacitor or the insulated base by using the key `tr gap fill` (default `none`, which means nothing is drawn there). This fill is done *after* any circle fill but before any additional modifier (see the example below). You can use it locally or set it globally (normal scoping works, as ever).



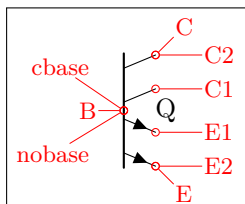
```

1 \begin{tikzpicture}
2   \node[nigfete, tr gap fill=green] at(0,0){};
3   \node[nigfete, tr gap fill=red, tr circle,
4     fill=cyan!30] at(1.5,0){};
5   \node[nmos, tr gap fill=cyan, ferroel gate](A)
6     at(3,0){};
7 \end{tikzpicture}

```

4.15.6 Multiple terminal transistors customization

You can create completely “bare” transistors (without the connection leads to the B, C y E terminals), by changing the parameter `tripoles/bjt/pins width` (default 0.3; it is expressed as a fraction of the basic (scaled) length) or using the style `bjt pins width`; and you can change the distance between multiple collectors/emitters setting with `\ctikzset{}` the parameter `tripoles/bjt/multi height` (default 0.5) or the style `bjt multi height`.

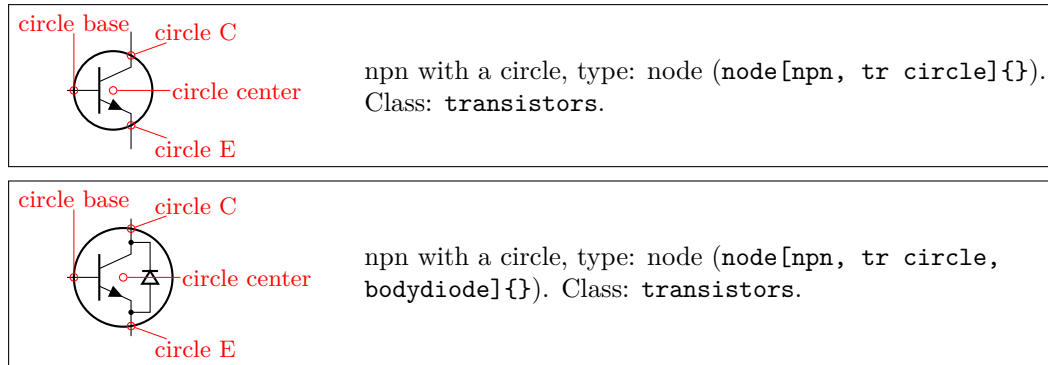


bjt npn with parameters, type: `node (node[bjtnpn, collectors=2, emitters=2, bjt pins width=0, bjt multi height=0.8]{}).` Class: transistors.

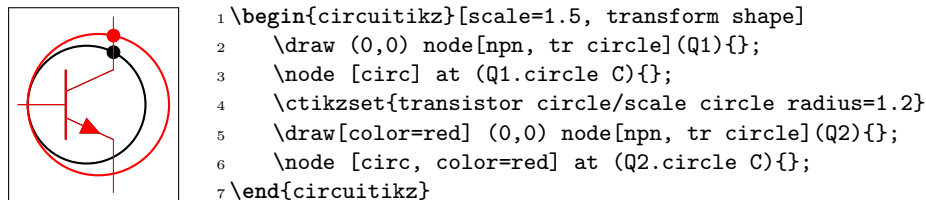
4.15.7 Transistor circle customization

4.15.7.1 Position and size.

You can see in the following diagram where the circle is positioned — when there is no bodydiode, it will pass through the anchors for the body diode and near the base connection. The dimension of the circle is bigger when the bodydiode is in, to encompass it. The anchors are present even if there is no circle, so you can use them to draw different kinds of circles (say, encompassing two transistors) in a coherent way.



The position of the circle on collector and emitter by default is the one shown above; the position along the base can be adjusted in most transistors using the `\ctikzset` parameter `transistor circle/default base in` (by default 0.9); `njfet` and `pjfet` use `transistor circle/njfet base in` (default 1.05; the same for `pjfet`) and, finally, `isfet` uses `transistor circle/isfet base in` (default 0.65). You can change the resulting size of the circle by setting to something different to 1.0 the parameter `transistor circle/scale circle radius` — that will move the anchors too; for example:

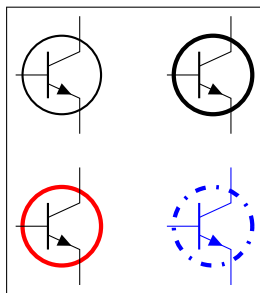


4.15.7.2 Line and color.

Normally the circle follows the style of the component — the line thickness is fixed by the class element `transistors/thickness` and the color is the same as the component color. You can change, if you need, all of these things using the parameters of the following table (the parameters are under the `\ctikzset` category root `transistor circle/`).

parameter	default	description
<code>relative thickness</code>	1.0	multiply the class thickness
<code>color</code>	default	stroke color: default is the same as the component
<code>dash</code>	default	dash pattern: none means solid line, default means keep the global patt
<code>partial borders</code>	none	draw only part of the circle border: none means draw all
<code>partial border dash</code>	<code>{{2pt}}{2pt}}</code>	dash pattern used in partial borders

⁵⁶Follows the syntax of the pattern sequence `\pgfsetdash` — see TikZ manual for details; phase is always zero. Basically you pass pairs of dash-length – blank-length dimensions, see the examples.

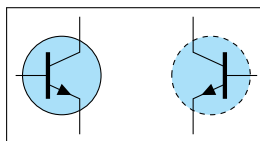


```

1 \begin{circuitikz}[]
2   \draw (0,2) node[npn, tr circle](Q1){};
3   \ctikzset{transistor circle/relative thickness=2}
4   \draw (2,2) node[npn, tr circle](Q1){};
5   \ctikzset{transistor circle/color=red}
6   \draw (0,0) node[npn, tr circle](Q1){};
7   \ctikzset{transistor circle/color=default}
8   \ctikzset{transistor circle/dash={{4pt}{4pt}}{1pt}{4pt}}
9   \draw[color=blue] (2,0) node[npn, tr circle](Q1){};
10 \end{circuitikz}

```

Finally, using the class style you can do quite interesting things.



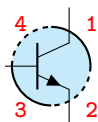
```

1 \begin{circuitikz}[]
2   \ctikzset{transistors/thickness=4, transistors/fill=cyan!30,
3     transistor circle/relative thickness=0.25,}
4   \draw (0,0) node[npn, tr circle](Q1){};
5   \ctikzset{transistor circle/dash={{2pt}{2pt}}{}}
6   \draw (1.5,0) node[npn, tr circle, xscale=-1](Q2){};
7 \end{circuitikz}

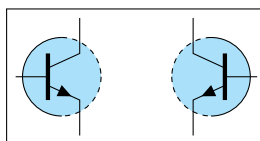
```

4.15.7.3 Partially drawn circle borders In some circuits, transistors are drawn with partial or dashed border (to convey the meaning of several active components encased in the same physical package, or to signify thermal contact). To achieve this effect, you can use the `transistor circle/partial border`⁵⁷ key (default none). This key can be set to none, or must be a sequence of **exactly** 4 numbers, that can have value 0, 1, or 2. Each number defines the style of a part of the border to be not drawn, solid or dashed respectively.

The part of the border are numbered from 1 to 4 as shown below:



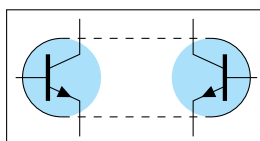
The dashed line pattern can be changed by setting the key `transistor circle/partial border dash` (default `{{2pt}{2pt}}`). Be careful with the extra set of braces here.



```

1 \begin{circuitikz}[]
2   \ctikzset{transistors/thickness=4, transistors/fill=cyan!30,
3     transistor circle/relative thickness=0.25,
4     transistor circle/partial borders=2211}
5   \draw (0,0) node[npn, tr circle](Q1){};
6   \ctikzset{transistor circle/dash={{2pt}{2pt}}{}}
7   \draw (1.5,0) node[npn, tr circle, xscale=-1](Q2){};
8 \end{circuitikz}

```



```

1 \begin{circuitikz}[]
2   \ctikzset{transistors/thickness=4, transistors/fill=cyan!30,
3     transistor circle/relative thickness=0.25,
4     transistor circle/partial borders=0011}
5   \draw (0,0) node[npn, tr circle](Q1){};
6   \ctikzset{transistor circle/dash={{2pt}{2pt}}{}}
7   \draw (1.5,0) node[npn, tr circle, xscale=-1](Q2){};
8   \draw[dashed] (Q1.circle top) -- (Q2.circle top);
9   \draw[dashed] (Q1.circle bottom) -- (Q2.circle bottom);
10 \end{circuitikz}

```

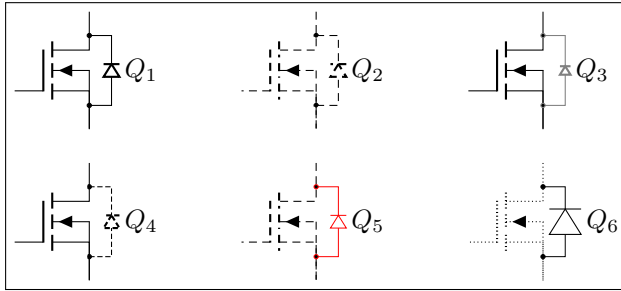
⁵⁷Suggested by [Jether Fernandes Reis](#) for tubes, implemented by Romano in v1.5.2.

4.15.8 Transistor bodydiode customization

You can change the style of the bodydiode⁵⁸ in a similar way to the one used for circles (albeit with less options), by setting keys with the `\ctikzset` command (or, like in the following example, directly in the node instantiation) under the `transistor bodydiode` hierarchy. The available keys are:

parameter	default	description
<code>relative thickness</code>	1.0	multiply the class thickness
<code>color</code>	<code>default</code>	stroke color: <code>default</code> is the same as the component
<code>dash</code>	<code>default</code>	dash pattern: <code>default</code> means not to change the setting for the component; <code>none</code> means unbroken line; every other input is a dash pattern ⁵⁹
<code>scale</code>	0.3	scale of the diode, with respect to the basic (not scaled) diode dimension.

The following is a quite extensive example. Obviously, a good strategy in this case is to define styles for the options, or, better, set the option in your style file or in the preamble (for coherency).



```

1 \begin{tikzpicture}[red solid thin bodydiode/.style={bodydiode,
2   circuitikz/transistor bodydiode/dash=none,
3   circuitikz/transistor bodydiode/color=red,
4   circuitikz/transistor bodydiode/relative thickness=0.3}]
5   \draw (0,0) node (mosfet1) [nigfete,anchor=D,bodydiode] {$Q_1$};
6   \draw[densely dashed] (3,0) node (mosfet1) [nigfete,anchor=D,bodydiode] {$Q_2$};
7   \draw (6,0) node (mosfet1) [nigfete,anchor=D,bodydiode,
8     circuitikz/transistor bodydiode/color=gray,
9     circuitikz/transistor bodydiode/scale=0.2] {$Q_3$};
10  \draw (0,-2) node (mosfet1) [nigfete,anchor=D,bodydiode,
11    circuitikz/transistor bodydiode/dash={{2pt}{1pt}}] {$Q_4$};
12  \draw[densely dashed] (3,-2) node (mosfet1) [nigfete,anchor=D,
13    red solid thin bodydiode] {$Q_5$};
14  \ctikzset{transistor bodydiode/relative thickness=.5,
15    transistor bodydiode/scale=0.6}% from now on, in scope
16  \draw[densely dotted] (6,-2) node (mosfet1) [nigfete,anchor=D,bodydiode,
17    circuitikz/transistor bodydiode/dash=none] {$Q_6$};
18  \path (7,0); %% adjust bounding box (node text is outside it!)
19 \end{tikzpicture}

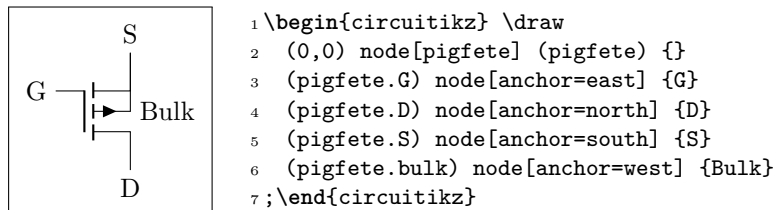
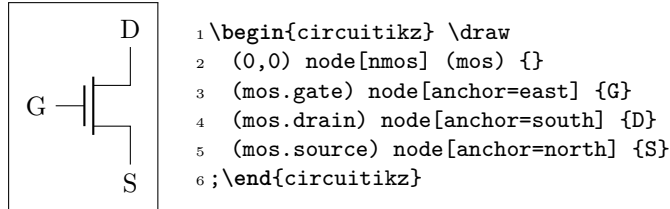
```

⁵⁸Suggested by [user Alex Ghilas on TeX.SX](#), implemented in v1.5.4; scale suggested by [user @sputeanus on GitHub](#) and added in version 1.6.2.

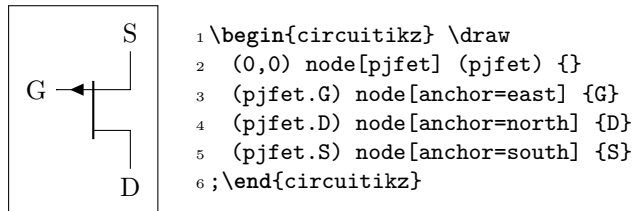
⁵⁹Follows the syntax of the pattern sequence `\pgfsetdash` — see TikZ manual for details; phase is always zero. Basically you pass pairs of dash-length – blank-length dimensions, see the examples.

4.15.9 Transistors anchors

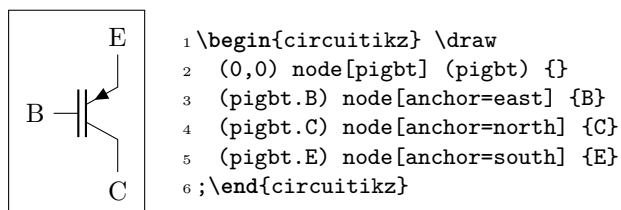
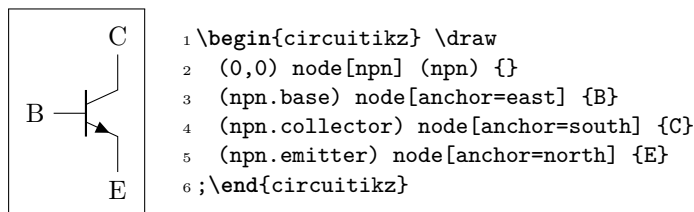
For NMOS, PMOS, NFET, NIGFETE, NIGFETD, PFET, PIGFETE, and PIGFETD transistors one has **base**, **gate**, **source** and **drain** anchors (which can be abbreviated with **B**, **G**, **S** and **D**):



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

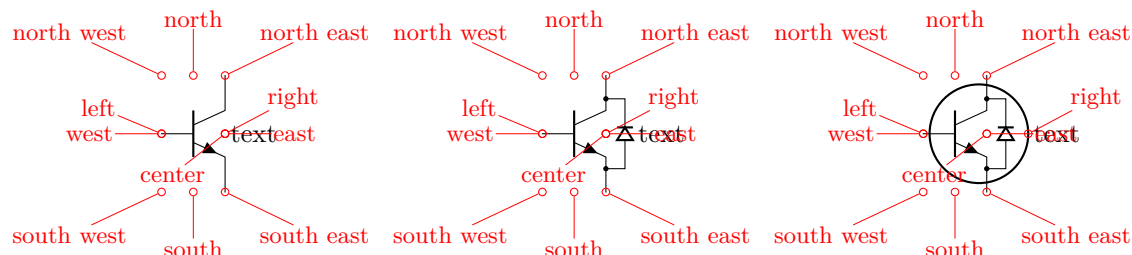


For NPN, PNP, NIGBT and PIGBT transistors, the anchors are **base**, **emitter** and **collector** anchors (which can be abbreviated with **B**, **E** and **C**):

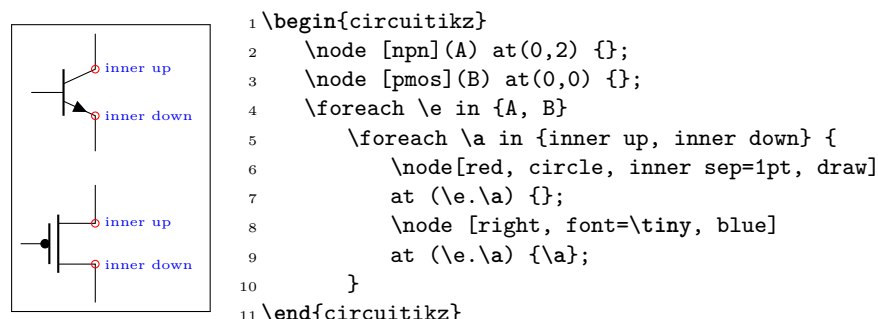


Notice that the geographical anchors of transistors are *not* affected by either the bodydiode and the circle options; the label text is also outside of them. This is to permit aligning the components independently from those features. On the other hand, this can sometimes create problems because that element is outside the bounding box automatically calculated by TikZ.

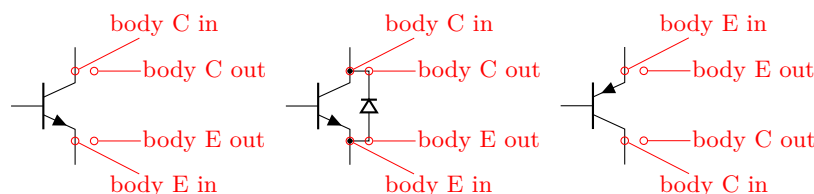
The exception is the **right** anchor which, when a circle is present, indicates the edge of the circle itself (since v1.3.2)



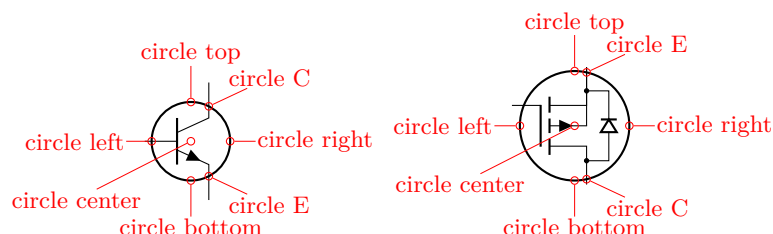
All transistors, except the multi-terminal **bjtnpn** and **bjtpnp**, (since 0.9.6) have internal nodes on the terminal corners, called **inner up** and **inner down**; you do not normally need them, but they are here for special applications:



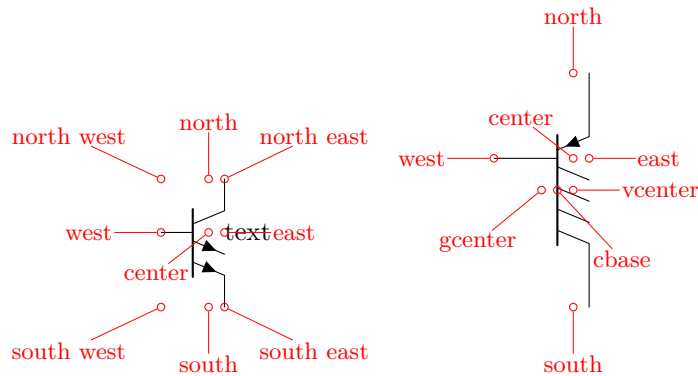
Additionally, you can access the position for the flyback diodes and possibly snubbers as shown in 4.15.5.4.



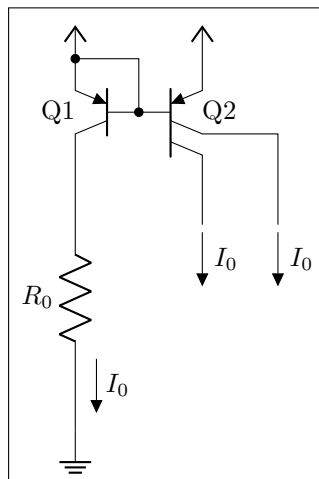
Transistor circles also have several anchors on them:



The multi-terminal transistors have all the geographical anchors; note though that the **center** anchor is not the geometrical center of the component, but the logical one (at the same height as the base). The additional anchors **vcenter** (vertical geometric center of the collector-emitter zone) and **gcenter** (graphical center) are provided, as shown in the following picture. They have no bodydiode anchors nor **inner up/down** ones.



A complete example of multiple terminal transistor application is the following PNP double current mirror circuit.

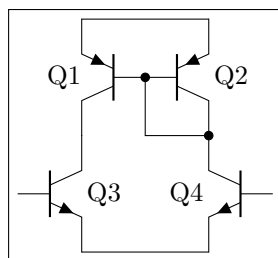


```

1 \begin{circuitikz}
2   \ctikzset{transistors/arrow pos=end}
3   \draw (0,0) node[bjtnpn, xscale=-1] (Q1){%
4     \scalebox{-1}[1]{Q1}};
5   \draw (Q1.B) node[bjtnpn, anchor=B, collectors=2]
6     (Q2){Q2} (Q1.B) node[circ]{};
7   \draw (Q1.E) node[circ]{} node[vcc]{} (Q2.E)
8     node[vcc]{} (Q1.E) -| (Q1.B);
9   \draw (Q1.C) to[R, l_=$R_0$, f=$I_0$] ++(0,-3.5)
10    node[ground] (GND){};
11   \draw (Q2.C) -- ++(0,-0.5) coordinate(a);
12   \draw (Q2.C1) -- ++(1,0) coordinate(b) -- (b|-a);
13   \draw (a) ++(0,-0.1) node[flowarrow, rotate=-90,
14     anchor=west]{\rotatebox{90}{$I_0$}};
15   \draw (b|-a) ++(0,-0.1) node[flowarrow, rotate=-90,
16     anchor=west]{\rotatebox{90}{$I_0$}};
17   \path (b) ++(0.5,0); % bounding box adjust
18 \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`):

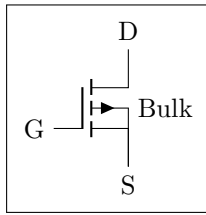


```

1 \begin{circuitikz} []\draw
2 (0,0) node[pnp] (pnp2) {Q2}
3 (pnp2.B) node[pnp, xscale=-1, anchor=B] (pnp1) {}
4 (pnp1) node[left, inner sep=0pt] {Q1}
5 (pnp1.C) node[npn, anchor=C] (npn1) {Q3}
6 (pnp2.C) node[npn, xscale=-1, anchor=C] (npn2)
7   {\scalebox{-1}[1]{Q4}}
8 (pnp1.E) -- (pnp2.E) (npn1.E) -- (npn2.E)
9 (pnp1.B) node[circ] {} |- (pnp2.C) node[circ] {}
10;\end{circuitikz}

```

Notice that the text labels of transistors are outside the bounding box of the component (that is, the set of geographical anchors). If it is a problem, use a separate text node to set the transistor's label. Of course, transistors like other components can be reflected vertically:

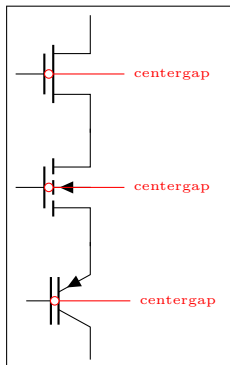


```

1 \begin{circuitikz} \draw
2   (0,0) node[pigfete, yscale=-1] (pigfete) {}
3   (pigfete.bulk) node[anchor=west] {Bulk}
4   (pigfete.G) node[anchor=east] {G}
5   (pigfete.D) node[anchor=south] {D}
6   (pigfete.S) node[anchor=north] {S}
7 ;\end{circuitikz}

```

Finally, double-gated components (MOSes, FETs, IGBTs) have an extra anchor `centergap` positioned in the middle of the “gate capacitor” or base.



```

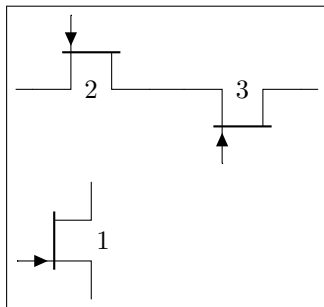
1 \begin{circuitikz}
2   \node [nmos](A) at (0,3) {};
3   \node [nfet](B) at (0,1.5) {};
4   \node [pigbt](C) at (0,0) {};
5   \foreach \myn in {A, B, C}
6     \draw[color=red] (\myn.centergap)
7       node[ocirc]{} -- ++(1,0)
8       node[right, font=\tiny]{centergap};
9 \end{circuitikz}

```

For UJT transistors anchors, see section [4.15.5.9](#).

4.15.10 Transistor paths

For syntactical convenience standard transistors (not multi-terminal ones) can be placed using the normal path notation used for bipoles. The transistor 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:

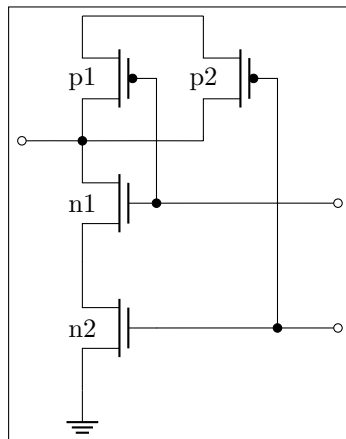


```

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

```

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



```

1 \begin{circuitikz} \draw[yscale=1.1, xscale=.8]
2   (2,4.5) -- (0,4.5) to[Tpmos=p1, n=p1] (0,3)
3     to[Tnmos=n1, n=n1] (0,1.5)
4     to[Tnmos=n2, n=n2] (0,0) node[ground] {}
5   (2,4.5) to[Tpmos=p2,n=p2] (2,3) to[short, -*] (0,3)
6   (p1.G) -- (n1.G) to[short, *-o] ($(n1.G)+(3,0)$)
7   (n2.G) ++(2,0) node[circ] {} -| (p2.G)
8   (n2.G) to[short, -o] ($(n2.G)+(3,0)$)
9   (0,3) to[short, -o] (-1,3)
10;\end{circuitikz}

```

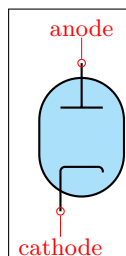
Transistors used in path are fully path-style components, so to flip and rotate them you should use `mirror` and `invert` as shown in section 3.1.4.2.

Transistor paths have the possibility to use the poles syntax (see section 6.1) but they have **no** voltage, current, flow, annotation options. Also, the positioning of the labels is very simple and is not foolproof for all rotations; if you need to control them more please name the node and position them by hand, or use the more natural node style for transistors.

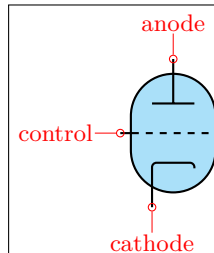
The `name` property is available also for bipoles; this is useful mostly for triac, potentiometer and thyristor (see 4.4.1).

4.16 Electronic Tubes

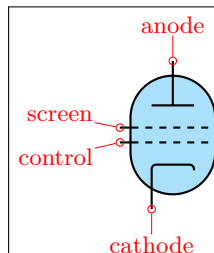
Electronic tubes, also known as vacuum tubes, control current flow between electrodes. They come in many different flavours. Contributed by J. op den Brouw (J.E.J.opdenBrouw@hhs.nl).



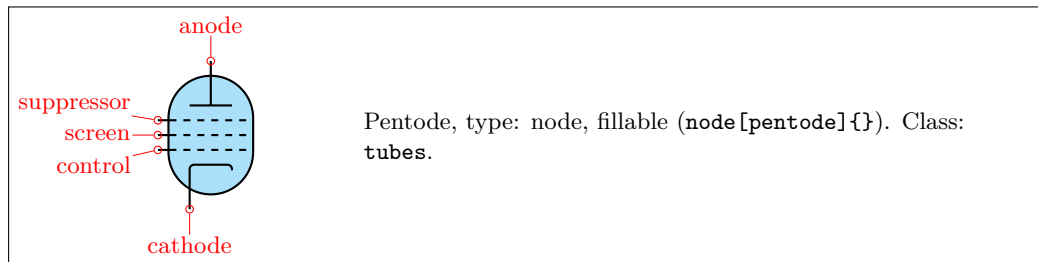
Tube Diode, type: node, fillable (`node[diode tube]`). Class: **tubes**.



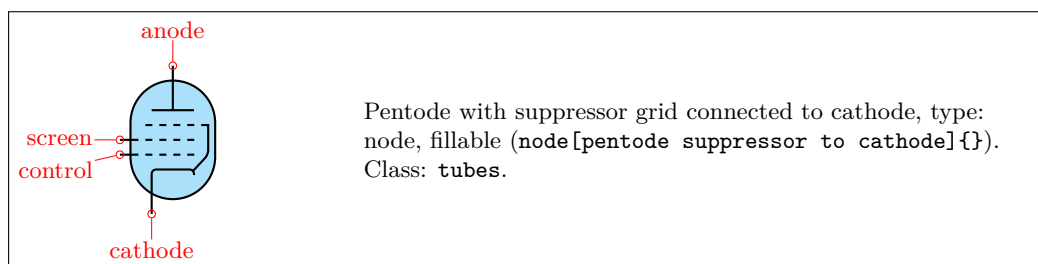
Triode, type: node, fillable (`node[triode]`). Class: **tubes**.



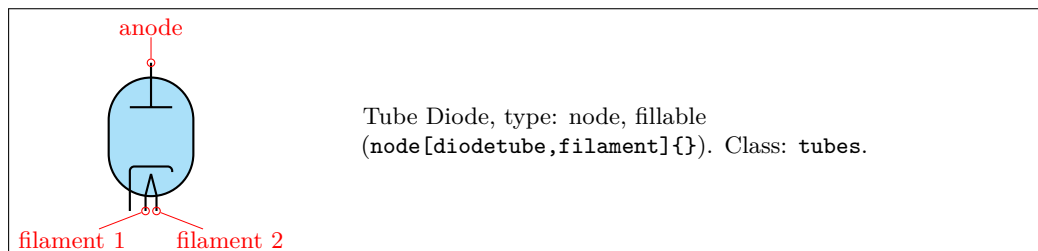
Tetrode, type: node, fillable (`node[tetrode]`). Class: **tubes**.



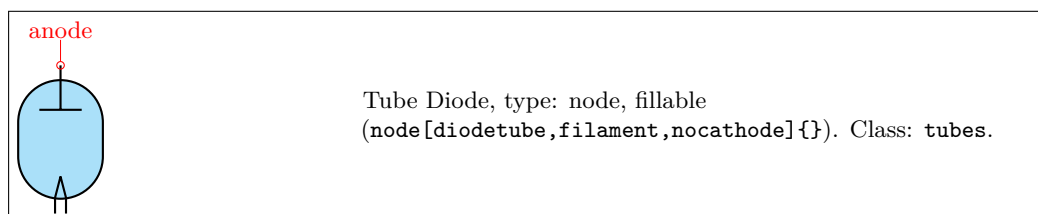
Some pentodes have the suppressor grid internally connected to the cathode, which saves a pin on the tube's housing.



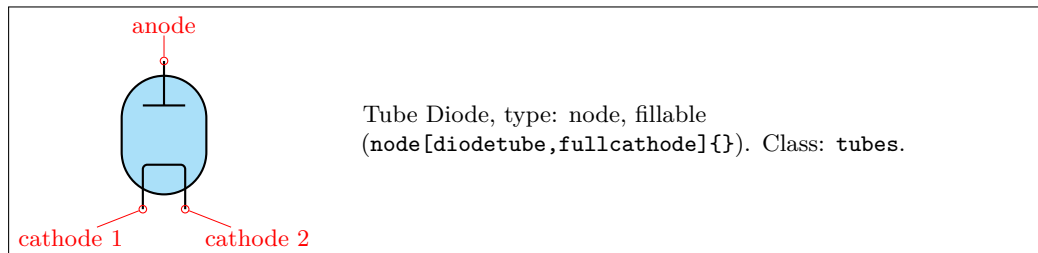
Note that the `diode tube` is used as component name to avoid clashes with the semiconductor diode. Normally, the filament is not drawn. If you want a filament, put the `filament` option in the node description:



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



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`.

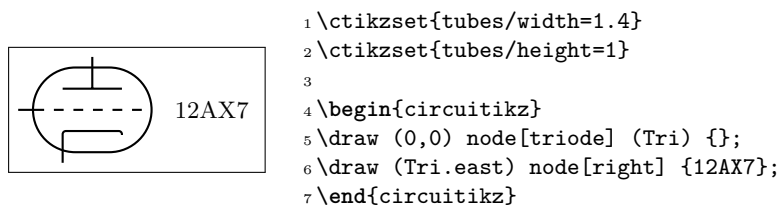


4.16.1 Tubes customization

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

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



Example triode amplifier:

```

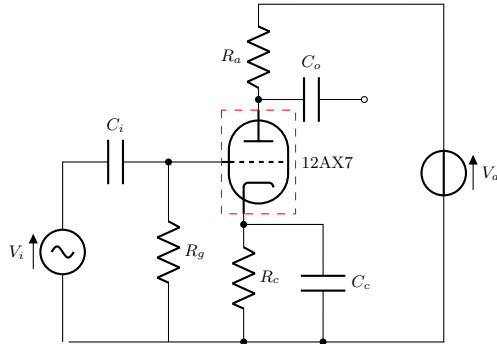
1 \begin{circuitikz}
2 \draw (0,0) node (start) {}
3     to[sV=$V_i$] ++(0,2+\ctikzvalof{tubes/height})
4     to[C=$C_i$] ++(2,0) coordinate(Rg)
5     to[R=$R_g$] (Rg |- start)
6 (Rg)     to[short,*-] ++(1,0)
7     node[triode,anchor=control] (Tri) {} ++(2,0)
8 (Tri.cathode) to[R=$R_c$,-*] (Tri.cathode |- start)
9 (Tri.anode)  to [R=$R_a$] ++(0,2)
10            to [short] ++(3.5,0) node(Vatop) {}
11            to [V<=$V_a$] (Vatop |- start)

```

```

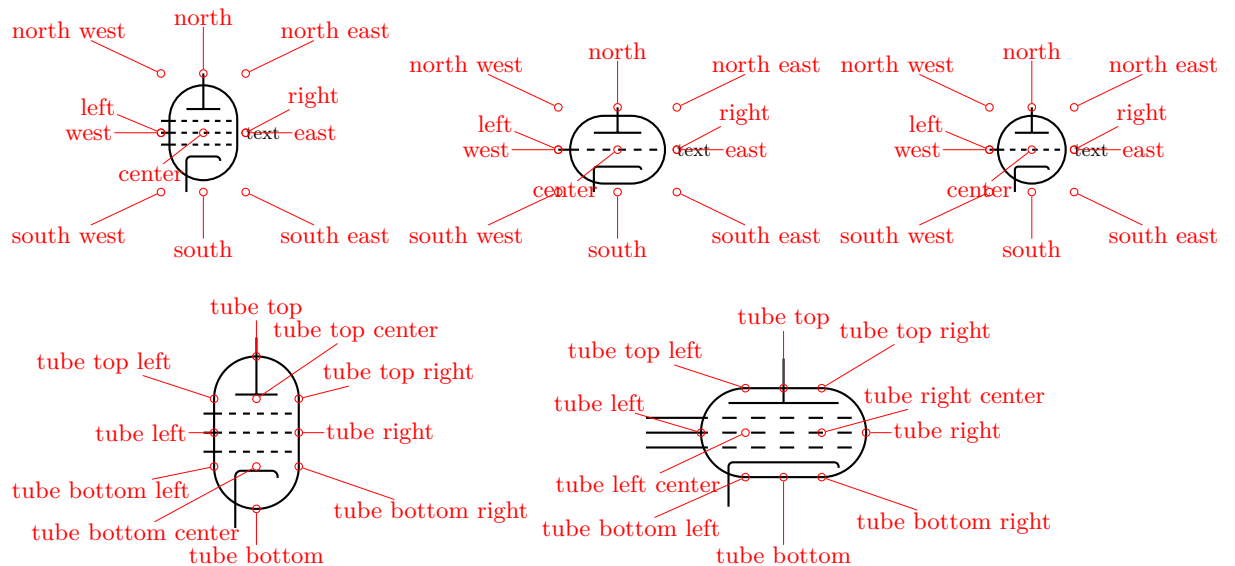
12         to [short] (start)
13 (Tri.anode)  ++(0,0.2) to[C=$C_o$,*-o] ++(2,0)
14 (Tri.cathode) ++(0,-0.2) to[short,*-] ++(1.5,0) node(Cctop) {}
15             to[C=$C_c$,*-] (start -| Cctop)
16 ;
17 \draw[red,thin,dashed] (Tri.north west) rectangle (Tri.south east);
18 \draw (Tri.east) node[right] {12AX7};
19 \end{circuitikz}

```



4.16.2 Tubes anchors

Apart from the geographic anchors, which take into account the leads of the components, you have several anchors on the border:



4.16.3 Partially drawn tube borders

In some circuits, tubes are drawn with partial or dashed border (to convey the meaning of several active components encased in the same physical tube). To achieve this effect, you can use the `tubes/partial border`⁶⁰ key (default `none`). This key can be set to `none`, or must be a sequence of **exactly** 6 numbers, which can have value 0, 1, or 2. Each number defines the style of a part of the border to be not drawn, solid or dashed respectively.

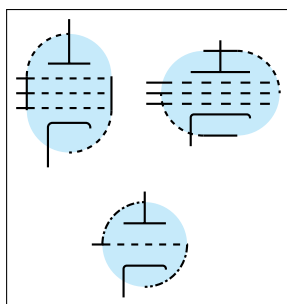
The part of the border are numbered from 1 to 6 as shown below:

⁶⁰Suggested by [Jether Fernandes Reis](#), implemented by Romano in v1.5.2.



(notice that the straight parts, if they exist, are numbered 2 and 5 in both tubes, vertical or horizontal).

The dashed line pattern can be changed by setting the key `tubes/partial border dash` (default `{{2pt}{2pt}}`).⁶¹ Be careful with the extra set of braces here.



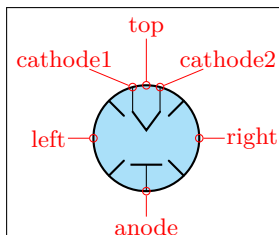
```

1 \begin{circuitikz}[circuitikz/tubes/fill=cyan!20,
2   circuitikz/tubes/partial borders=012012]
3   \draw (0,0) node[pentode]{};
4   \draw (2,0) node[triode,
5     circuitikz/tubes/width=1.4,
6     circuitikz/tubes/height=1]{};
7   \draw (1,-2) node[diode,
8     circuitikz/tubes/height=1,
9     circuitikz/tubes/partial border dash=%
10      {{3pt}{1pt}}]{};
11 \end{circuitikz}

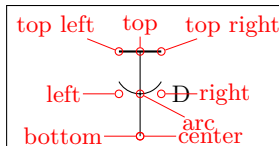
```

4.16.4 Other tubes-like components

The `magnetron` and `dynode` shapes will also scale with `tubes/scale`.



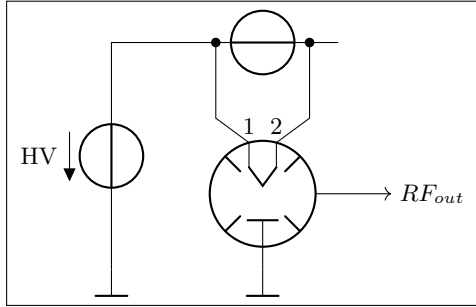
Magnetron, type: node, fillable (`node[magnetron]{}).` Class: **tubes.**



Dynode⁶², type: node (`node[dynode]{D}.`) Class: **tubes.**

⁶¹Follows the syntax of the pattern sequence `\pgfsetdash` — see TikZ manual for details; phase is always zero. Basically you pass pairs of dash-length – blank-length dimensions, see the examples.

⁶²Suggested by the user `ferdymercury` on [GitHub](#).



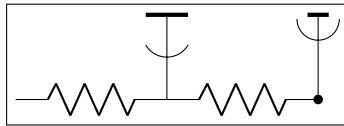
```

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

```

4.16.4.1 Dynode customization. The dynode element can be heavily customized. The parameters are the following (all of them under the `\ctikzset` family `monopoles/dynode`):

parameter	default	description
<code>width</code>	0.4	Total width (relative to the base length) measured at the arc width.
<code>height</code>	0.8	Total height (same units as width).
<code>arc angle</code>	30	Angle (from the horizontal, going down) where the arc starts. A value of 90 don't plot any arc, 0 plots a semicircle. To avoid artifacts, use a value between -60 and 90; the arc horizontal size is always equal to the <code>width</code> .
<code>arc pos</code>	0.5	Vertical position (relative to the height) of the arc center.
<code>top width</code>	1.0	Relative width of the top bar; a value of 1 means full width, 0 means no bar.



```

1 \begin{circuitikz}[american] \ctikzset{tubes/thickness=4}
2 \draw (0,0) to[R] (2,0) node[dynode]{} to[R,*] (4,0);
3 \ctikzset{monopoles/dynode/.cd,
4   arc angle=0, arc pos=0.7, top width=0.5}
5 \draw (4,0) node[dynode]{};
6 \end{circuitikz}

```

You can use styles and the parameters to create different types of electrodes:



```

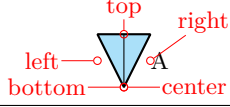
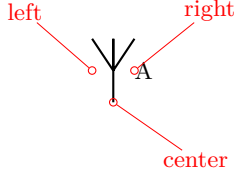
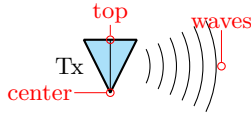
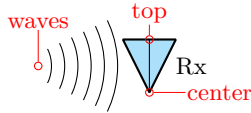
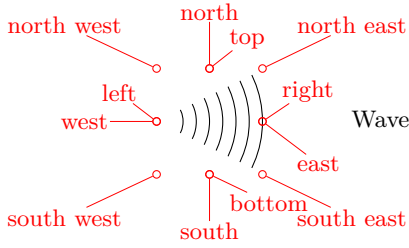
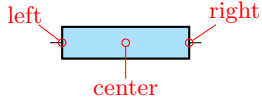
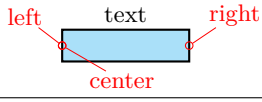
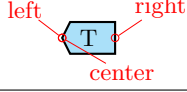
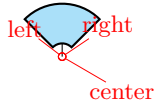
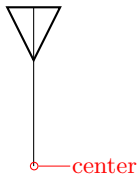
1 \begin{circuitikz}[american] \ctikzset{tubes/thickness=4}
2 \tikzset{anode/.style={dynode,
3   circuitikz/monopoles/dynode/arc angle=90},
4   photocathode/.style={dynode,
5   circuitikz/monopoles/dynode/arc pos=1,
6   circuitikz/monopoles/dynode/top width=0},
7 }
8 \draw (0,0) node[dynode]{} (1,0) node[anode]{}
9   (2,0) node[photocathode]{};
10 \end{circuitikz}

```

4.17 RF components

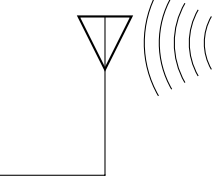
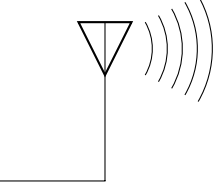
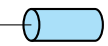
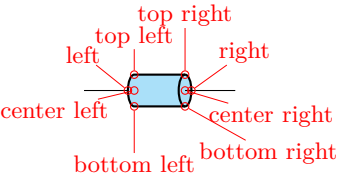
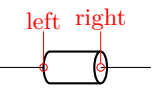

For the RF components, similarly to the grounds and supply rails, the `center` anchor is put on the connecting point of the symbol, so that you can use them directly in a `path` specification.

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

	Bare Antenna, type: node, fillable (<code>node[bareantenna]{A}</code>). Class: RF.
	DIN antenna ⁶³ , type: node, fillable (<code>node[dinantenna]{A}</code>). Class: RF.
	Bare TX Antenna, type: node, fillable (<code>node[bareTXantenna]{Tx}</code>). Class: RF.
	Bare RX Antenna, type: node, fillable (<code>node[bareRXantenna]{Rx}</code>). Class: RF.
	Waves, type: node (<code>node[waves]{}</code>). Class: RF.
	mstline : Microstrip transmission line ⁶⁴ , type: path-style, fillable, nodename: mstlineshape. Class: RF.
	Microstrip linear stub, type: node, fillable (<code>node[mslstub]{text}</code>). Class: RF.
	Microstrip port, type: node, fillable (<code>node[msport]{T}</code>). Class: RF.
	Microstrip radial stub, type: node, fillable (<code>node[msrstub]{}</code>). Class: RF.
	Legacy antenna (with tails), type: node (<code>node[antenna]{}</code>). Class: RF.

⁶³Since 1.5.0, suggested by [GitHub user myzinsky](#)

⁶⁴These four components were suggested by [@tcpluuss](#) on GitHub

	Legacy receiving antenna (with tails), type: node (<code>node[rxantenna]{}</code>). Class: RF.
	Legacy transmitting antenna (with tails), type: node (<code>node[txantenna]{}</code>). Class: RF.
	Transmission line stub, type: node, fillable (<code>node[tlinestub]{}</code>). Class: RF.
	TL: Transmission line, type: path-style, fillable, nodename: <code>tlineshape</code> . Aliases: transmission line, <code>tline</code> . Class: RF.
	TL, <code>bipoles/tline/bare=true</code> : Transmission line without wires (notice that if you fill it, the fill will overwrite the ex- iting wire), type: path-style, nodename: <code>tlineshape</code> . Aliases: transmission line, <code>tline</code> . Class: RF.
	match, type: node (<code>node[match]{}</code>). Class: RF.

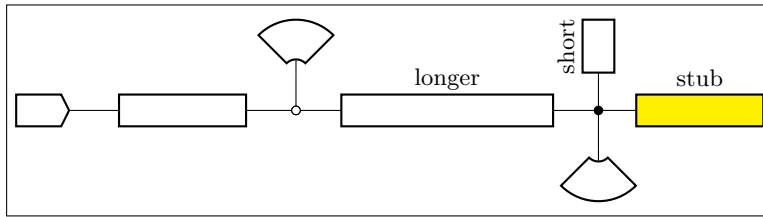
4.17.1 RF elements customization

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

4.17.2 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`.

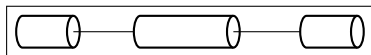


```

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

```

The legacy `tline` can be used as in the following example. You can change the length with the key `bipoles/tline/width` (default 0.6). The “bare” version, which differs only for the small line on the visible ellipse, and activated with the boolean key `.../bare`, is useful as a substitute for `tlinestub` (with more flexibility).



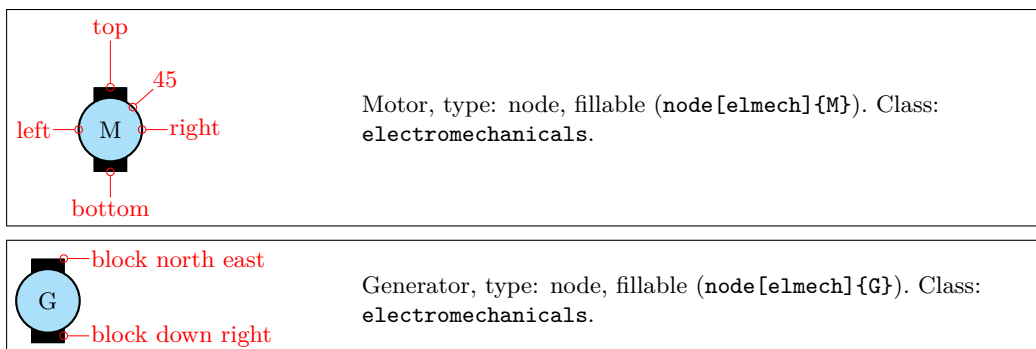
```

1 \begin{tikzpicture}[]
2   \tikzset{bare t1/.style={tlineshape,
3     circuitikz/bipoles/tline/bare=true}}
4   \draw (0,0) node[bare t1](A){} (A.right)
5     to[TL, bipoles/tline/width=1] ++(3,0)
6     node[bare t1, anchor=left]{};
7 \end{tikzpicture}

```

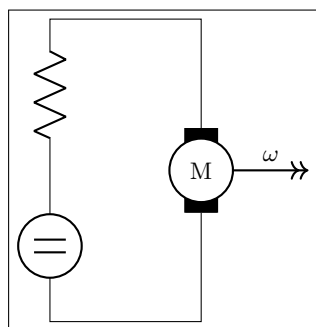
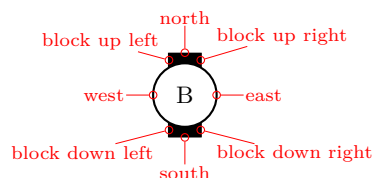
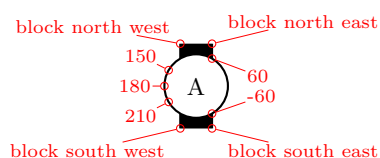
4.18 Electro-Mechanical Devices

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



4.18.1 Electro-Mechanical Devices anchors

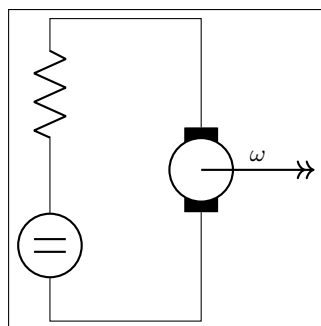
Apart from the standard geographical anchors, `elmech` has the border anchors (situated on the inner circle) and the following anchors on the “block”:



```

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

```

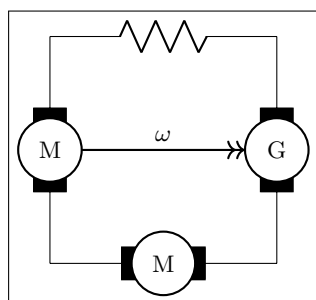


```

1 \begin{circuitikz}
2 \draw (2,0) node[elmech](motor){};
3 \draw (motor.north) |-(0,2) to [R] ++(0,-2) to [dcvsource]
  ++(0,-2) -| (motor.bottom);
4 \draw[thick,->>](motor.center)--++(1.5,0)node[midway,above]
  {$\omega$};
5 \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 4.15.10). Don't forget to use parameter n to name the node and get access to the anchors:



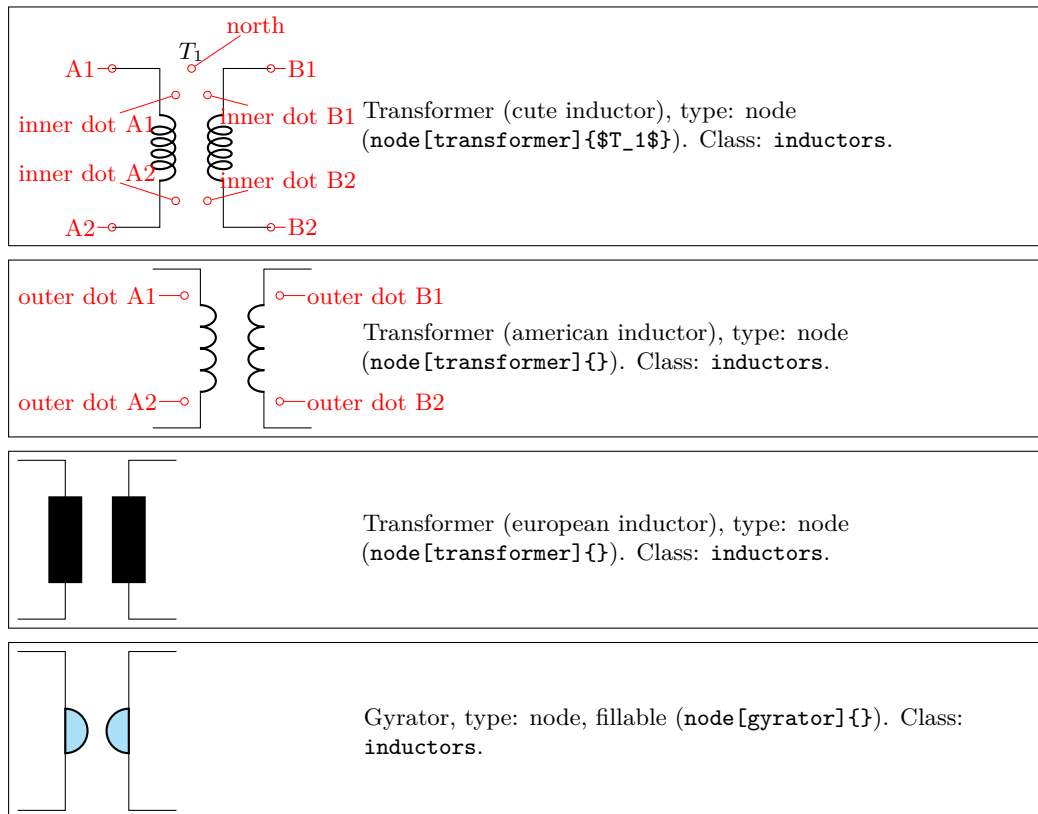
```

1 \begin{circuitikz}
2 \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);
3 \draw[thick,->>](motor.left)--(generator.left)node[midway,
  above]{$\omega$};
4 \end{circuitikz}

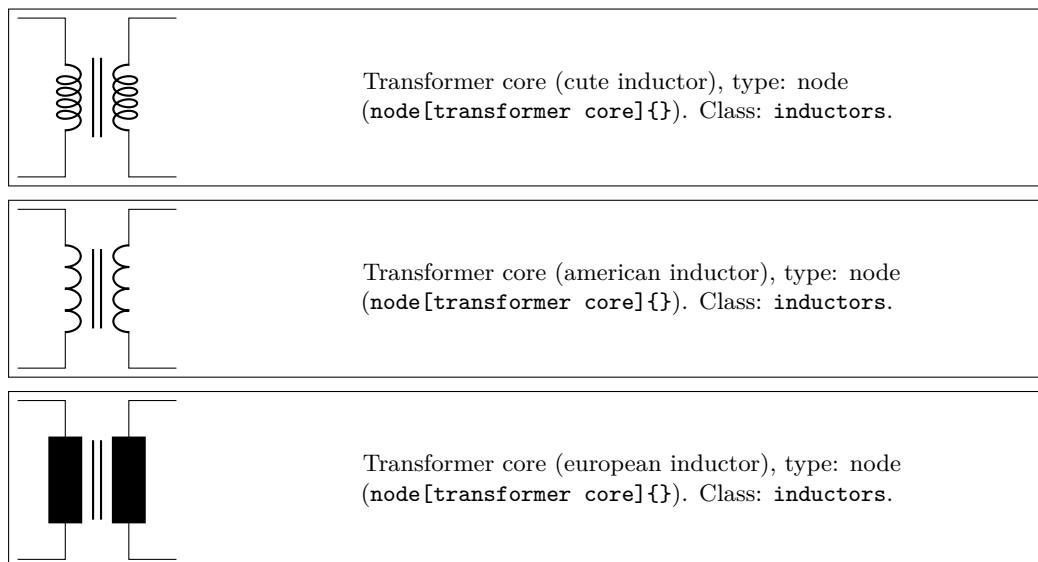
```

4.19 Double bipoles (transformers)

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

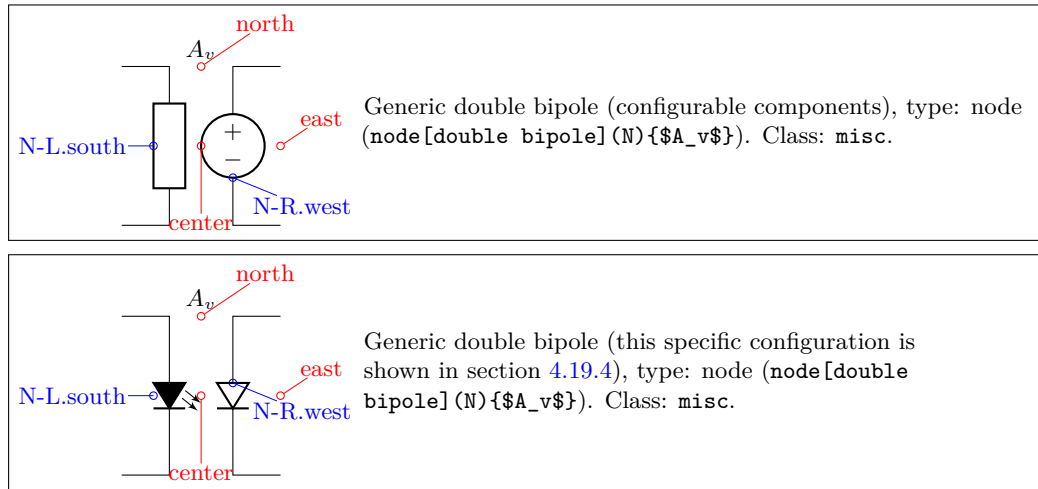


Transformers with core are also available:



You can also build generic double bipoles⁶⁵ (although it's often better to use subcircuits in this case; see section 3.4).

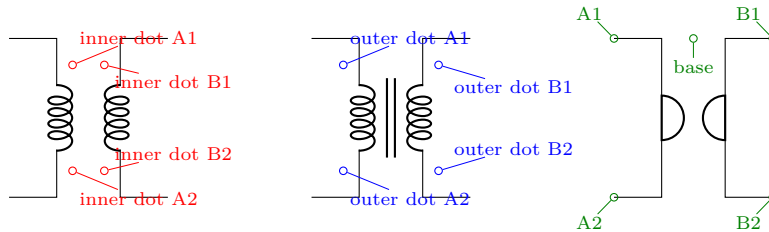
⁶⁵The idea of generic double bipoles was originated by user [erwindenboer on GitHub](#).



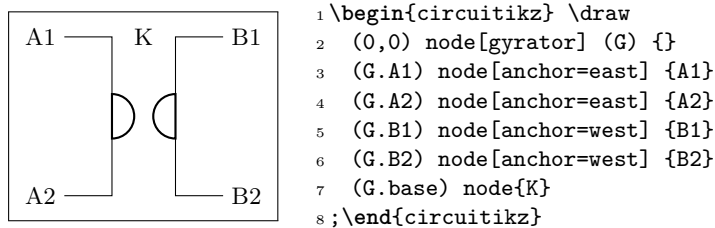
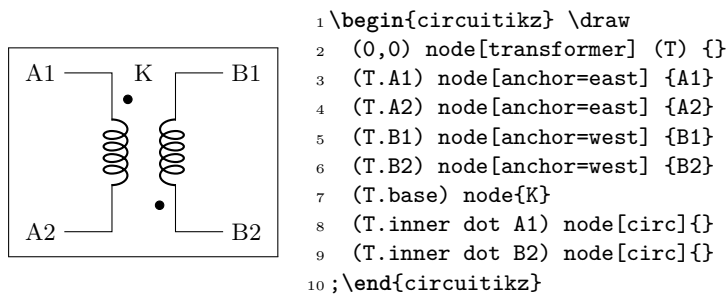
4.19.1 Transformer anchors

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.

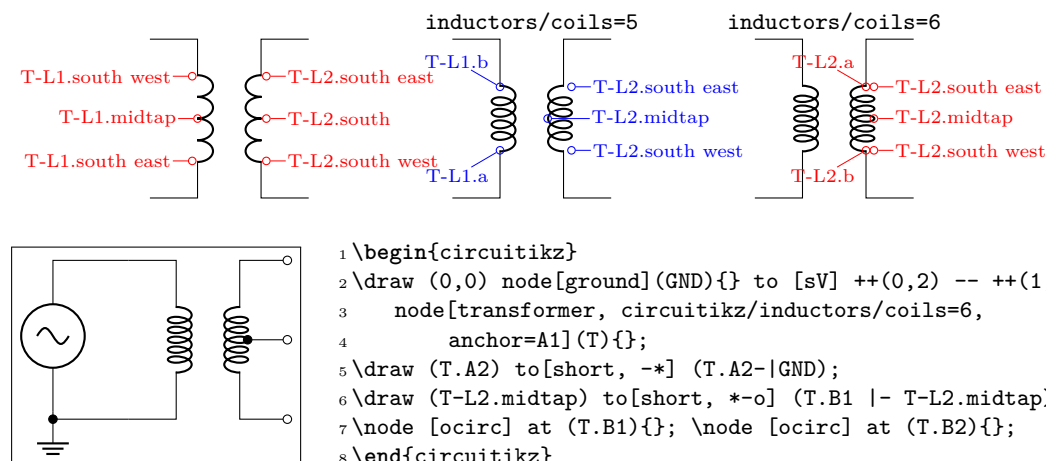
They also expose the **base** anchor, for labeling, and anchors for setting dots or signs to specify polarity. The set of anchors, to which the standard “geographical” **north**, **north east**, etc. is here:



Also, the standard “geographical” **north**, **north east**, etc. are defined. A couple of examples follow:

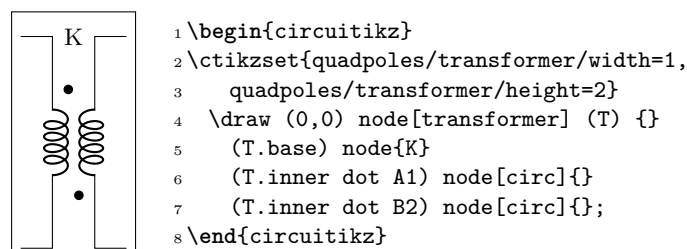


Moreover, you can access the two internal coils (inductances); if your transformer node is called T, they are named T-L1 and T-L2. Notice that the two inductors are rotated (by -90 degrees the first, +90 degrees the second) so you have to be careful with the anchors. Also, the `midtap` anchor of the inductors can be on the external or internal side depending on the numbers of coils. Finally, the anchors L1.a and L1.b are marking the start and end of the coils.



4.19.2 Transformers 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/*/height` (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!



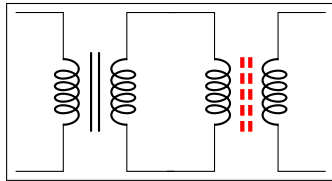
Transformers also inherit the `inductors/scale` (see 4.3.5) and similar parameters. It's your responsibility to set the aforementioned parameters if you change the scale or width of inductors.

Transformers core line distance is specified by the parameter `quadpoles/transformer core/core width` (default 0.05) and the thickness of the lines follows the choke one; in other words, you can set it changing `bipoles/cutechoke/cthick`.

You can change the style of the core lines⁶⁶ in a similar way to the one used for transistor's bodydiodes, by setting keys with the `\ctikzset` command under the `transformer core` hierarchy. The available keys are:

⁶⁶Suggested by [user myzinsky on GitHub](#), implemented in v1.6.2.

parameter	default	description
relative thickness	1.0	multiply the default thickness (which is the same of the choke component).
color	default	stroke color: default is the same as the component.
dash	default	dash pattern: default means not to change the setting for the component; none means unbroken line; every other input is a dash pattern. ⁶⁷

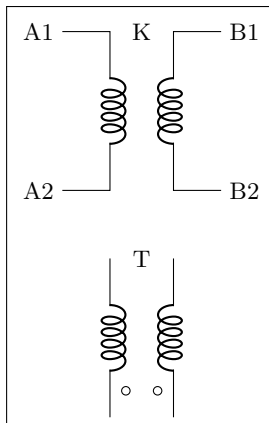


```

1 \begin{circuitikz}[]
2   \draw (0,0) node[transformer core](A){};
3   \ctikzset{transformer core/.cd, relative thickness=2,
4             color=red, dash={4pt}{2pt}}
5   \draw (2,0) node[transformer core](B){};
6 \end{circuitikz}

```

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):

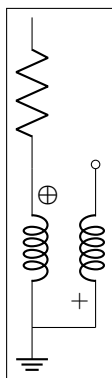


```

1 \begin{circuitikz}
2 \draw (0,0) node[transformer] (T) {}
3 (T.A1) node[anchor=east] {A1}
4 (T.A2) node[anchor=east] {A2}
5 (T.B1) node[anchor=west] {B1}
6 (T.B2) node[anchor=west] {B2}
7 (T.base) node{K} ;
8 \ctikzset{quadpoles/transformer/inner=1, quadpoles/transformer/
9 width=0.6}
10 \draw (0,-3) node[transformer] (P) {}
11 (P.base) node{T}
12 (P.inner dot A2) node[ocirc]{}
13 (P.inner dot B2) node[ocirc]{};
14 \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**):



```

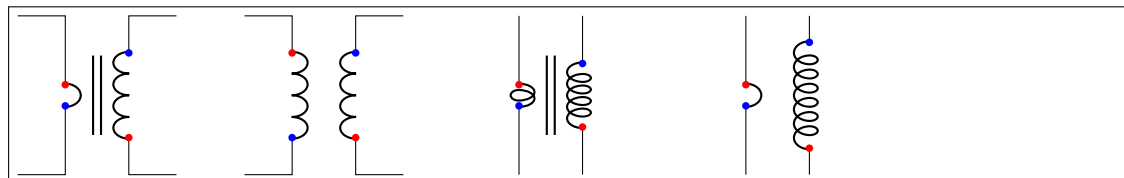
1 \begin{circuitikz}
2 \ctikzset{inductor=cute, quadpoles style=inline}
3 \draw
4 (0,0) to[R] ++(0,-2)
5 node[transformer, anchor=A1] (T){}
6 (T.A2) node[ground] (GND){}
7 (T.inner dot A1) node[font=\small\boldmath]{$\oplus$}
8 (T.inner dot B2) node[]{$+$}
9 (T.B1) node[above, ocirc]{}
10 (T.B2) -- (GND);
11 \end{circuitikz}

```

⁶⁷Follows the syntax of the pattern sequence `\pgfsetdash` — see TikZ manual for details; phase is always zero. Basically you pass pairs of dash-length – blank-length dimensions, see the examples.

4.19.3 Styling transformer's coils independently

Since 0.9.6, you can tweak the style of each of the coils of the transformers by changing the value of the two styles `transformer L1` and `transformer L2`; the default for both are {}, that means inherit the inductors style in force.

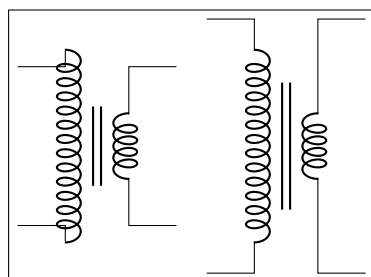


```

1 \begin{circuitikz}[american]
2   \begin{scope}
3     \ctikzset{transformer L1/.style={inductors/coils=1, inductors/width=0.2}}
4     \draw (0,0) node[transformer core](T1){};
5   \end{scope}
6   \draw (3,0) node[transformer](T2){};
7   \ctikzset{cute inductors, quadpoles style=inline}
8   \ctikzset{transformer L1/.style={inductors/coils=2, inductors/width=0.2}}
9   \draw (6,0) node[transformer core](T3){};
10  \ctikzset{transformer L1/.style={american inductors, inductors/coils=1, inductors/
    width=0.2}}
11  \ctikzset{transformer L2/.style={inductors/coils=7, inductors/width=1.0}}
12  \draw (9,0) node[transformer](T4){};
13  \foreach \t in {T1, T2, T3, T4} {
14    \foreach \l in {L1, L2} {
15      \foreach \a/\c in {a/blue, b/red}
16        \node [circle, fill=\c, inner sep=1pt] at (\t-\l.\a) {};
17    }
18  }
19 \end{circuitikz}

```

Caveat: the size of the transformer is independent from the styles for L1 and L2, so they follow whatever the parameters for the inductances were before applying them. In other words, the size of the transformer could result too small if you are not careful.



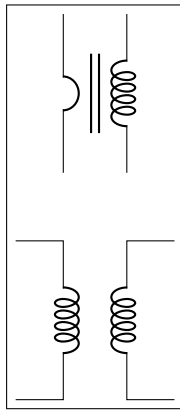
```

1 \begin{circuitikz}
2   \ctikzset{transformer L1/.style={inductors/width=1.8,
    inductors/coils=13}}
3   % too small!
4   \draw (0,0) node[transformer core](T1){};
5   % adjust it
6   \ctikzset{quadpoles/transformer core/height=2.4}
7   \draw (2.5,0) node[transformer core](T1){};
8 \end{circuitikz}

```

You can obviously define a style for a “non-standard” transformer. For example, you can have a current transformer⁶⁸ defined like this:

⁶⁸Suggested by Alex Pacini on [GitHub](#)



```

1 \begin{circuitikz}[
2   TA core/.style={transformer core,
3     % at tikz level, you have to use circuitikz/ explicitly
4     circuitikz/quadpoles style=inline,
5     circuitikz/transformer L1/.style={
6       american inductors, inductors/coils=1,
7       inductors/width=0.3},
8   } ]
9   \draw (0,0) node[TA core](T1){};
10  % changes are local
11  \draw (0,-3) node[transformer]{};
12 \end{circuitikz}

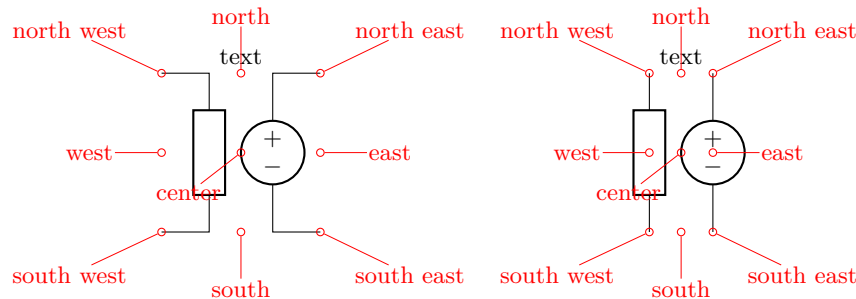
```

Remember that the default `pgfkeys` directory is `/tikz` for nodes and for the options of the environment, so you *have* to use the full path (with `circuitikz/`) there.

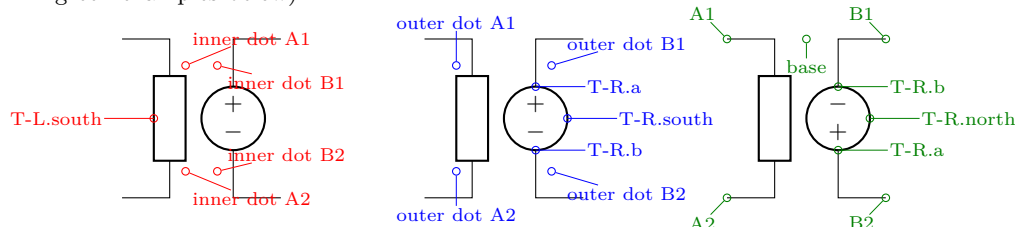
4.19.4 Generic double bipoles

Generic double bipoles have more or less the same keys for size that the transformers (like `../width`, `../inner` etc.) using the component name `double bipole`. Also the anchors are similar, with the main difference that the “dot” anchors are fixed, so they do *not* adapt to the size of the component. Another important difference is that the class of the generic double bipole is `misc`, not `inductors` (which is reserved to transformers and, for an historical hiccup, to the gyrator).

By default, the left component is a generic impedance, and the right one is an (American-style, it will not change automatically) voltage generator. You can use `quadpoles style=inner` as shown in the rightmost drawing below.



The other anchors behave similarly to the transistor’s ones; you also have access to the internal components nodes by using `nodename-R` and `nodename-L` names for the right and left element, which is supposed to be T in the drawing below. Be wary that given that here you can (see later) reverse the direction of one or both of the elements, the rotation (and so the anchors) is not fixed (you can see that in the blue and green examples below).

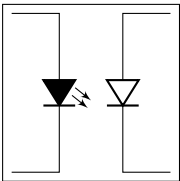


Generic double bipoles are meant to be used through a style, choosing the left and right components. The keys that let you change the components are the following ones:

- `double bipole L`, `double bipole R`: the `nodename` of the component you want on the left and right side (default: `genericshape` and `vsourceshape`).

- `double bipole L invert, double bipole R invert`: controls the direction of the element inserted (default `false` for both; that means that the left bipole goes “down” and the second one “up”).
- `every double bipole L, every double bipole R`: a style that is enacted when drawing the component; by default it’s void.

For example, the LED-diode double bipole at the start of the section could be obtained this way:

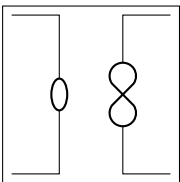


```

1 \begin{circuitikz}[
2   led to D/.style={double bipole,
3     % at tikz level, you have to use circuitikz/ explicitly
4     circuitikz/double bipole L=fulllediodeshape,
5     circuitikz/double bipole R=emptydiodeshape,
6     circuitikz/every double bipole L/.style={diodes/scale=0.6},
7     circuitikz/every double bipole R/.style={diodes/scale=0.6},
8     circuitikz/double bipole R invert,
9   },
10 ]
11 \draw (0,0) node[led to D]{};
12 \end{circuitikz}

```

As a final example, and given that the addition of generic double bipole was stimulated by an issue opened by [user erwindenboer on GitHub](#) suggesting the addition of a nullor shape, the nullor can be obtained like this:



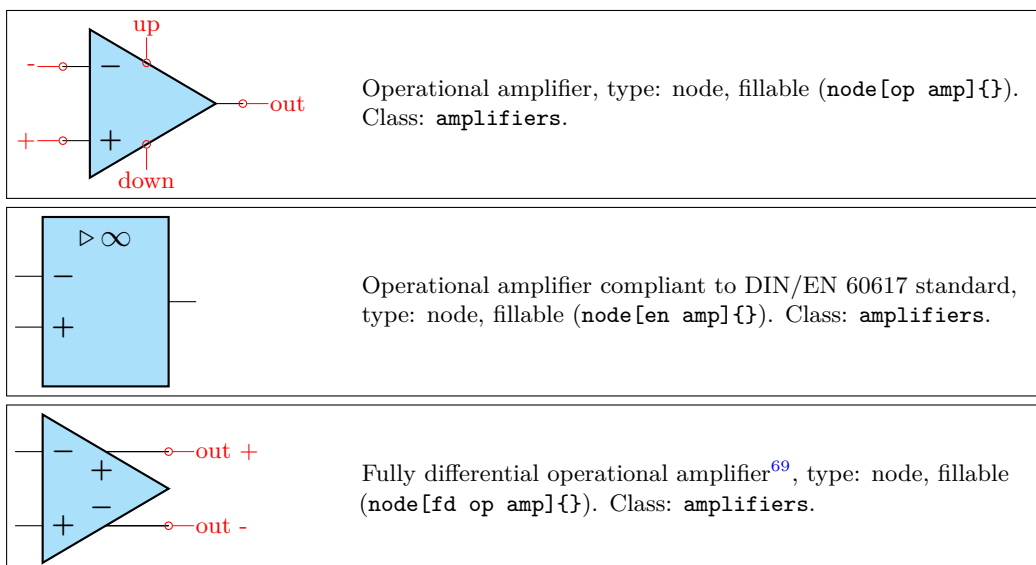
```

1 \begin{circuitikz}[
2   nullor/.style={double bipole,
3     % at tikz level, you have to use circuitikz/ explicitly
4     circuitikz/double bipole L=nullatorshape,
5     circuitikz/double bipole R=noratorshape,
6     circuitikz/every double bipole L/.style={sources/scale=0.5},
7   },
8 ]
9 \draw (0,0) node>nullor(T1){};
10 \end{circuitikz}

```

although now adding currents and voltages is not as trivial as if the component is built with a subcircuit...

4.20 Amplifiers

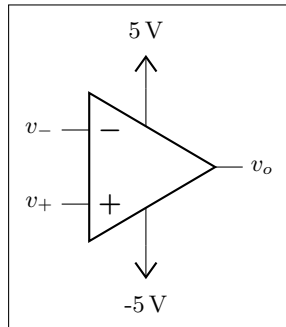


⁶⁹Contributed by Kristofer M. Monisit.

	<p>transconductance amplifier, type: node, fillable (<code>node[gm amp]{}</code>). Class: amplifiers.</p>
	<p>plain instrumentation amplifier, type: node, fillable (<code>node[inst amp]{}</code>). Class: amplifiers.</p>
	<p>Fully differential instrumentation amplifier, type: node, fillable (<code>node[fd inst amp]{}</code>). Class: amplifiers.</p>
	<p>instrumentation amplifier with amplification resistance terminals, type: node, fillable (<code>node[inst amp ra]{}</code>). Class: amplifiers.</p>
	<p>Plain amplifier, unmarked, two inputs, type: node, fillable (<code>node[plain amp]{A\$_1\$}</code>). Class: amplifiers.</p>
	<p>Plain amplifier, one input, type: node, fillable (<code>node[plain mono amp]{}</code>). Class: amplifiers.</p>
	<p>Buffer, type: node, fillable (<code>node[buffer]{}</code>). Class: amplifiers.</p>

4.20.1 Amplifiers anchors

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

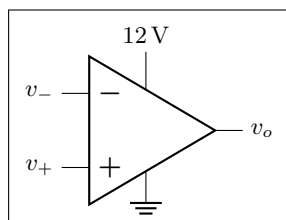


```

1 \begin{circuitikz} \draw
2   (0,0) node[op amp] (opamp) {}
3   (opamp.+) node[left] {$v_+$}
4   (opamp.-) node[left] {$v_-$}
5   (opamp.out) node[right] {$v_o$}
6   (opamp.up) --++(0,0.5) node[vcc]{5\,\textnormal{V}}
7   (opamp.down) --++(0,-0.5) node[vee]{-5\,\textnormal{V}}
8 ;\end{circuitikz}

```

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

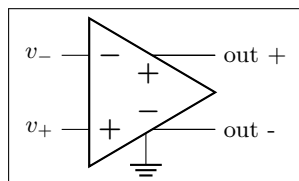


```

1 \begin{circuitikz} \draw
2   (0,0) node[op amp] (opamp) {}
3   (opamp.+) node[left] {$v_+$}
4   (opamp.-) node[left] {$v_-$}
5   (opamp.out) node[right] {$v_o$}
6   (opamp.down) node[ground] {}
7   (opamp.up) ++ (0,.5) node[above] {\SI{12}{\volt}}
8   -- (opamp.up)
9 ;\end{circuitikz}

```

The fully differential op amp defines two outputs:

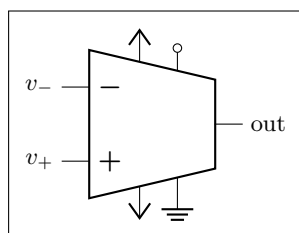


```

1 \begin{circuitikz} \draw
2   (0,0) node[fd op amp] (opamp) {}
3   (opamp.+) node[left] {$v_+$}
4   (opamp.-) node[left] {$v_-$}
5   (opamp.out +) node[right] {out +}
6   (opamp.out -) node[right] {out -}
7   (opamp.down) node[ground] {}
8 ;\end{circuitikz}

```

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

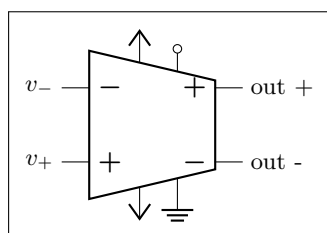


```

1 \begin{circuitikz} \draw
2   (0,0) node[inst amp] (opamp) {}
3   (opamp.+) node[left] {$v_+$}
4   (opamp.-) node[left] {$v_-$}
5   (opamp.out) node[right] {out}
6   (opamp.up) node[vcc]{}
7   (opamp.down) node[vee] {}
8   (opamp.refv down) node[ground]{}
9   (opamp.refv up) to[short, -o] ++(0,0.3)
10 ;\end{circuitikz}

```

The fully differential instrumentation amplifier inst amp defines two outputs:

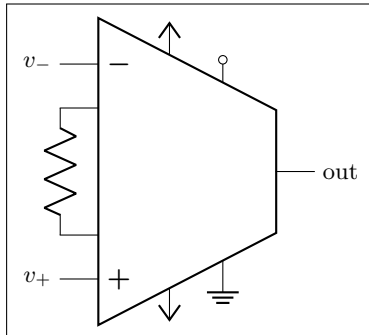


```

1 \begin{circuitikz} \draw
2   (0,0) node[fd inst amp] (opamp) {}
3   (opamp.+) node[left] {$v_+$}
4   (opamp.-) node[left] {$v_-$}
5   (opamp.out +) node[right] {out +}
6   (opamp.out -) node[right] {out -}
7   (opamp.up) node[vcc]{}
8   (opamp.down) node[vee] {}
9   (opamp.refv down) node[ground]{}
10  (opamp.refv up) to[short, -o] ++(0,0.3)
11 ;\end{circuitikz}

```

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

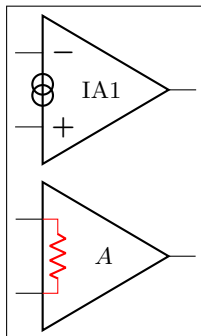


```

1 \begin{circuitikz} \draw
2 (0,0) node[inst amp ra] (opamp) {}
3 (opamp.+) node[left] {$v_+$}
4 (opamp.-) node[left] {$v_-$}
5 (opamp.out) node[right] {out}
6 (opamp.up) node[vcc]{}
7 (opamp.down) node[vee] {}
8 (opamp.refv down) node[ground]{}
9 (opamp.refv up) to[short, -o] ++(0,0.3)
10 (opamp.ra-) to[R] (opamp.ra+)
11 ;\end{circuitikz}

```

Amplifiers also have “border” anchors (just add `b`, without space, to the anchor, like `b+` or `bin up` and so on). These can be useful to add “internal components” or to modify the component. Also the `leftedge` anchor (on the border midway between input) is available.



```

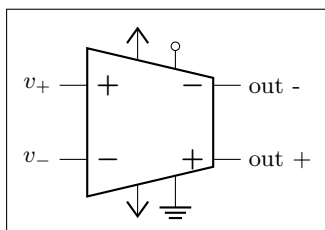
1 \begin{circuitikz}[]
2 \draw (0,2.2) node[op amp] (OA){IA1};
3 \node[oosourceshape, rotate=90, scale=0.5]
4 at (OA.leftedge) {};
5 \draw (0,0) node[plain amp] (A){$A$};
6 \draw [color=red] (A.bin up) -- ++(0.2,0)
7 coordinate (tmp)
8 to[R, resistors/scale=0.5]
9 (tmp|-A.bin down) -- (A.bin down);
10 \end{circuitikz}

```

4.20.2 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.

4.20.2.1 Input polarity. All these amplifiers 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:



```

1 \begin{circuitikz} \draw
2 (0,0) node[fd inst amp,
3 noinv input up,
4 noinv output down] (opamp) {}
5 (opamp.+) node[left] {$v_+$}
6 (opamp.-) node[left] {$v_-$}
7 (opamp.out +) node[right] {out +}
8 (opamp.out -) node[right] {out -}
9 (opamp.up) node[vcc]{}
10 (opamp.down) node[vee] {}
11 (opamp.refv down) node[ground]{}
12 (opamp.refv up) to[short, -o] ++(0,0.3)
13 ;\end{circuitikz}

```

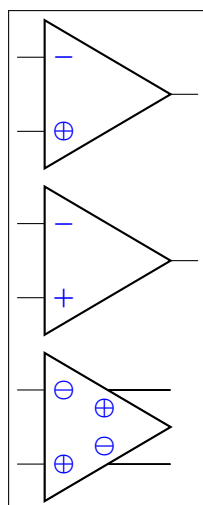
When you use the `noinv input/output ...` keys the anchors (+, -, out +, out -) will change with the effective position of the terminals. You also have the anchors `in up`, `in down`, `out up`, `out down` that will not change with the positive or negative sign.

4.20.2.2 Input and output pins symbols. You can change the symbols “+” or “−” appearing in the amplifiers if you want, both globally and on component-by-component basis. The plus and minus symbols can be changed with `\ctikzset` of the keys `amplifiers/plus` and `amplifiers/minus` (which defaults to the math mode plus or minus cited before), or using the styles `amp plus` and `amp minus`.

The font used is set in several keys, but you can change it globally with `\tikzset{amp symbol font}`, which has a default of 10-point (in L^AT_EX, and the corresponding one in ConT_EXt). You can change it for example with

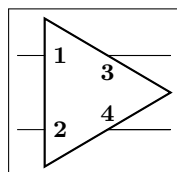
```
1 \tikzset{amp symbol font={%
2 \color{blue}\fontsize{12}{12}\selectfont\boldmath}}
```

to have plus and minus symbols that are bigger and blue.



```
1 \begin{circuitikz}[]
2   % change in this circuit only
3   \tikzset{amp symbol font={\color{blue}\small\boldmath}}
4   % local change
5   \draw (0,2.2) node[op amp, amp plus=$\oplus$]{};
6   \draw (0,0) node[op amp]{};
7   % from now on...
8   \ctikzset{amplifiers/plus={\oplus}}
9   \ctikzset{amplifiers/minus={\ominus}}
10  \draw (0,-2.2) node[fd op amp]{};
11 \end{circuitikz}
```

If you want different symbols for input and output you can use a null symbol and put them manually using the border anchors.

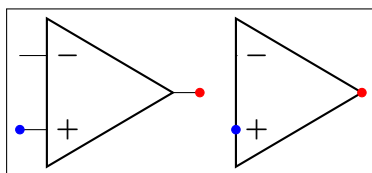


```
1 \begin{circuitikz}[]
2   \ctikzset{amplifiers/plus={}}
3   \ctikzset{amplifiers/minus={}}
4   \draw (0,0) node[fd op amp](A){};
5   \node [font=\small\bfseries, right] at(A.bin up) {1};
6   \node [font=\small\bfseries, right] at(A.bin down) {2};
7   \node [font=\small\bfseries, below] at(A.bout up) {3};
8   \node [font=\small\bfseries, above] at(A.bout down) {4};
9 \end{circuitikz}
```

4.20.2.3 Input and output pins length. The length of the wires that extends outside the main amplifier shape is not easily changed globally. You can use a trick⁷⁰ though if you want to remove them completely: the size of an amplifier (included the pins) is set by the `circuitikz` key `tripoles/amplifier style/width` and the size of the body of the amplifier, relative to it, is set by the key `tripoles/amplifier style/port width`. Making the latter equal to one will set the length of the pin to zero; if you want to maintain the same aspect ratio of the shape you need to compensate with the width.

For example, for the normal operational amplifier the key `tripoles/op amp/width` defaults to 1.7 and `tripoles/op amp/port width` is 0.7 (you need to peek that values in the source file `pgfcirctripoles.tex`). So you can do this:

⁷⁰See the discussion with [user @erwindenboer on GitHub](#); notice that this method is using internal keys and can stop working in the future.

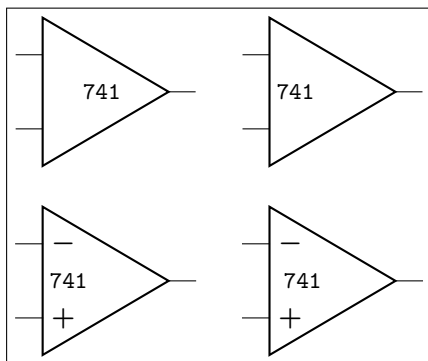


```

1 \begin{tikzpicture}[]
2   \draw (0,0) node[op amp](A){};
3   \ctikzset{tripoles/op amp/port width=1,
4     tripoles/op amp/width=1.19, % 1.7*0.7
5   }
6   \draw (2.5,0) node[op amp](B){};
7   \draw
8     (A.out) node[red, circ]{} (A.+) node[blue, circ]{}
9     (B.out) node[red, circ]{} (B.+) node[blue, circ]{};
10 \end{tikzpicture}

```

4.20.2.4 Main amplifier label. The amplifier label (given as the text of the node) is normally more or less centered in the shape (in the case of the triangular shape, it is shifted a bit to the left to *seem* visually centered); since version 1.1.0 you can move it at the left side plus a fixed offset setting the key `component text` or the style with the same name to `left`; by default the key is `center`. You can change the offset with the key `left text distance` (default 0.3em; you must use a length here). These parameters are shared with IEEE-style logic ports.



```

1 \begin{circuitikz}[]
2   \draw (0,2.5) node[plain amp]{\texttt{741}};
3   \draw (3,2.5)
4     node[plain amp, component text=left]
5     {\texttt{741}};
6   \ctikzset{component text=left}
7   \draw (0,0) node[op amp]{\texttt{741}};
8   \ctikzset{left text distance=0.6em}
9   \draw (3,0) node[op amp]{\texttt{741}};
10 \end{circuitikz}

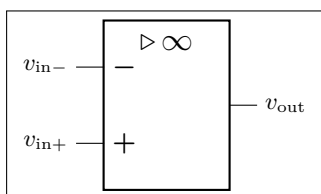
```

These keys are also used for the positioning of the labels in the label positioning of IEEE logic gates (see 4.22.2).

4.20.2.5 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):

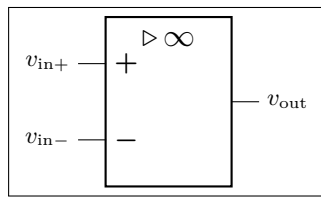


```

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

```

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

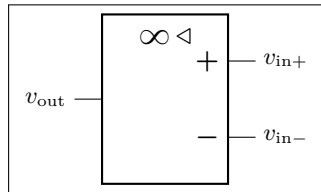


```

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

```

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

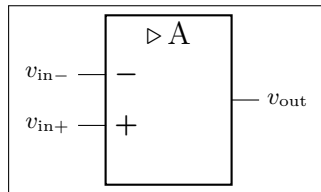


```

1 \begin{circuitikz}
2   \ctikzset{tripoles/en amp/input height=0.45}
3   \draw (0,0)node[en amp, xscale=-1, noinv input up](E){}
4     (E.out) node[left] {$v_{\mathrm{out}}$}
5     (E.-) node[right] {$v_{\mathrm{in-}}$}
6     (E.+) node[right] {$v_{\mathrm{in+}}$};
7 \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:

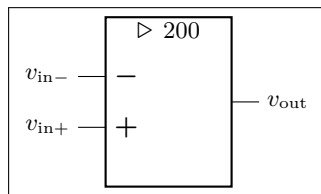


```

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

```

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



```

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

```

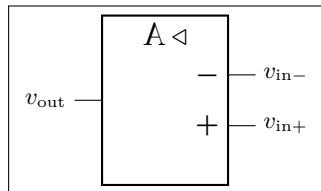
Notice two things here: the first, that `\triangleright` is enclosed in braces to remove the default spacing it has as a binary operator, and that `en amp text A` is simply a shortcut for

```

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

```

To combine flipping with a generic label you just do:

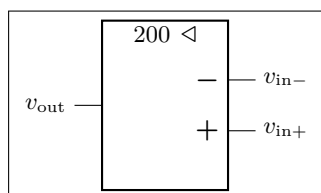


```

1 \begin{circuitikz}
2   \draw (0,0)node[en amp, xscale=-1, en amp text A](E){}
3     (E.out) node[left] {$v_{\mathrm{out}}$}
4     (E.-) node[right] {$v_{\mathrm{in-}}$}
5     (E.+) node[right] {$v_{\mathrm{in+}}$} ;
6 \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:



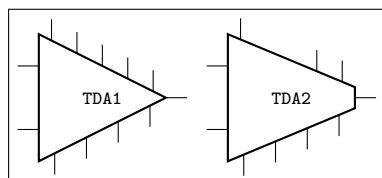
```

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

```


4.20.3 Designing your own amplifier

If you need a different kind of amplifier, you can use the `muxdemux` (see section 4.24) shape for defining one that suits your needs (you need version 1.0.0 for this to work, and 1.3.8 for the `draw only...` option).



```

1 \tikzset{tdax/.style={muxdemux,
2     muxdemux def={NL=2, Lh=3, NR=1, Rh=0,
3     NB=4, NT=5}, font=\scriptsize\ttfamily}}
4 \begin{circuitikz}
5   \draw (0,0) node[tdax](A){TDA1};
6   \draw (2.5,0) node[tdax, muxdemux def={Rh=0.5},
7     draw only top pins={1,4-5}]{TDA2};
8 \end{circuitikz}

```

4.21 Switches, buttons and jumpers

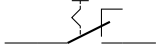

Switches and buttons come in to-style (the simple ones and the pushbuttons), and as nodes.

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 4.12).

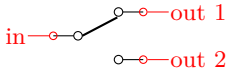
4.21.1 Traditional switches

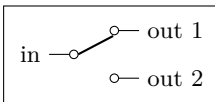
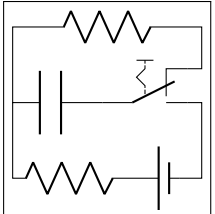
These are all of the to-style type:

	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.
	normally open push button closed: Normally open push button (in closed position), type: path-style, nodename: pushbuttoncshape. Aliases: nopbc. Class: switches.
	normally closed push button open: Normally closed push button (in open position), type: path-style, nodename: ncpushbuttonoshape. Aliases: ncpbo. Class: switches.

	toggle switch: Toggle switch, type: path-style, nodename: toggleswitchshape. Class: default.
	reed: Reed switch, type: path-style, fillable, nodename: reedshape. Class: switches.

while this is a node-style component:

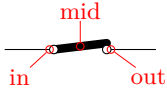
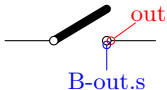


	spdt, type: node (node[spdt]{}). Class: switches.
---	--

	<pre> 1 \begin{circuitikz} \draw 2 (0,0) node[spdt] (Sw) {} 3 (Sw.in) node[left] {in} 4 (Sw.out 1) node[right] {out 1} 5 (Sw.out 2) node[right] {out 2} 6 ;\end{circuitikz} </pre>
	<pre> 1 \begin{circuitikz} \draw 2 (0,0) to[C] (1,0) to[toggle switch , n=Sw] (2.5,0) 3 -- (2.5,-1) to[battery1] (1.5,-1) to[R] (0,-1) - (0,0) 4 (Sw.out 2) - (2.5, 1) to[R] (0,1) -- (0,0) 5 ;\end{circuitikz} </pre>

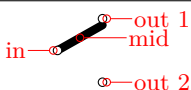
4.21.2 Cute switches

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.
---	--

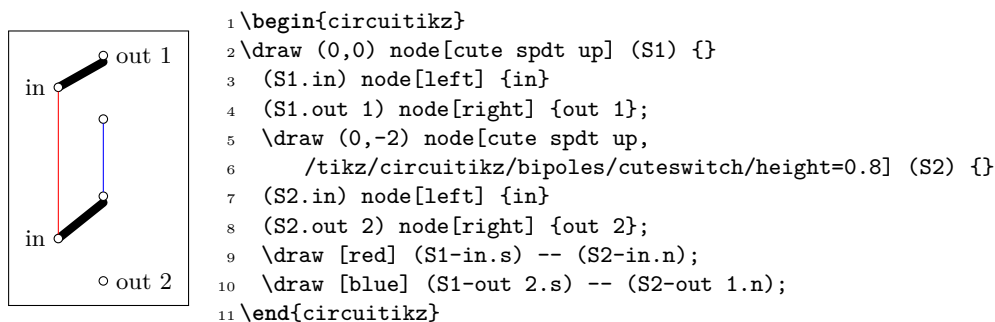
	Cute spdt mid, type: node (<code>node[cute spdt mid]{}).</code> Class: <code>switches</code> .
	Cute spdt down, type: node (<code>node[cute spdt down]{}).</code> Class: <code>switches</code> .
	Cute spdt up with arrow, type: node (<code>node[cute spdt up arrow]{}).</code> Class: <code>switches</code>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

4.21.2.1 Cute switches anchors The nodes-style switches have the following anchors:

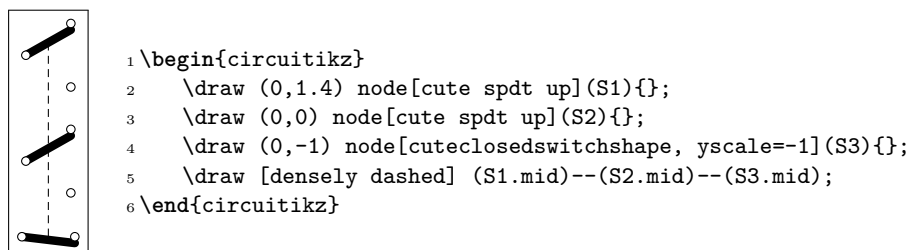


Please notice the position of the normal anchors at the border of the `ocirc` shape for the cute switches; they are thought to be compatible with an horizontal wire going out. Additionally, you have the `cin`, `cout 1` y `cout 2` which are anchors on the center of the contacts.

For more complex situations, the contact nodes are available⁷¹ using the syntax *name of the node*-in, ...-out 1 and ...-out 2, with all their anchors.

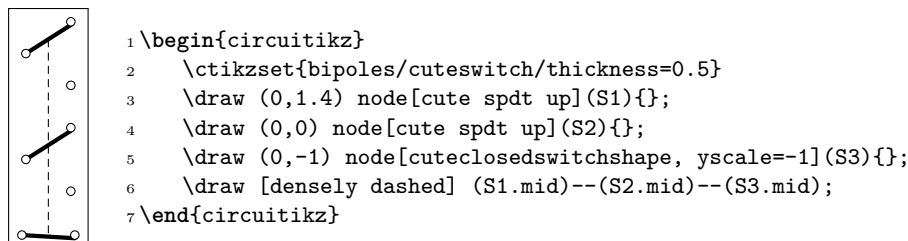


The `mid` anchor in the cute switches (both path- and node-style) can be used to combine switches to get more complex configurations:

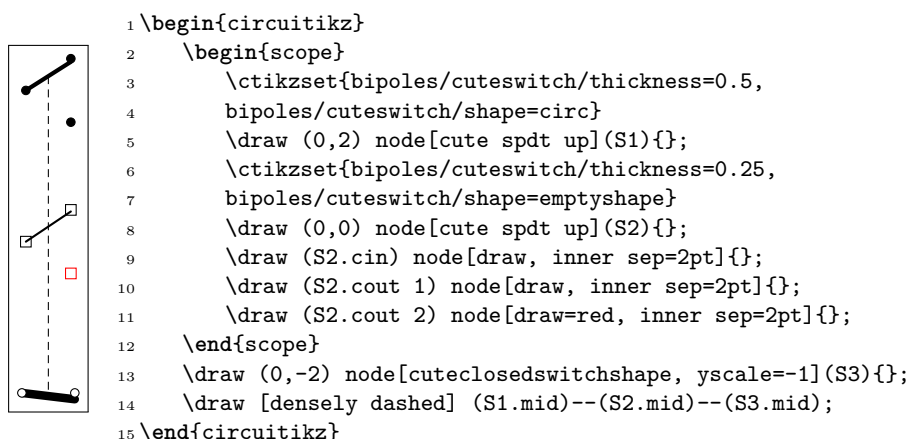


⁷¹Thanks to @marmot on tex.stackexchange.com.

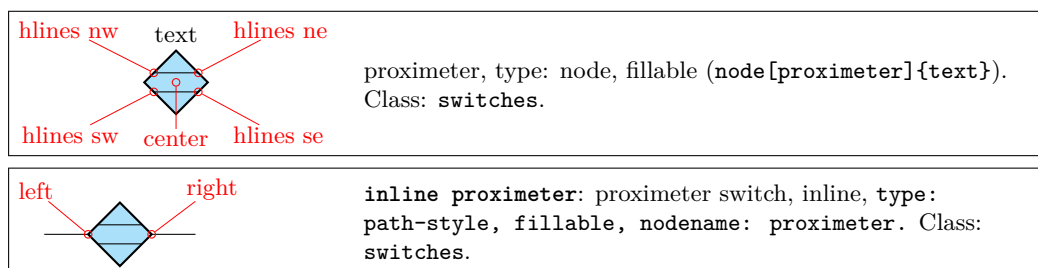
4.21.2.2 Cute switches customization You can use the key `bipoles/cuteswitch/thickness` to decide the thickness of the switch lever. The units are the diameter of the `ocirc` connector, and the default is 1.



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.



4.21.3 Proximity switches

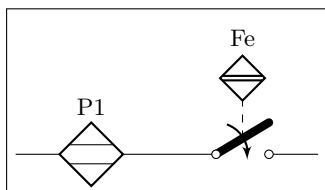


The `proximeter` shape⁷² can be used as a dipole with the `inline proximeter` variant.

It has been assigned to the `switches` class; you can adjust the (relative) thickness of the inside horizontal lines with the key `proximeter/hlines thickness` (default 0.5) and their vertical position with `proximeter/hlines position` (default 0.3). You can also change the default size of *all* proximeter symbols by changing `proximeter/width` (only safe at picture level; better set in the preamble if you need to change it). The default value is 0.3).

Notice in the following example that, as ever for node-type shape, the text is not included in the bounding box:

⁷²Suggested by [Anisio Rogerio Braga](#), implemented in v1.5.2; see also [here](#).



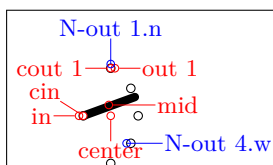
```

1 \begin{tikzpicture}
2   \tikzset{small up proxi/.style={proximeter, solid,
3     circuitikz/switches/scale=0.707,
4     circuitikz/proximeter/hlines thickness=1,
5     circuitikz/proximeter/hlines position=0.1}}
6   \draw (0,0) to[inline proximeter, l=P1] ++(2,0)
7     to[ccgsw, name=P2] ++(2,0);
8   \draw[dashed] (P2.mid) -- ++(0,0.5)
9     node[small up proxi, above](P2p){Fe}
10    (P2p.north) ++ (0,0.5); % extend bounding box
11 \end{tikzpicture}

```

4.21.4 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).



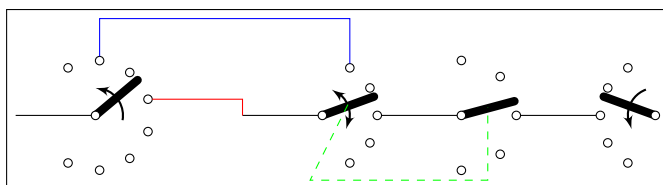
Rotary switch, type: node (`node[rotaryswitch] (N){}`). Class: `switches`.

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 from -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.

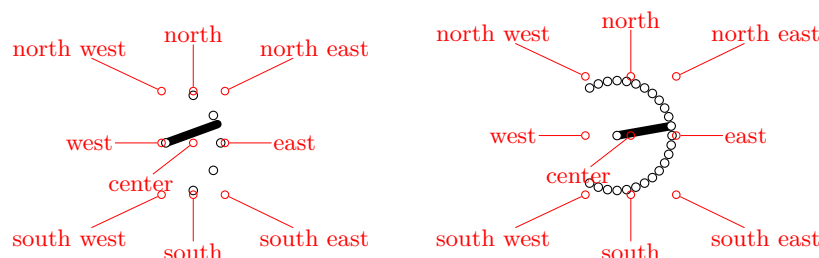


```

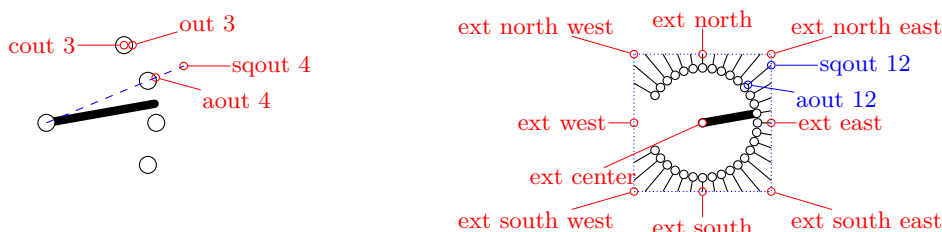
1 \begin{circuitikz}
2   \ctikzset{multipoles/rotary/arrow=both}
3   \draw (0,0) -- ++(1,0) node[rotary switch <-=8 in 120 wiper 40, anchor=in](A){};
4   \draw (3,0) -- ++(1,0) node[rotary switch, anchor=in](B){}; % default values
5   \draw[red] (A.out 4) -| (3,0);
6   \draw[blue] (A.out 2.n) -- ++(0,0.5)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

```

4.21.4.1 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.



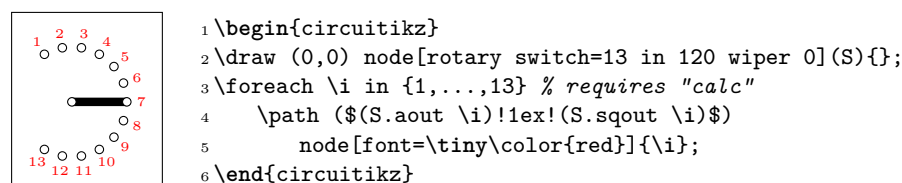
In addition to the anchors they have in common with the cute switches, the rotary switch has the so called “angled” anchors and the “external square anchors”. *Angled anchors*, called `aout 1`, `aout 2` and so forth, are anchors placed on the output poles at the same angle as the imaginary lines coming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



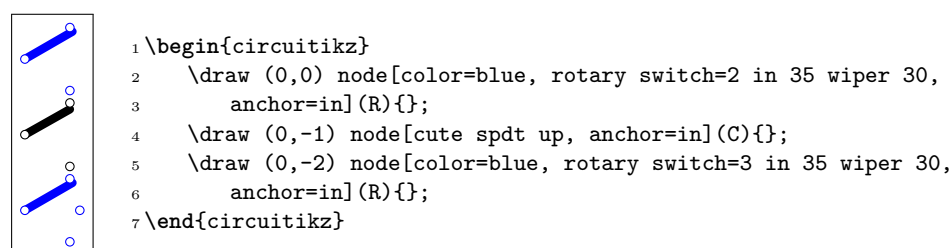
The code for the diagram at the left, above, without the markings for the anchors, is:

```
1 \begin{circuitikz}
2   \draw (8,0) node[rotary switch --=31 in 150 wiper 10](D){};
3   \foreach \i in {1,...,31} \draw (D.sqout \i) -- (D.aout \i);
4   \draw[blue, densely dotted] (D.ext north west) rectangle (D.ext south east);
5 \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:



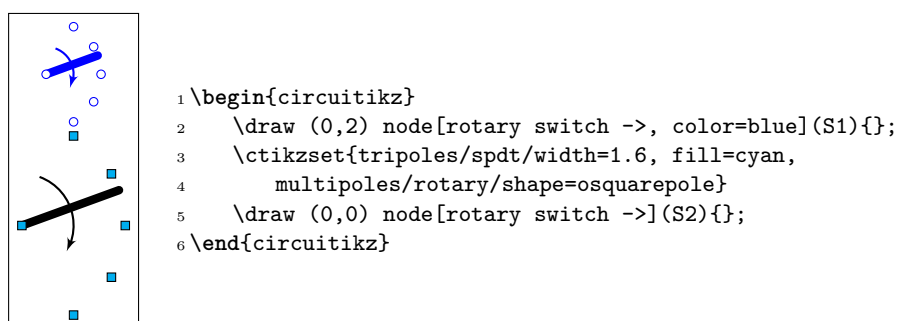
Finally, notice that the value of width for the rotary switches is taken from the one for the “cute switches” which in turn is taken from the width of traditional `spdt` switch, so that they match (notice that the “center” anchor is better centered in the rotary switch, so you have to explicitly align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



4.21.4.2 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).



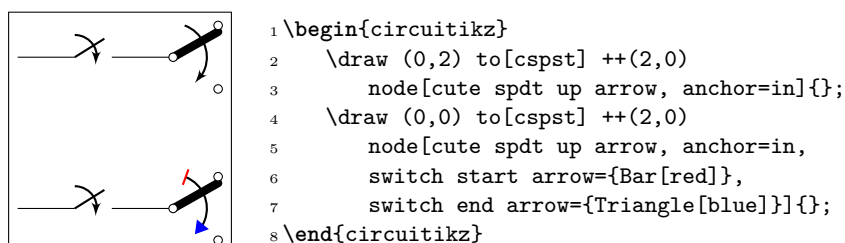
Finally, the size can be changed using the parameter `tripoles/spdt/width` (default 0.85).



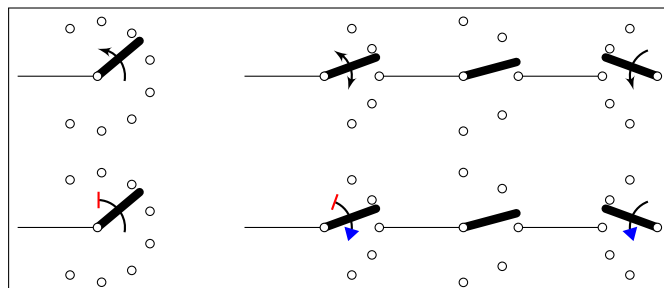
4.21.5 Switch arrows

You can change the arrow tips used in all switches (traditional and “cute”) with the key `switch end arrow` (by default the key is the word “default” to obtain the default arrow, which is `latexslim`). Also you can change the start arrow with the corresponding `switchable start arrow` or `wiper start arrow` (the default value “default” is equivalent to {}, which means no arrow). The keys are settable with `\ctikzset` as with `\tikzset` (to ease their usage in nodes).

You can change that globally or locally, as ever. The tip specification is the one you can find in the TikZ manual (“Arrow Tip Specifications”).



4.21.5.1 Rotary switch arrows. You can change the rotary switch arrow shape in the same way as you change the ones in regular switches. Notice however that if you set either `switch end arrow` or `switch start arrow` they will be followed only if you have set both arrows with `<->` or equivalent, otherwise just one will be used.



```

1 \begin{circuitikz}
2 \ctikzset{multipoles/rotary/arrow=both}
3 \draw (0,0) -- ++(1,0) node[rotary switch <--=8 in 120 wiper 40, anchor=in](A){};
4 \draw (3,0) -- ++(1,0) node[rotary switch, anchor=in](B){}; % default values
5 \draw (B.out 3) -- ++(1,0) node[rotary switch --=5 in 90 wiper 15, anchor=in](C){};
6 \draw (C.out 3) -- ++(1,0) node[rotary switch ->, xscale=-1, anchor=out 3](D){};
7 \ctikzset{switch end arrow={Triangle[blue]}}
8 \ctikzset{switch start arrow={Bar[red]}}
9 \begin{scope}[yshift=-2cm]
10 \draw (0,0) -- ++(1,0) node[rotary switch <--=8 in 120 wiper 40, anchor=in](A){};
11 \draw (3,0) -- ++(1,0) node[rotary switch, anchor=in](B){}; % default values
12 \draw (B.out 3) -- ++(1,0) node[rotary switch --=5 in 90 wiper 15, anchor=in](C){};
13 \draw (C.out 3) -- ++(1,0) node[rotary switch ->, xscale=-1, anchor=out 3](D){};
14 \end{scope}
15 \end{circuitikz}

```

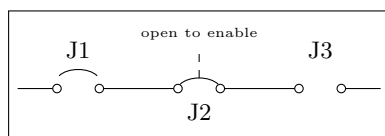
4.21.6 Jumpers

You can think of jumpers like a kind of switches (they have the same function, just the way of operating them is different). CircuitikZ has two types of jumper symbols available, the simple ones and the three pins (or two-ways) ones.

4.21.6.1 Simple jumpers. These are the most common ones. They come in three variations, bare, open and closed.

	<p>bare jumper: Bare jumper, type: path-style, fillable, name=B, nodename: bjumbershape. Class: switches.</p>
	<p>open jumper: Open jumper, type: path-style, fillable, name=B, nodename: ojumbershape. Class: switches.</p>
	<p>closed jumper: Closed jumper, type: path-style, fillable, name=B, nodename: cjumpershape. Class: switches.</p>

The `top arc` anchor can be used to locate the position of the top of the wire (when present). In bare jumper, the anchor is located in the middle of the connectors gap.



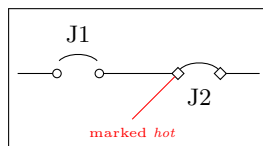
```

1 \begin{circuitikz}[scale=0.8]
2   \draw (0,0) to[open jumper, l=J1] ++(2,0)
3     to[closed jumper, l_=J2, name=J2] ++(2,0)
4     to[bare jumper=J3] ++(2,0);
5   \draw [dashed] (J2.top arc) -- ++(0,0.5)
6     node[above] {\tiny open to enable};
7 \end{circuitikz}

```

Similarly to switches, you have access to the subnodes representing the contacts, to be able to draw wires at different angles.

The kind of poles used in the diagram can be changed with the `\ctikzset` key `bipoles/jumpers/shape`.



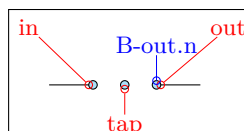
```

1 \begin{circuitikz}[scale=0.8]
2   \draw (0,0) to[open jumper, l=J1, name=J1] ++(2,0)
3     to[closed jumper, l_=J2, name=J2,
4       bipoles/jumper/shape=odiamondpole] ++(2,0);
5   \draw [red] (J2-in.-135) -- ++(-135:1)
6     node[font=\tiny, below]{marked \emph{hot}};
7 \end{circuitikz}

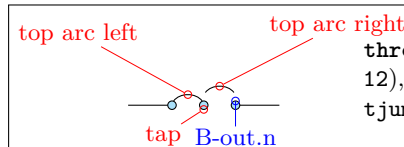
```

4.21.6.2 Two-ways (three-pins) jumpers. In this case, the symbol represent two-ways jumpers (normally, three pins that can be connected in a couple of ways).

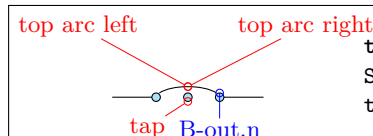
To maintain flexibility, every possible combination of bare, open or closed is available; but to avoid having to define too much different bipoles, a different approach is used here. You have to specify the style using a different key, namely `tjumper connections`.



three-pins jumper: Three-pins jumper (see later for connections), type: path-style, fillable, name=B, nodename: tjumpershape. Class: switches.



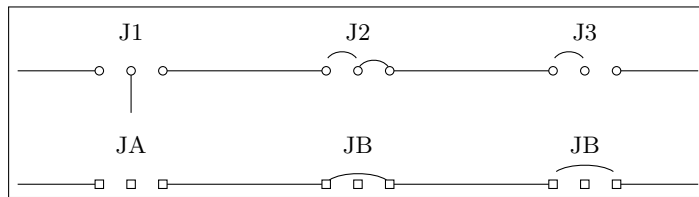
three-pins jumper: Three-pins jumper (connections 12), type: path-style, fillable, name=B, nodename: tjumpershape. Class: switches.



three-pins jumper: Three-pins jumper (connections S1), type: path-style, fillable, name=B, nodename: tjumpershape. Class: switches.

The option is used as shown in the following example, or by setting the key using `\ctikzset`. The value **must** be two characters, either two numbers (where 0 means “bare”, 1 means “open”, and 2 means “closed”) or the letter S (for “span”) and one number. In the latter case, the arc will connect the first and last pole⁷³

⁷³Although really I never saw an example of this use...You never know.



```

1 \begin{circuitikz}
2   \draw (0,1.5) to[three-pins jumper, l=J1, name=T] ++(3,0)
3     to[three-pins jumper, tjumper connections=21, l=J2] ++(3,0)
4     to[three-pins jumper, tjumper connections=20, l=J3] ++(3,0);
5   \draw (T.tap) -- ++(0,-0.5);
6   \ctikzset{bipoles/jumper/shape=osquarepole}
7   \draw (0,0) to[three-pins jumper, l=JA, name=T] ++(3,0)
8     to[three-pins jumper, tjumper connections=S1, l=JB] ++(3,0)
9     to[three-pins jumper, tjumper connections=S2, l=JB] ++(3,0);
10 \end{circuitikz}

```

4.21.7 Solder jumpers.

Solder jumpers are basically jumpers that can be closed or opened on the printed circuit board. Although electrically they behave exactly as jumpers, these are thought to change configurations in a more stable way (to change them from their default connection you have to use a cutter and/or a soldering iron).

	open solder jumper: Open solder jumper, type: path-style, nodename: osjumpershape. Class: switches.
	closed solder jumper: Closed solder jumper, type: path-style, nodename: csjumpershape. Class: switches.
	open double solder jumper: Open double solder jumper, type: path-style, nodename: odsjumpershape. Class: switches.
	left double solder jumper: Left double solder jumper, type: path-style, nodename: ldsjumpershape. Class: switches.
	right double solder jumper: Right double solder jumper, type: path-style, nodename: rdsjumpershape. Class: switches.
	closed double solder jumper: Closed double solder jumper, type: path-style, nodename: cdsjumpershape. Class: switches.

```

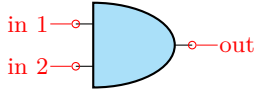
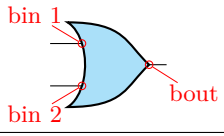
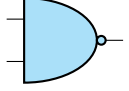
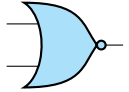
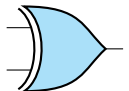
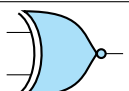
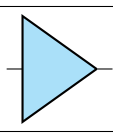
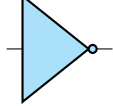
1 \begin{circuitikz}[scale=0.8]
2   \draw (0,0) to[open solder jumper, l=J1] ++(2,0)
3     to[closed solder jumper, l=J2, name=J2] ++(2,0)
4     to[right double solder jumper, l=J3,
5       name=J3] ++(2,0);
6   \draw (J3.tap down) -- ++(0,-1) node[ocirc]{};
7 \end{circuitikz}

```

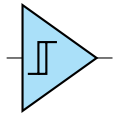
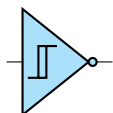
4.22 Logic gates

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

4.22.1 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.

There is no “european” version of the following symbols; for now they are used both in **american** and **european** styles, but it may change in the future.

	Non-Inverting Schmitt trigger, type: node, fillable (node[schmitt]{}). Class: logic ports.
	Inverting Schmitt trigger, type: node, fillable (node[invschmitt]{}). Class: logic ports.

4.22.2 IEEE logic gates

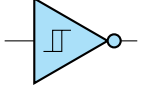
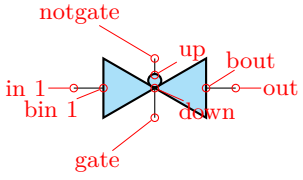
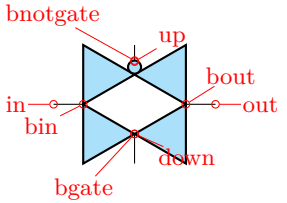

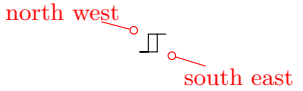
In addition to the legacy ports, since release 1.1.0, logic ports following the recommended geometry of distinctive-shape symbols in IEEE Std 91a-1991 Annex A (Recommended symbol proportions) are also available⁷⁴.

These ports are completely independent from the legacy set (either **american** or **european**); they are not enabled by default because the relative size of the ports is very different from the legacy ones, and that will disrupt every schematic (especially if drawn with absolute coordinate). If you want to use them as default, you can use the command `\ctikzset{logic ports=ieee}` and by default the shapes **and** **port**, **or** **port** and so on will be the IEEE standard ones.


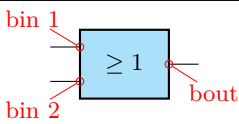

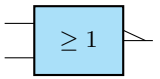
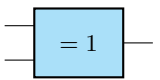
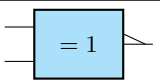

The transmission gate (also known as “bowtie”) components are not described in the IEEE standard, so they are simply inspired by the other IEEE ports — this is why their name is prefixed by **ieee** and not by **ieeestd**. They are aliased to **tgate** and **double tgate** though, and it is recommended to use those names (maybe in the future there will be **american ports** and/or **european ports** versions available).


	IEEE standard “and” port, type: node, fillable (<code>node[ieeestd and port]{}).</code> Class: logic ports.
	IEEE standard “nand” port, type: node, fillable (<code>node[ieeestd nand port]{}).</code> Class: logic ports.
	IEEE standard “or” port, type: node, fillable (<code>node[ieeestd or port]{}).</code> Class: logic ports.
	IEEE standard “nor” port, type: node, fillable (<code>node[ieeestd nor port](N){}.</code> Class: logic ports.
	IEEE standard “xor” port, type: node, fillable (<code>node[ieeestd xor port]{}).</code> Class: logic ports.
	IEEE standard “xnor” port, type: node, fillable (<code>node[ieeestd xnor port]{}).</code> Class: logic ports.
	IEEE standard buffer port, type: node, fillable (<code>node[ieeestd buffer port]{}).</code> Class: logic ports.
	IEEE standard “not” port, type: node, fillable (<code>node[ieeestd not port]{}).</code> Class: logic ports.
	Schmitt port matched to IEEE standard ports, type: node, fillable (<code>node[ieeestd schmitt port]{}).</code> Class: logic ports.

⁷⁴Thanks to Jason for proposing it and digging out the info, see this [GitHub issue](#).

	Inverting Schmitt port matched to IEEE standard ports, type: node, fillable (<code>node[ieeestd invschmitt port]{}).</code> Class: logic ports.
	IEEE style transmission gate, type: node, fillable (<code>node[ieee tgate]{}).</code> Class: logic ports.
	IEEE style double transmission gate, type: node, fillable (<code>node[ieee double tgate]{}).</code> Class: logic ports.
	Inverting dot for IEEE ports, type: node, fillable (<code>node[notcirc]{}).</code> Class: logic ports.
	Schmitt symbol to add to input pins if needed, type: node, fillable (<code>node[schmitt symbol]{}).</code> Class: logic ports.

4.22.3 European Logic gates

	European AND port, type: node, fillable (<code>node[european and port]{}).</code> Class: logic ports.
	European OR port, type: node, fillable (<code>node[european or port]{}).</code> Class: logic ports.
	European NAND port, type: node, fillable (<code>node[european nand port]{}).</code> Class: logic ports.
	European NOR port, type: node, fillable (<code>node[european nor port]{}).</code> Class: logic ports.
	European XOR port, type: node, fillable (<code>node[european xor port]{}).</code> Class: logic ports.
	European XNOR port, type: node, fillable (<code>node[european xnor port]{}).</code> Class: logic ports.
	European BUFFER port, type: node, fillable (<code>node[european buffer port]{}).</code> Class: logic ports.

	European NOT port, type: node, fillable (node[<code>europaean not port</code>]{}). Class: logic ports.
---	--

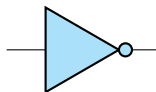
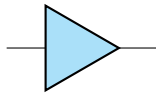
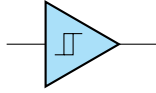
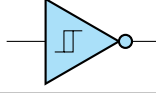
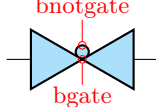
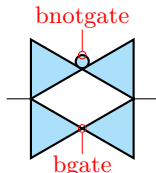
If (default behaviour) `americanports` option is active (or the style [`american ports`] is used), the shorthands `and port`, `or port`, `buffer port`, `nand port`, `nor port`, `not port`, `xor port`, `xnor port`, `schmitt port` and `invschmitt port` are equivalent to the american version of the respective logic port.

If otherwise `europaeanports` option is active (or the style [`europaean ports`] is used), the shorthands `and port`, `or port`, `buffer port`, `nand port`, `nor port`, `not port`, `xor port`, `xnor port` are equivalent to the european version of the respective logic port; `schmitt port` and `invschmitt port` are the same as in `american ports` style.

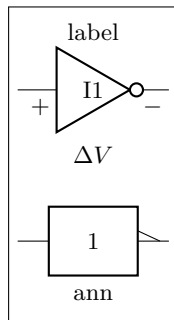
Finally, for version 1.1.0 and up, you can use the style `ieee ports` to set the shorthands to the set of `ieeestd` ports. (There is no global option for this).

4.22.4 Path-style logic ports

The one-input, one-output ports have a handy path-style equivalent; they are the following:

	<code>inline not</code> : “not” logic port, type: <code>path-style</code> , fillable, nodename: <code>not port</code> . Class: logic ports.
	<code>inline buffer</code> : “buffer” logic port, type: <code>path-style</code> , fillable, nodename: <code>buffer port</code> . Class: logic ports.
	<code>inline schmitt</code> : Schmitt logic port, type: <code>path-style</code> , fillable, nodename: <code>schmitt port</code> . Class: logic ports.
	<code>inline invschmitt</code> : Inverting Schmitt logic port, type: <code>path-style</code> , fillable, nodename: <code>invschmitt port</code> . Class: logic ports.
	<code>inline tgate</code> : transmission gate, type: <code>path-style</code> , fillable, nodename: <code>tgate</code> . Class: logic ports.
	<code>inline double tgate</code> : double transmission gate, type: <code>path-style</code> , fillable, nodename: <code>double tgate</code> . Class: logic ports.

Those ports follow the current selected style, although you can change it on the fly (even if it does not have a lot of sense); you can apply labels, annotations and (again, not a lot of sense) voltages to them. The assigned value is typeset as if it were the main text of the node.

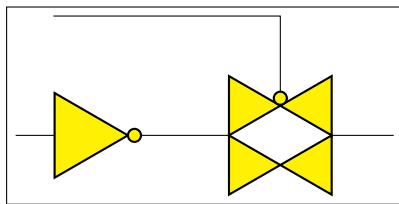


```

1 \begin{circuitikz}[american]
2   \ctikzset{logic ports=ieee}
3   \draw (0,0) to[inline not=I1, l=label, v=\Delta V] ++(2,0);
4   \draw (0,-2) to[inline not, a=ann, european ports] ++(2,0);
5 \end{circuitikz}

```

Notice that in the inline version the leading pins are not drawn, so in the case of the transmission gates you have to use the border pins to connect the gates.



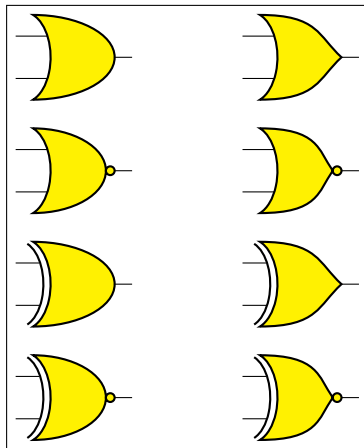
```

1 \begin{circuitikz}[ ]
2   \ctikzset{logic ports=ieee,
3     logic ports/fill=yellow}
4   \draw (0,0) to[inline not] ++(2,0)
5     to[inline double tgate, name=P] ++(3,0)
6     (P.bnotgate) |- ++(-3,1);
7 \end{circuitikz}

```

4.22.5 American ports usage

Since version 1.0.0, the default shape of the family of american “or” ports has changed to a more “pointy” one, for better distinguish them from the “and”-type ports. You can still go back to the previous aspect with the key `american or shape` that can be set to `pointy` or `roundy`. The `legacy` style will enact the old, roundy style also.



```

1 \begin{circuitikz}[
2   american]
3   % legacy shapes
4   \ctikzset{american or shape=roundy}
5   \ctikzset{logic ports/fill=yellow}
6   \node [or port](O1) at (0,0) {};
7   \node [nor port](O2) at (0,-1.5) {};
8   \node [xor port](O3) at (0,-3) {};
9   \node [xnor port](O4) at (0,-4.5) {};
10  \begin{scope}[xshift=3cm]
11    % new shapes
12    \ctikzset{american or shape=pointy}
13    \node [or port](O1) at (0,0) {};
14    \node [nor port](O2) at (0,-1.5) {};
15    \node [xor port](O3) at (0,-3) {};
16    \node [xnor port](O4) at (0,-4.5) {};
17  \end{scope}
18 \end{circuitikz}

```

4.22.5.1 American 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).

It is possible to change height and width of the logic ports using the parameters `tripoles/american type port/` plus `width` or `height`:

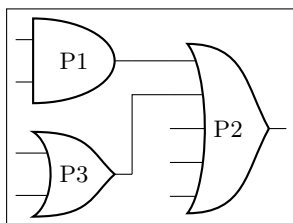


```

1 \tikz \draw (0,0) node[nand port] {}; \par
2 \ctikzset{tripoles/american nand port/input height=.2}
3 \ctikzset{tripoles/american nand port/port width=.4}
4 \ctikzset{tripoles/thickness=4}
5 \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` :

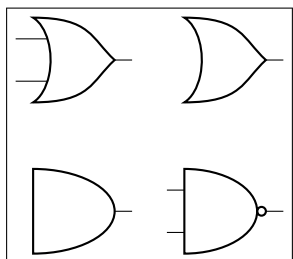


```

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

```

You can suppress the drawing of the logic ports input leads by using the boolean key `logic ports draw input leads` (default `true`) or, locally, with the style `no inputs leads` (that can be reverted with `input leads`), like in the following example. The anchors do not change and you have to take responsibility to make the connection to the “border”-anchors.

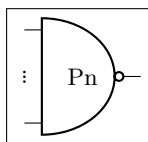


```

1 \begin{circuitikz}
2 \node [or port](O1) at (0,2) {};
3 \node [or port, no input leads](O1) at (2,2) {};
4 \ctikzset{logic ports draw input leads=false}
5 \node [and port](O1) at (0,0) {};
6 \node [nand port, input leads](O1) at (2,0) {};
7 \end{circuitikz}

```

This is useful if you need to draw a generic port, like the one following here:



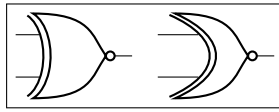
```

1 \begin{circuitikz}
2 \ctikzset{tripoles/american nand port/height=1.6}
3 \draw (0,0)
4 node[american nand port,
5 circuitikz/tripoles/american nand port/height=1.1,
6 number inputs=5, no input leads,
7 ] (B) {Pn};
8 \draw (B.in 1) -- (B.bin 1) (B.in 5) -- (B.bin 5);
9 \node[rotate=90] at (B.in 3) {\dots};
10 \end{circuitikz}

```

In an analogous manner, there is a setting `logic ports draw output leads` (and a corresponding style `no output leads`) that suppresses the drawing of the output lead. A shortcut boolean key `logic ports draw leads` will suppress or enable all leads (the corresponding styles are `no leads` and `all leads`).

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 goes “into” the shape, default 0.3) and `angle` (the angle at which the base starts, default 70).



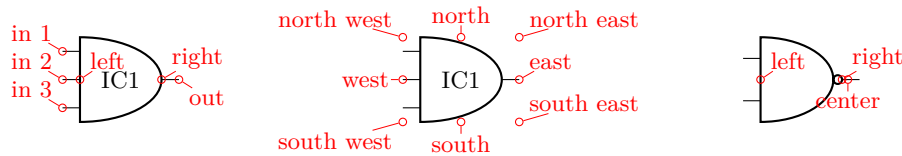
```

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

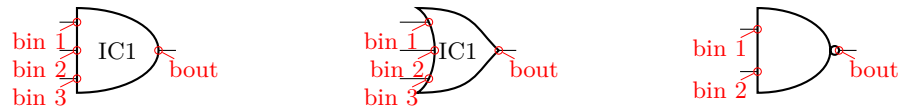
```

4.22.5.2 American logic port anchors

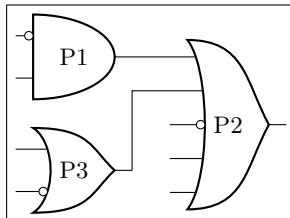
These are the anchors for logic ports:



You also have “border pin anchors”:



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

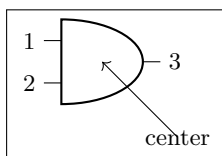


```

1\begin{circuitikz}
2\draw (0,3) node[american and port] (A) {P1};
3\node at (A.bin 1) [ocirc, left]{} ;
4\begin{scope}
5\ctikzset{tripoles/american or port/height=1.6}
6\draw (A.out) -- ++(0.5,0) node[american or port,
7number inputs=5, anchor=in 1] (B) {P2};
8\node at (B.bin 3) [ocirc, left]{} ;
9\end{scope}
10\draw (0,1.5) node[american or port] (C) {P3};
11\node at (C.bin 2) [ocirc, left]{} ;
12\draw (C.out) |- (B.in 2);
13\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}`:



```

1\begin{circuitikz}
2\ctikzset{logic ports origin=center}
3\draw (0,0) node[and port] (myand) {}
4(myand.in 1) node[anchor=east] {1}
5(myand.in 2) node[anchor=east] {2}
6(myand.out) node[anchor=west] {3};
7\draw[<-] (myand.center) -- ++(1,-1)
8node{center};
9\end{circuitikz}

```

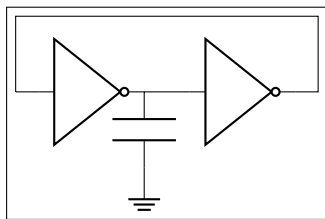


```

1 \begin{circuitikz} \draw
2   (0,2) node[and port] (myand1) {}
3   (0,0) node[and port] (myand2) {}
4   (2,1) node[xnor port] (myxnor) {}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 \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`):

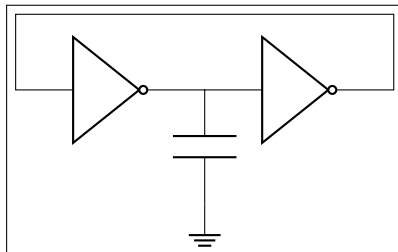


```

1 \begin{circuitikz} \draw
2   (1,0) node[not port] (not1) {}
3   (3,0) node[not port] (not2) {}
4   (0,0) -- (not1.in)
5   (not2.in) -- (not1.out)
6   ++(0,-1) node[ground] {} to[C] (not1.out)
7   (not2.out) -| (4,1) -| (0,0)
8 \end{circuitikz}

```

This last circuit could be drawn also (and probably in a more natural manner) using the path-style components:



```

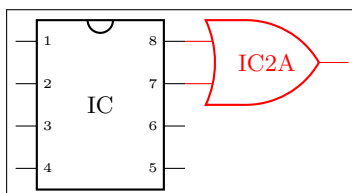
1 \begin{circuitikz}[american]
2   \draw (0,0) node[ground]{} to[C] ++(0,1.5)
3   coordinate(c)
4   to[inline not] ++(2.5,0) -- ++(0,1)
5   -| ++(-5,-1)
6   to[inline not] (c);
7 \end{circuitikz}

```

4.22.6 IEEE logic gates usage.

The rest of this section will assume you have issued the command `\ctikzset{logic ports=ieee}`, so that the short form of the names is used.

IEEE standard logic gates have a basic difference with the legacy ones: the proportions of their shapes do not change when you change the size, so you can't have a "tall" port or a "squatty" one. The two-inputs gates, by default, have their default size designed so that they match the chips component (see 4.25).

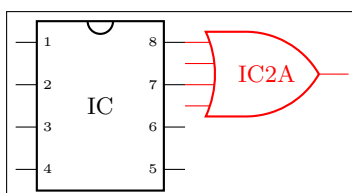


```

1 \begin{circuitikz}
2   \draw (0,0) node[dipchip](C){IC} (C.pin 8)
3   node[or port, anchor=in 1,
4   color=red] (A){IC2A};
5 \end{circuitikz}

```

If you need, say, a 4-inputs port, the port will look like this:

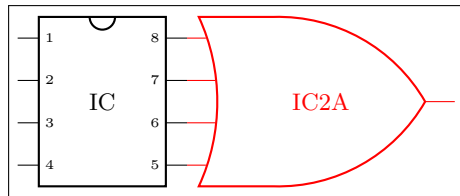


```

1 \begin{circuitikz}
2   \draw (0,0) node[dipchip](C){IC} (C.pin 8)
3   node[or port, anchor=in 1, number inputs=4,
4   color=red] (A){IC2A};
5 \end{circuitikz}

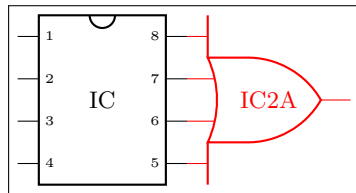
```

...and in this case it is clear that it does not match. With standard ports, there are two possibilities. The first one is to scale the port; if you set the port height so that it has the same size (see “IEEE logic gates customization” below for details) as the number of ports, they will match again.



```
1 \begin{circuitikz}
2   \draw (0,0) node[dipchip](C){IC} (C.pin 8)
3     node[or port, anchor=in 1,
4       number inputs=4,
5       circuitikz/ieeestd ports/height=4,
6       color=red](A){IC2A};
7 \end{circuitikz}
```

But then the size of the port is quite “unusual”. The solution in technical literature is to use what we can call a “rack” for the inputs; basically, only a certain number of pins are kept on the port, and the others are put on an extended input line.

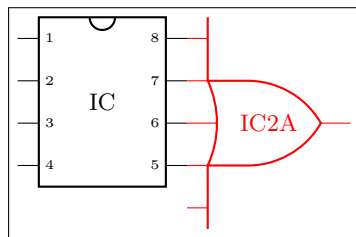


```
1 \begin{circuitikz}
2   \draw (0,0) node[dipchip](C){IC} (C.pin 8)
3     node[or port, anchor=in 1,
4       number inputs=4,
5       inner inputs=2,
6       color=red](A){IC2A};
7 \end{circuitikz}
```

When using the `inner inputs` key, keep in mind the rule of thumbs:

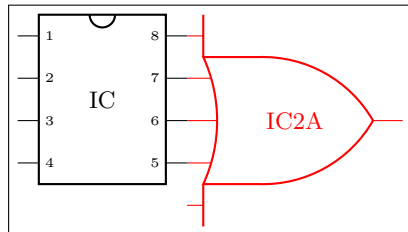
- the distance between the pins is matched with the chip ones when the `inner inputs` match the `/ieeestd ports/height` key;
- when the number of pins in the rack is odd, the result is often quite ugly, so try to avoid it.

For example, look at the following example; given that we are asking an odd number of pins on the rack, some of the inputs are drawn on the port’s border, resulting in a less-than-ideal diagram.



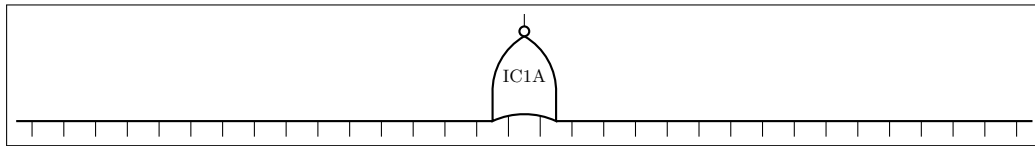
```
1 \begin{circuitikz}
2   \draw (0,0) node[dipchip](C){IC} (C.pin 8)
3     node[or port, anchor=in 1,
4       number inputs=5,
5       inner inputs=2,
6       color=red](A){IC2A};
7 \end{circuitikz}
```

In this case, if you don’t like the solution, the better approach is to let the gate grow a bit.



```
1 \begin{circuitikz}
2   \draw (0,0) node[dipchip](C){IC} (C.pin 8)
3     node[or port, anchor=in 1,
4       number inputs=5,
5       inner inputs=3,
6       circuitikz/ieeestd ports/height=3,
7       color=red](A){IC2A};
8 \end{circuitikz}
```

The good thing about the rack mechanism is that you can have quite big ports without problems.

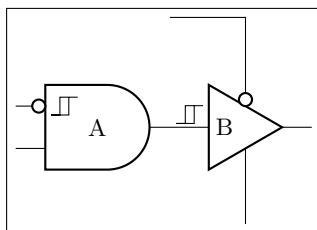


```

1 \begin{circuitikz}[scale=0.75, transform shape]
2   \draw node[nor port, number inputs=32, inner inputs=2,
3             rotate=90] (A) {\rotatebox{-90}{IC1A}};
4 \end{circuitikz}

```

You can use the additional elements (the `notcirc` and the `schmitt` symbol to obtain circuits like the following ones (well, a bit of a mix of conventions, but...):



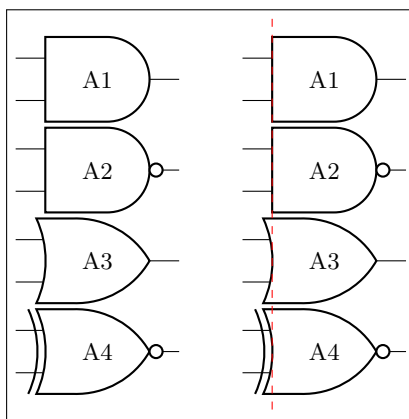
```

1 \begin{circuitikz}
2   \draw (0,0) node[and port] (A){A} (A.out)
3     node[buffer port, anchor=in,
4         component text=left] (B){B} (B.bin)
5     node[schmitt symbol, above left] {}
6     (A.bin 1) node[schmitt symbol, right] {};
7   \node [notcirc, left] at (A.bin 1) {};
8   \node [notcirc, above] (C) at (B.up) {};
9   \draw (C.north) |- ++(-1,1) (B.down) --++(0,-1);
10 \end{circuitikz}

```

Notice the key `component text=left` that moves the label near to the left border of the component. There is also a `\ctikzset{component text=left}` if you prefer to have it as a default for all the IEEE ports.⁷⁵

4.22.6.1 Stacking and aligning IEEE standard gates. The standard gates are designed so that they stack up nicely when positioned using the external leads as anchors. Notice that the ports **do** have different sizes, but the leads lengths are designed to counter the differences.



```

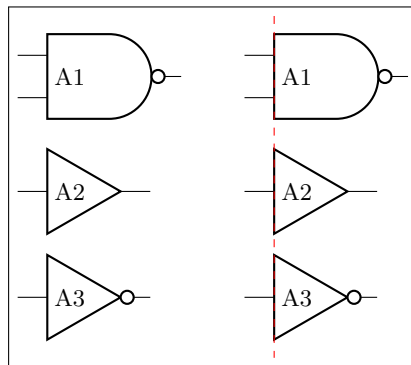
1 \begin{circuitikz}
2   \draw
3     (0,0) node[and port, anchor=in 1]{A1}
4     (0,-1.2) node[nand port, anchor=in 1]{A2}
5     (0,-2.4) node[or port, anchor=in 1]{A3}
6     (0,-3.6) node[xnor port, anchor=in 1]{A4};
7   \draw
8     (3,0) node[and port, anchor=in 1] (A1){A1}
9     (3,-1.2) node[nand port, anchor=in 1] (A2){A2}
10    (3,-2.4) node[or port, anchor=in 1] (A3){A3}
11    (3,-3.6) node[xnor port, anchor=in 1] (A4){A4};
12   \draw[red, dashed] ([yshift=0.8cm]A1.body left)
13     -- ([yshift=-0.8cm]A4.body left);
14 \end{circuitikz}

```

The length of the external leads can be changed by the user, but notice that if you use a too small value you can jeopardize that property.

The single input ports (`not port`, `buffer port` and their Schmitt equivalent) are smaller than the six standard ports, so they are not kept aligned by default; they just have the same distance at the input side. For the `not` ports, the `left` position of the text results often in a better look (the centered text in the triangle seems to be much more at the right).

⁷⁵You can use the same key with amplifiers, too.



```

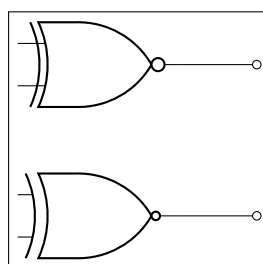
1 \begin{circuitikz}
2   \ctikzset{component text=left}
3   \draw (0,0) node[nand port, anchor=in 1]{A1}
4     (0,-1.8) node[buffer port, anchor=in 1]{A2}
5     (0,-3.2) node[not port, anchor=in 1]{A3};
6   \draw (3,0) node[nand port, anchor=in 1](A1){A1}
7     (3,-1.8) node[buffer port, anchor=in 1]{A2}
8     (3,-3.2) node[not port, anchor=in 1](A3){A3};
9   \draw[red, dashed]([yshift=0.8cm]A1.body left)
10     -- ([yshift=-0.8cm]A3.body left);
11 \end{circuitikz}

```

4.22.6.2 IEEE standard ports customization There are several parameters that can be used to customize the IEEE standard ports, although less than the ones in the legacy american ones — the basic shape is set to follow the IEEE recommendation. The basic parameters are shown in the following table, and they can be set via `\ctikzset{ieeestd ports/...}`

key	default	description
<code>baselen</code>	0.4	the basic length for every dimension, as a fraction of the (scaled) resistor length
<code>height</code>	2	the height of the port, in term of <code>baselen</code> . Pin distance is given by this parameter divided by the inner pins.
<code>pin length</code>	0.7	length of the external pin leads that are drawn with the port. This length is always calculated starting from the inner body of the shape.
<code>not radius</code>	0.154	radius of the “not circle” added to the negated-output ports. The default value is the IEEE recommended one.
<code>xor bar distance</code>	0.192	distance of the detached input shape in <code>xor</code> and <code>xnor</code> ports. The default value is the IEEE recommended one.
<code>xor leads in</code>	1	If set to 0, there will be no leads drawn between the detached input line and the body in the <code>xor</code> and <code>xnor</code> ports. IEEE recommends 1 here.
<code>schmitt symbol size</code>	0.3	Size of the small Schmitt symbol to use near input leads.

For example, using a `not radius` of 0.1 will give a “not ball” of the same size of a connecting pole, as it is in the legacy ports.



```

1 \begin{circuitikz}
2   \draw (0,2) node[xnor port](P){}
3     (P.out) to[short, -o] ++(1,0);
4   \ctikzset{ieeestd ports/.cd, not radius=0.1,
5     xor bar distance=0.3, xor leads in=0}
6   \draw (0,0) node[xnor port](P){}
7     (P.out) to[short, -o] ++(1,0);
8 \end{circuitikz}

```

In addition to the specific parameters, you can also apply to these ports the boolean style `no input leads` as in legacy ones (this simply *does not draw* the input leads, but the anchors stays where they should):

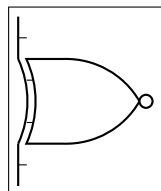


```

1 \begin{circuitikz}
2 \draw (0,0) node[nand port,
3   number inputs=5, no input leads,](B){Pn};
4 \draw (B.in 1) -- (B.bin 1) (B.in 5) -- (B.bin 5);
5 \node[rotate=90] at (B.in 3) {\dots};
6 \end{circuitikz}

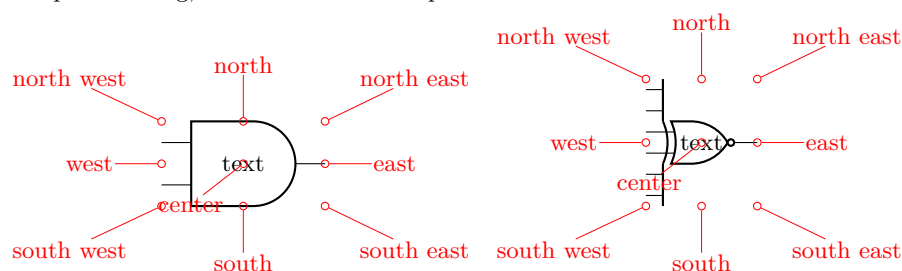
```

Changing the leads length must be done with a bit of care, because if the length is shorter than the port left or right extrusions strange things can happen (yes, a 4-inputs xnor gates is not so well defined...but it's a nice example to show):

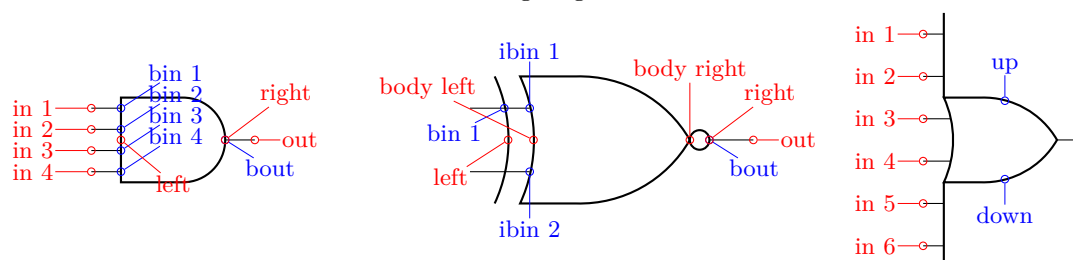


```
1 \begin{circuitikz}
2   \ctikzset{ieeestd ports/pin length=0.2}
3   \draw (0,0) node[xnor port,
4     number inputs=4, inner inputs=2] (B){};
5 \end{circuitikz}
```

4.22.6.3 IEEE standard ports anchors Geographical anchors define the rectangular space that the port is using, included the leads if presents.



Most of the anchors can be seen in the following diagram:



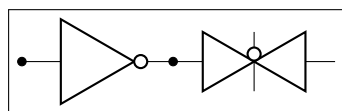
The inputs anchor are **in number** (on the tip of the lead) and **bin number** (border inputs) on the component's border (useful if you draw the ports with **no inut leads**). Additionally, you have **ibin number** (inner border inputs) for the *x*-type ports. The anchor named **left** is where a central border input would be.

In one-input ports (**not port**, the buffer, and Schmitt-type ports) you can use plain **in** or **in 1** indifferently. On the output, **out** is on the tip of the lead, and **bout** on the rightmost border (so, if there is a negation circle, it is on it); **right** is the same as **bout**.

The main body of the port is marked with **body left** and **body right** anchors (as seen in the middle port in the diagram above); you also have an **up** and **down** anchors centered on the body (you can use them as enable signals or similar things).

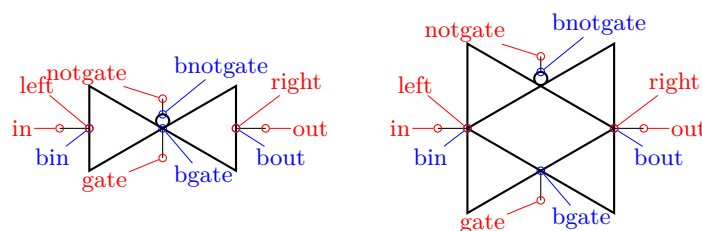
Finally, the internal **notcirc** node used for the output negation is accessible with the name **nodename-not**, where **nodename** is the name given to the logic port node.

4.22.6.4 Transmission gate symbols. The **tgate** and double **tgate** components are available since 1.2.4 but only in the IEEE style. An additional parameter **tgate scale** (default 0.7; if you set this to 1 the triangles will have the same size as a **ieeestd buffer port**) select the relative scale of the components.



```
1 \begin{circuitikz}
2   \ctikzset{logic ports=ieee}
3   \draw (0,0) to[inline not, *-*] ++(2,0)
4     node[tgate, anchor=in]{};
5 \end{circuitikz}
```

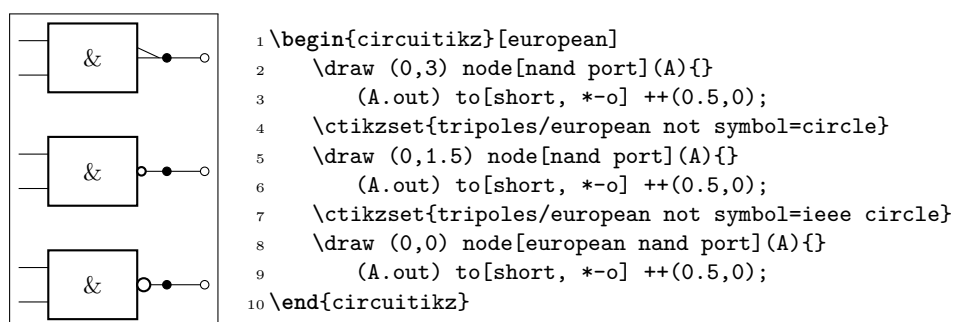
The anchors for the tgate's control point are called **gate** and **notgate** (and the corresponding **bgate** and **bnotgate** for the border anchors).



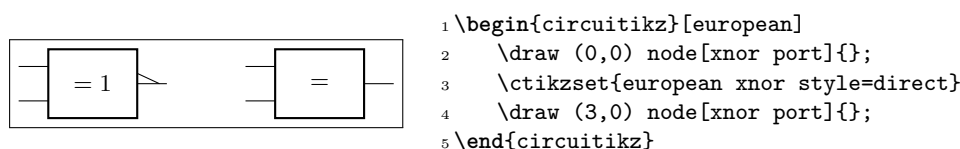
4.22.7 European logic port usage

European logic ports are the same class as American and IEEE-style ones, and they obey the same class modifier. Moreover, you can use the `no inputs pin` as in the other logic ports to suppress input pins.

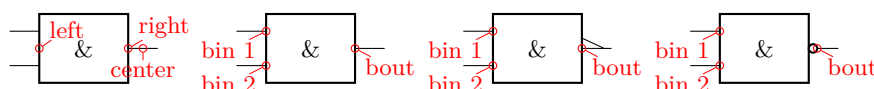
4.22.7.1 European logic port customization Normally the European-style logic port with inverted output are marked with a small triangle; if you want you can change it with the key `tripoles/european not symbol=circle`; its default is `triangle` but you can set it to `circle` like in the following example. As you can see, the circle size is the same as the circuit poles; if you prefer the size used in the IEEE standard ports, you can use set it to `ieee circle`.



In some standard, the **xnor** port is different — without the negation at the end and with just an `=` sign.⁷⁶ You can switch to this if you like, with the key `european xnor style=direct` that can be `default` or `direct`.



4.22.7.2 European logic port anchors The anchors are basically the same as in the American-style ports.

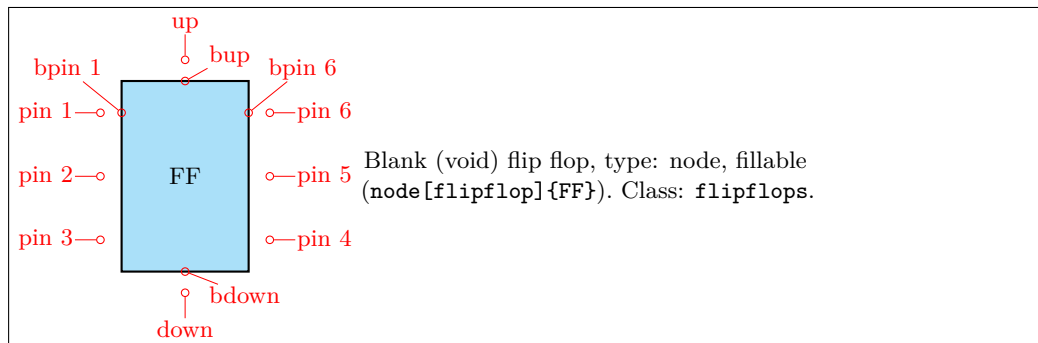


⁷⁶Suggested by user [Schlepptop](#) on GitHub.

4.23 Flip-flops

Flip-flops (available since version 1.0.0) are an hybrid between the logic ports and the chips. They have a class by themselves (`flipflops`) but the default parameters are set at the same values as the logic gates one.

The default flip flop is empty: it is just a rectangular box like a blank `dipchip` with 6 pins.



As you can see, in a void flip flop no external pins are drawn: you have to define the meaning of each of them to see them. To define a specific flip-flop, you have to set a series of keys under the `\ctikzset` directory `multipoles/flipflop/`, corresponding to pins 1...6, `u` for “up” and `d` for “down”:

- a *text* value `t0`, `t1`, ..., `t6`, and `tu` and `td` (the last ones for up and down) which will set a label on the pin;
- a *clock wedge* flag (`c0`, ..., `c6`, `cu`, `cd`), with value 0 or 1, which will draw a triangle shape on the border of the corresponding pin;
- a *negation* flag (`n0`, ..., `n6`, `nu`, `nd`), with value 0 or 1, which will put an `ocirc` shape on the outer border of the corresponding pin.

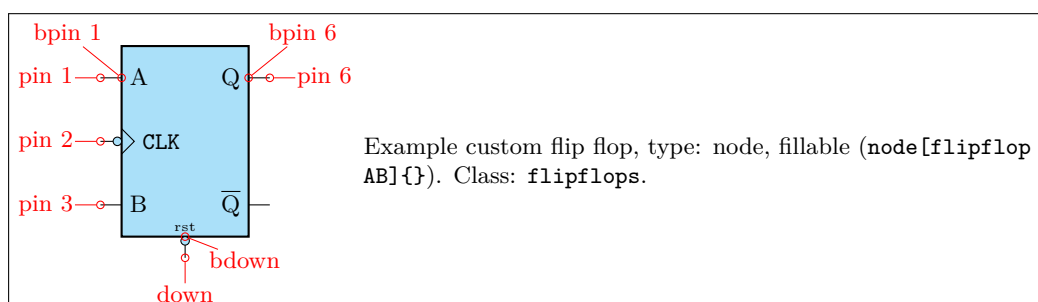
To set all these keys, an auxiliary style `flipflop` def is defined, so that you can do the following thing:

```

1 \tikzset{flipflop AB/.style={flipflop,
2   flipflop def={t1=A, t3=B, t6=Q, t4={\ctikztextnot{Q}},
3   td=rst, nd=1, c2=1, n2=1, t2={\texttt{CLK}}}},
4 }}

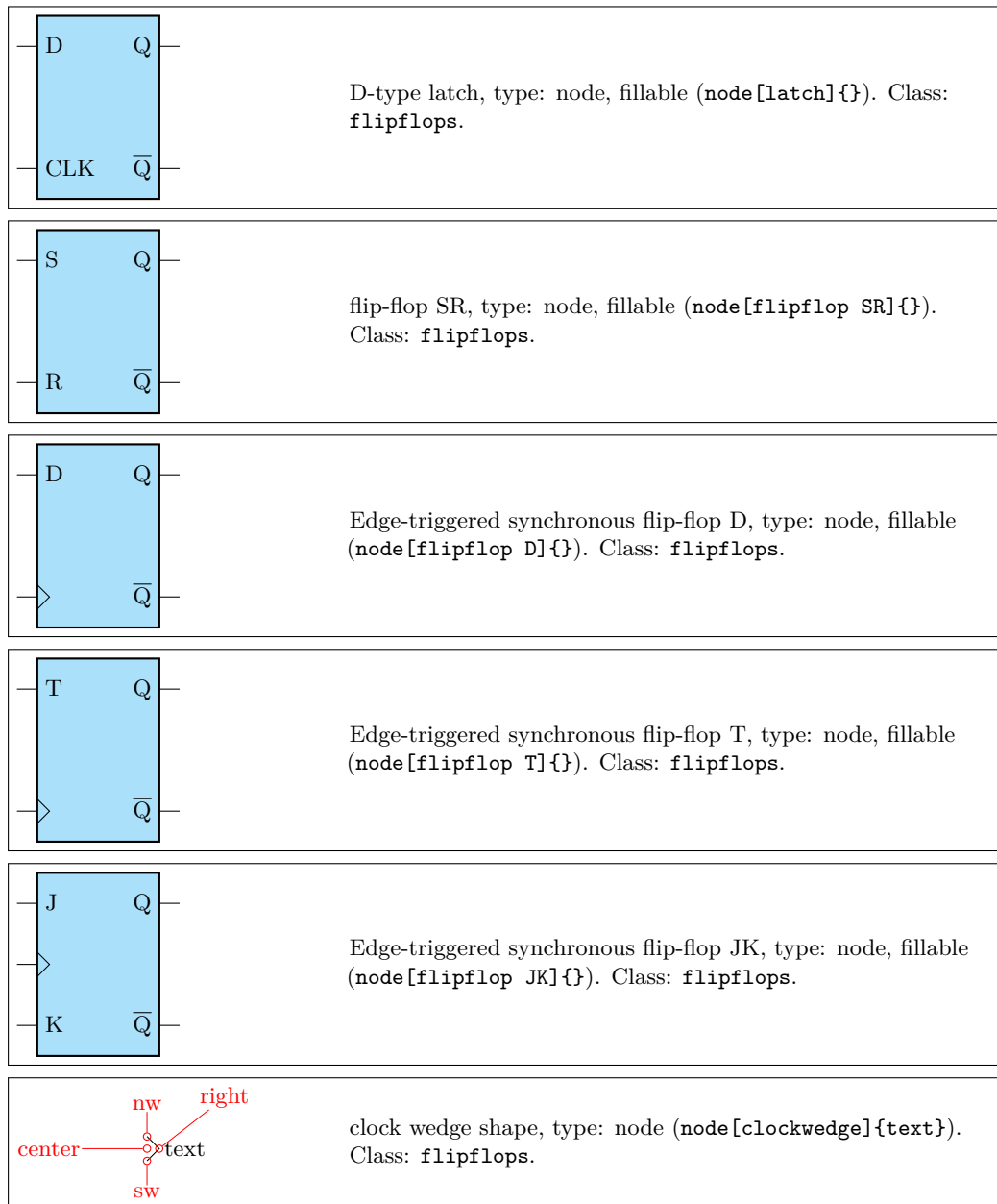
```

to obtain:

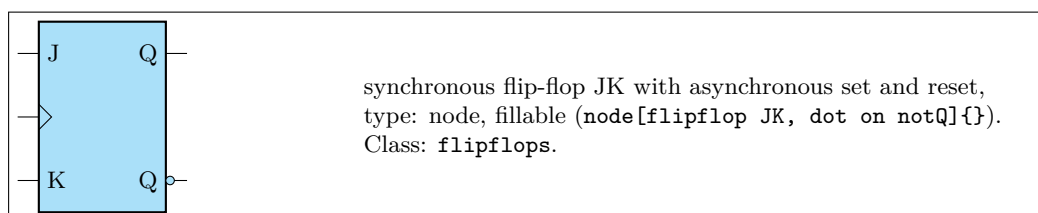


`\ctikztextnot{}` is a small utility macro to set a overbar to a text, like $\overline{\text{RST}}$ (created by `\ctikztextnot{RST}`).

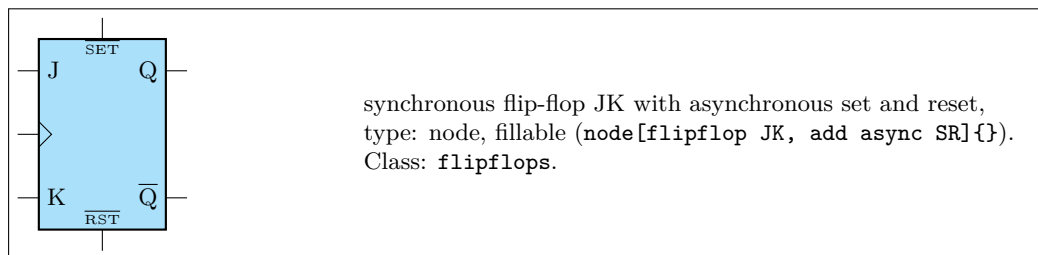
By default, the following flip-flops are defined, as well as a support shape for the clock wedge:



If you prefer that the negated output is labelled \bar{Q} and a dot indicating negation is shown, you can add the `dot on notQ` key:



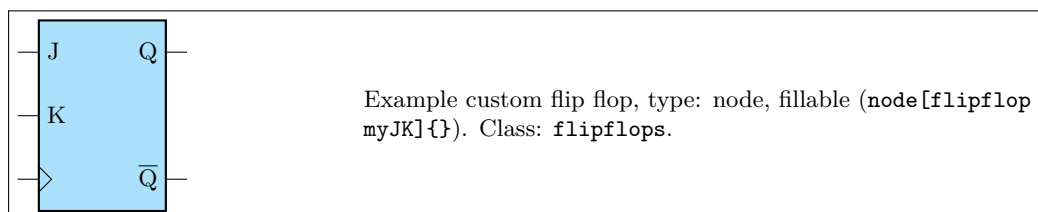
You can also add “vertical” asynchronous set and reset (active low) adding the style `add async SR` to all of them:



4.23.1 Custom flip-flops

If you like different pin distributions, you can easily define different flip-flops to your taste. For example, somebody likes the clock pin on the bottom pin:

```
1 \tikzset{flipflop myJK/.style={flipflop,
2   flipflop def={t1=J, t2=K, t6=Q, t4={\ctikztextnot{Q}}, c3=1}}
3 }
```

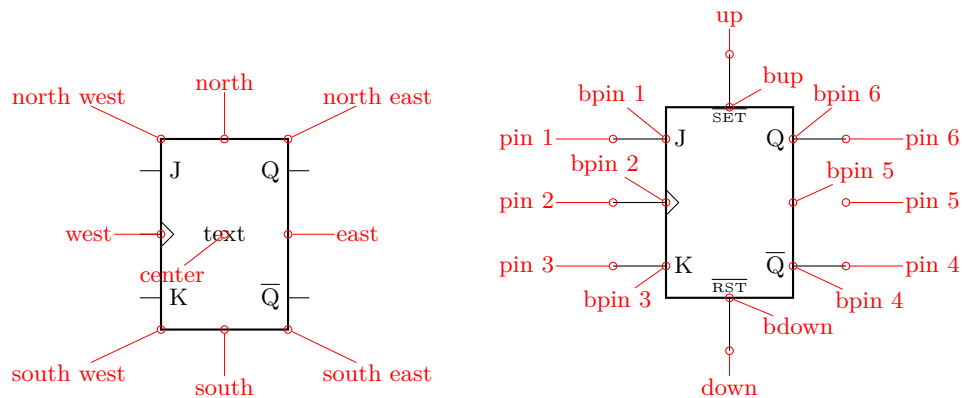


The standard definition of the default flip-flops are the following (in the file `pgfcircmultipoles.tex`):

```
1 \tikzset{
2   % async
3   latch/.style={flipflop, flipflop def={t1=D, t6=Q, t3=CLK, t4={\ctikztextnot{Q}}}},
4   flipflop SR/.style={flipflop, flipflop def={t1=S, t3=R, t6=Q, t4={\ctikztextnot{Q}}}},
5   % sync
6   flipflop D/.style={flipflop, flipflop def={t1=D, t6=Q, c3=1, t4={\ctikztextnot{Q}}}},
7   flipflop T/.style={flipflop, flipflop def={t1=T, t6=Q, c3=1, t4={\ctikztextnot{Q}}}},
8   flipflop JK/.style={flipflop,
9     flipflop def={t1=J, t3=K, c2=1, t6=Q, t4={\ctikztextnot{Q}}}},
10  % additional features
11  add async SR/.style={flipflop def={%
12    tu={\ctikztextnot{SET}}, td={\ctikztextnot{RST}}}},
13  dot on notQ/.style={flipflop def={t4={Q}, n4=1}},
14 }
```

4.23.2 Flip-flops anchors

Flip-flops have all the standard geometrical anchors, although it should be noticed that the external pins are *outside* them. The pins are accessed by the number 1 to 6 for the lateral ones (like in DIP chips), and with the `up` and `down` anchors for the top and bottom one. All the pins have the “border” variant (add a `b` in front of them, no spaces).



If you have negated pins, you can access the `ocirc` shapes with the name as `<nodename>-N<pin number>`, and all the respective anchors (for example — `myFFnode-N4.west`).

4.23.3 Flip-flops customization

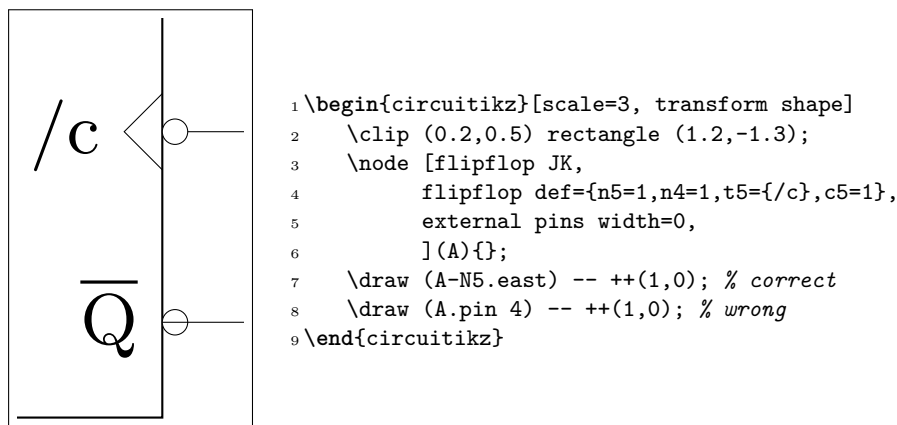
Flip-flop's size is controlled by the class parameters (like `flipflops/scale`) and the specific `\ctikzset` keys `multipoles/flipflop/width` and `multipoles/flipflop/pin spacing`. Class parameters are also used for line thickness and fill color. The default values are matched with the logic ports ones.

The fonts used for the pins 1...6 is set by the key `multipoles/flipflop/font` (by default `\small` in \LaTeX and the equivalent in other formats) and the font used for pins `u` and `d` is `multipoles/flipflop/fontud` (`\tiny` by default). You can change it globally or specifically for each flip flop.

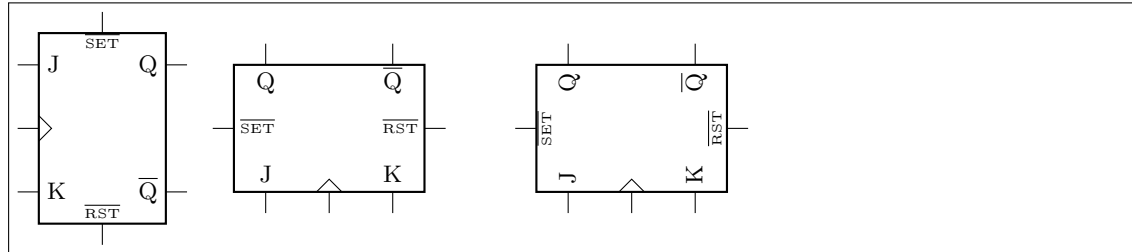
As in chips, you can change the length of the external pin with the key `external pins width`; you can for example have a pinless flip-flop like this:



Notice however that negated pins when the pins width is zero has to be handled with care. As explained in the poles sections, the `ocirc` shape is drawn at the end of the shape to cancel out the wires below; so if you use a pinless flipflop when you make the connection you should take care of connecting the symbol correctly. To this end, the shapes of the negation circles are made available as `<nodename>-N<pin number>`, as you can see in the next (contrived) example.



Normally the symbols on the flip-flop are un-rotated when you rotate the symbol, but as in case of chips, you can avoid it.

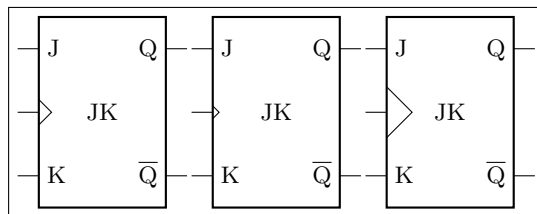


```

1 \begin{tikzpicture}
2   \draw (0,0) node[flipflop JK, add async SR]{};
3   \draw (3,0) node[flipflop JK, add async SR, rotate=90]{};
4   \draw (7,0) node[flipflop JK, add async SR, rotate=90, rotated numbers]{};
5 \end{tikzpicture}

```

You can also change the size of the wedge, with the key `multipoles/flipflop/clock wedge size` (default value 0.2).

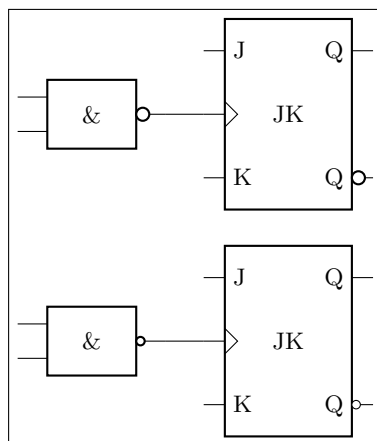


```

1 \begin{circuitikz}[]
2   \draw (0,0) node[flipflop JK]{JK};
3   \ctikzset{multipoles/flipflop/clock
4     wedge size=0.1}
5   \draw (2.3,0) node[flipflop JK]{JK};
6   \ctikzset{multipoles/flipflop/clock
7     wedge size=0.4}
8   \draw (4.6,0) node[flipflop JK]{JK};
9 \end{circuitikz}

```

Flip-flops “not circles” follows the current logic port setting (either if you choose `ieee ports`, or if you are using `european ports` with `european not symbol` set to `circle` or `ieee circle`).



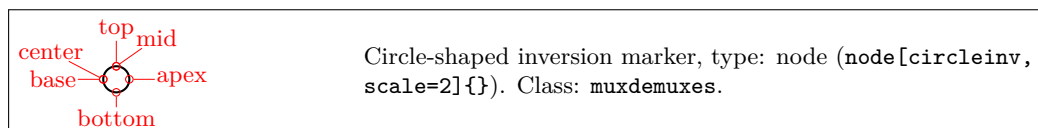
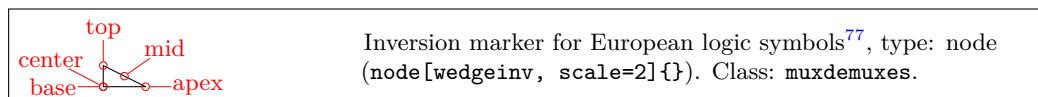
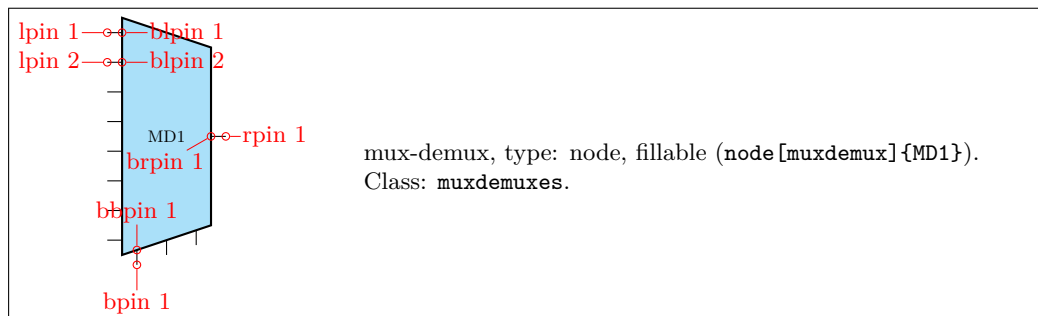
```

1 \begin{circuitikz}[]
2 \ctikzset{logic ports=european,
3   tripoles/european not symbol=ieee circle}
4 \draw (0,0) node[nand port](A){}
5   (A.out) to[short] ++(0.5,0)
6   node[flipflop JK, dot on notQ, anchor=pin 2]{JK};
7 \ctikzset{logic ports=european,
8   tripoles/european not symbol=circle}
9 \draw (0,-3) node[nand port](A){}
10   (A.out) to[short] ++(0.5,0)
11   node[flipflop JK, dot on notQ, anchor=pin 2]{JK};
12 \end{circuitikz}

```

4.24 Multiplexer and de-multiplexer

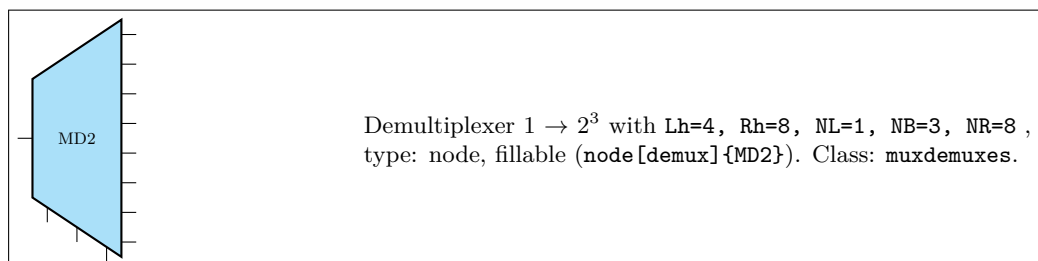
The shape used for muxes and de-muxes is probably the most configurable shape of the package; it has been added by Romano in v1.0.0. The basic shape is a multiplexer with 8 input pin, one output pin, and three control pins ($2^3 \rightarrow 1$ multiplexer). The pins are not named as input or output pins (see below for a full description for anchors) for reasons that will be clear later.



You can define a custom shape for the muxdemuxes using an interface similar to the one used in flip-flops; for example:

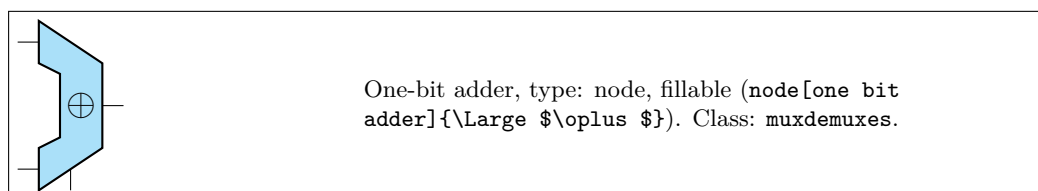
```
\tikzset{demux/.style={muxdemux, muxdemux def={Lh=4, Rh=8, NL=1, NB=3, NR=8}}}
```

will generate the following shape (the definition above is already defined in the package):



The shape can also be defined with an inset. For example it can be used like this to define a 1-bit adder (also already available):

```
\tikzset{one bit adder/.style={muxdemux,
muxdemux def={Lh=4, NL=2, Rh=2, NR=1, NB=1, w=1.5,
inset w=0.5, inset Lh=2, inset Rh=1.5}}}
```



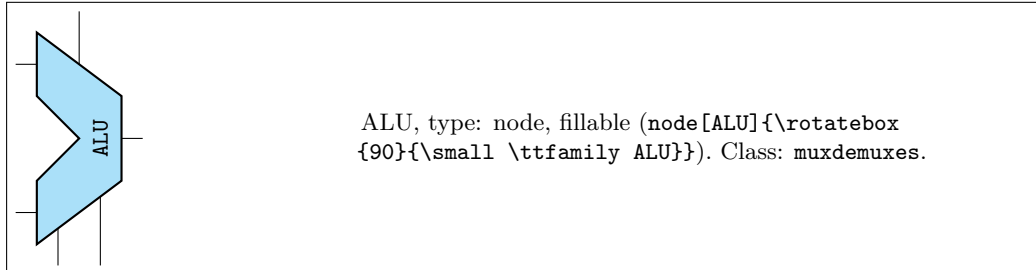
Or a Arithmetic Logic Unit (again, already defined by default):

⁷⁷Thanks for the contribution by [yashpalgoyal1304 on GitHub](#).

```

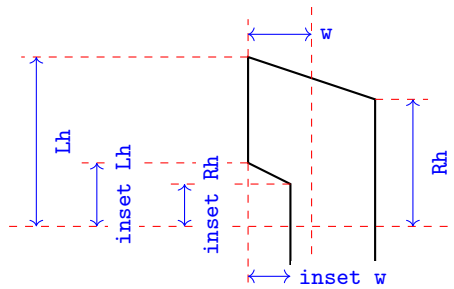
1 \tikzset{ALU/.style={muxdemux,
2     muxdemux def={Lh=5, NL=2, Rh=2, NR=1, NB=2, NT=1, w=2,
3     inset w=1, inset Lh=2, inset Rh=0, square pins=1}}}

```



4.24.1 Mux-Demux: design your own shape

In designing the shape there are several parameters to be taken into account. In the diagram on the right they are shown in a (hopefully) practical way. The parameter can be set in a node or in a style using the `muxdemux def` key as shown above, or set with `\ctikzset` as `multipoles/muxdemux/Lh` keys and so on.

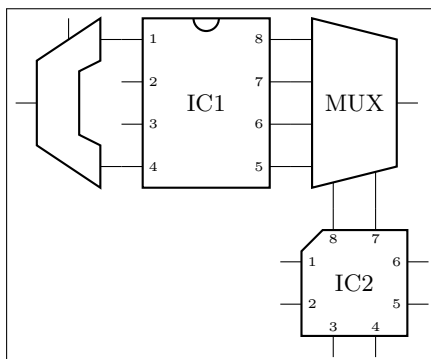


The default values are `Lh = 8`, `Rh = 6`, `w = 3` and no inset: `inset Lh = inset Rh = inset w = 0`. In addition, you can set the following parameters:

NL, NR, NB, NT : number of pins relatively on the left, right, bottom and top side (default 8, 1, 3, 0). When an inset is active (in other words, when `Lh > 0`) the pins are positioned on the top and bottom part, not in the inset; the exception is when the number of left pins is odd, in which case you have one pin set on the center of the inset. If you do not want a pin in one side, use 0 as number of pins.

square pins : set to 0 (default) if you want the square pins to stick out following the slope of the bottom or top side, 1 if you want them to stick out in a square way (see the example above for the ALU).

All the distances are multiple of `multipoles/muxdemux/base len` (default 0.4, to be set with `\ctikzset`), which is relative to the basic length. That value has been chosen so that, if you have a number of pins which is equal to the effective distance where they are spread (which is `Lh` without inset, `Lh - (inset Lh)` with an inset), then the distance is the same as the default pin distance in chips, as shown in the next circuit. In the same drawing you can see the effect of `square pins` parameters (without it, the rightmost bottom lead of the `mux 4by2` shape will not connect with the below one).



```

1 \begin{circuitikz}
2   \tikzset{mux 4by2/.style={muxdemux,
3     muxdemux def={Lh=4, NL=4, Rh=3,
4     NB=2, w=2, square pins=1}}}
5   \node [dipchip, num pins=8] (A) at (0,0) {IC1};
6   \node [one bit adder, scale=-1, anchor=lpin 2]
7     at (A.pin 1){};
8   \node [mux 4by2, anchor=lpin 1] (B)
9     at (A.pin 8){MUX};
10  \node [qfpchip, num pins=8, anchor=pin 8] at
11    (B.bpin 1) {IC2};
12 \end{circuitikz}

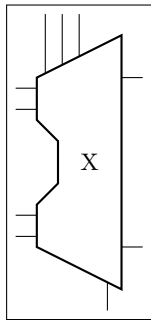
```

4.24.2 Mux-Demux customization

Mux-demuxes have the normal parameters of their class (`muxdemuxes`): you can scale them with the `\ctikzset` key `muxdemuxes/scale`, control the border thickness with `muxdemuxes/thickness` and the default fill color with `muxdemuxes/fill` — they are set, by default, at the same values than `logic ports`.

External pins' length is controlled by the key `multipoles/external pins width` (default 0.2) or by the style `external pins width`. The parameter `multipoles/external pins thickness` is also respected, like in chips. In addition, like in logic ports, you can suppress the drawing of the leads by using the boolean key `logic ports draw input leads` (default `true`) or, locally, with the style `no inputs leads` (that can be reverted with `input leads`). The main difference between setting `external pins width` to 0 or using `no inputs lead` is that in the first case the normal pin anchors and the border anchors will coincide, and in the second case they will not move and stay where they should have been if the leads were drawn.

You can draw only selected pins and leave out the rest by setting the keys `multipoles/draw only side pins` and the corresponding style `draw only side pins` where `side` can be `left`, `right`, `top`, `bottom`. Those key accept a comma-separated list of pin numbers or ranges of pin numbers (a range is given as `<start> - <end>`, ends are inclusive). The numbers will not be expanded in any way, except those given as ends of ranges. A special value (and the initial one) is `all`, in which case all pins are drawn. The anchors will be adjusted, such that each `xpin n` will be placed at the end of the pins which are drawn, and coincide with the `bpin n` anchors for the suppressed pins.



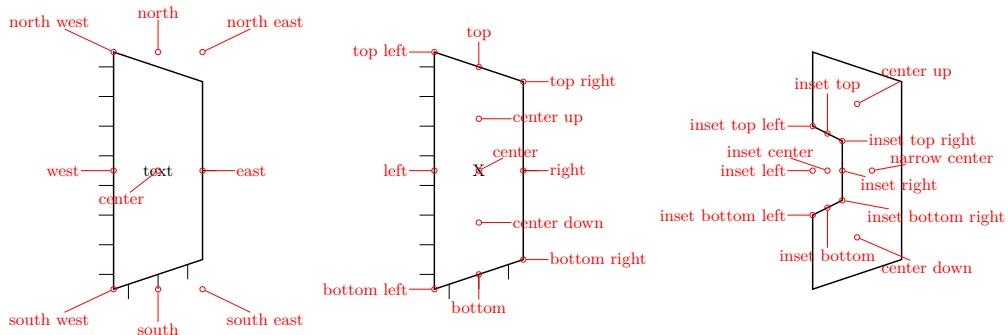
```

1 \begin{circuitikz}
2 \node [muxdemux, muxdemux def={NL=4, NR=3, NT=5, NB=3, w=2,
3     inset w=0.5, Lh=4, inset Lh=2.0, inset Rh=1.0,
4     square pins=1},
5     draw only right pins={1,3},
6     draw only top pins={1-3},
7     draw only bottom pins={3}](C) at (0,0) {X};
8 \end{circuitikz}

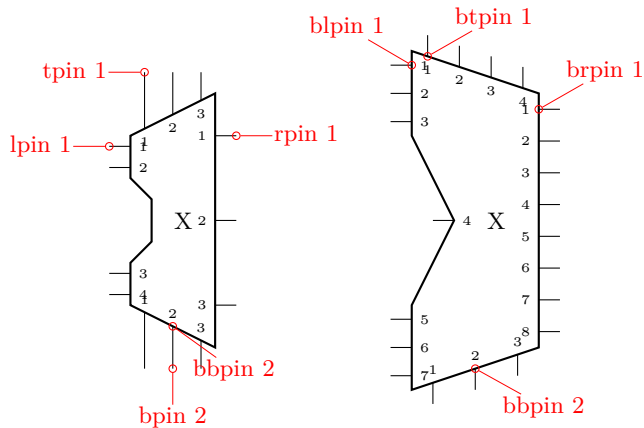
```

4.24.3 Mux-Demux anchors

Mux-demuxes have a plethora of anchors. As in the case of chips, the geographic anchors mark the rectangle occupied by the component, without taking into account the pin leads.



The pins anchors are named `lpin`, `rpin`, `bpin` and `tpin` for the left, right, bottom and top pin respectively, and points to the “external” pin. The border pins are named the same, with a `b` added in front: `blpin`, `brpin`, `bbpin` and `btpin`. The following graph will show the numbering and position of the pin anchors.



The code that implemented the printing of the numbers (which in `muxdemuxes`, differently from chips, are never printed automatically) in the last graph is the following one.

```

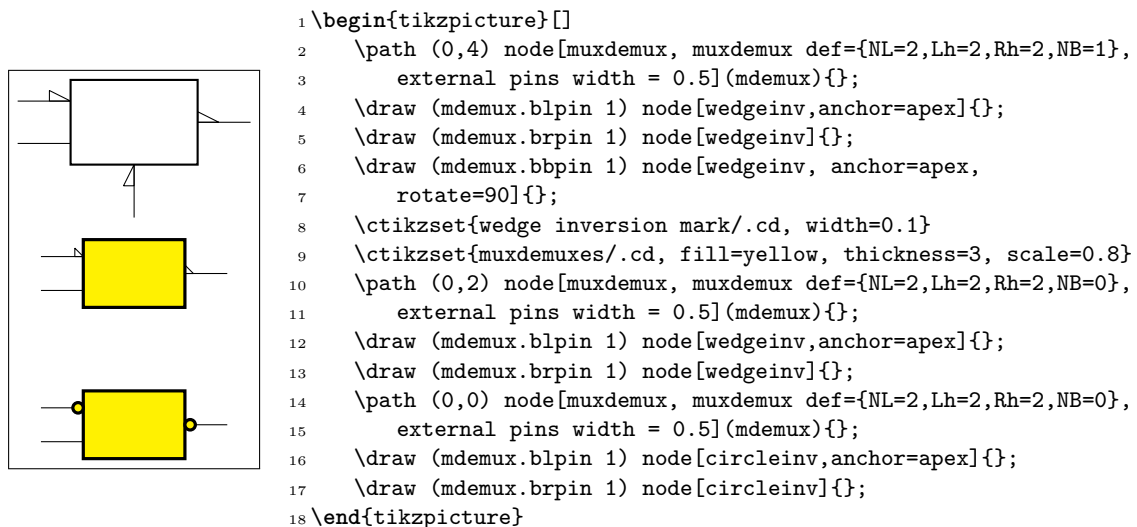
1 \begin{circuitikz}
2 \node [muxdemux, muxdemux def={NL=4, NR=3, NT=3, NB=3, w=2, inset w=0.5,
3   Lh=4, inset Lh=2.0, inset Rh=1.0, square pins=1}] (C) at (0,0) {X};
4 \node [muxdemux, muxdemux def={NL=7, NR=8, NT=4, inset w=1.0,
5   inset Lh=4.0, inset Rh=0.0}] (D) at (4,0) {X};
6 \foreach \myn/\NL/\NR/\NB/\NT in {C/4/3/3/3,D/7/8/3/4} {
7   \foreach \myp in {1,...,\NL} \node[right, font=\tiny] at (\myn.blpin \myp){\myp};
8   \foreach \myp in {1,...,\NR} \node[left, font=\tiny] at (\myn.brpin \myp) {\myp};
9   \foreach \myp in {1,...,\NB} \node[above, font=\tiny] at (\myn.bbpin \myp){\myp};
10  \foreach \myp in {1,...,\NT} \node[below, font=\tiny] at (\myn.btpin \myp){\myp};
11 }

```

4.24.4 Adding wedge or circular inversion markers

Although you can add “negation balls” as seen for, for example, flip-flops (see section 4.23.3), sometimes the European-style notation (also accepted by the IEEE standard) with the small wedge is preferred. The `wedgeinv` shape will nicely do. It’ll scale with the `muxdemuxes` class, and the length and height can be changed with the keys `wedge inversion mark/width` (default 0.2) and `height` (default 0.1), with the same units that are used for the `external pins width` and similar keys.

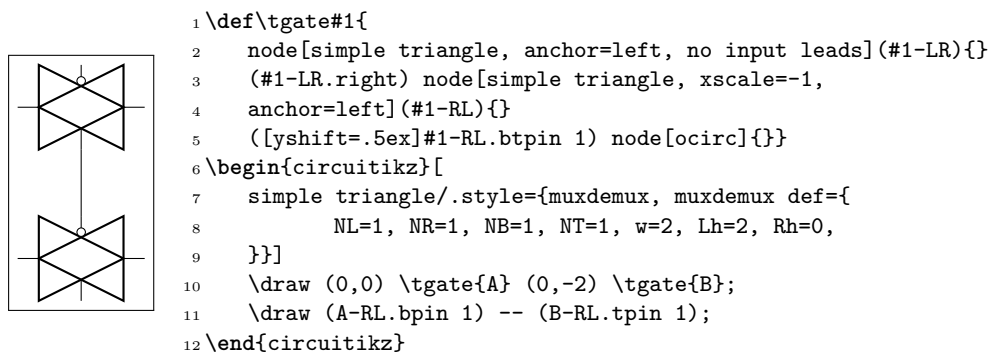
Similarly, there is also a `circleinv` shape, which is basically the same as the `notcirc` (see 4.22.2) one, but that scales with the `muxdemuxes` class and that has the default anchor at its left, similarly to `wedgeinv`. This one will be filled if the class says so, contrary to the wedge-like shapes that are always open.



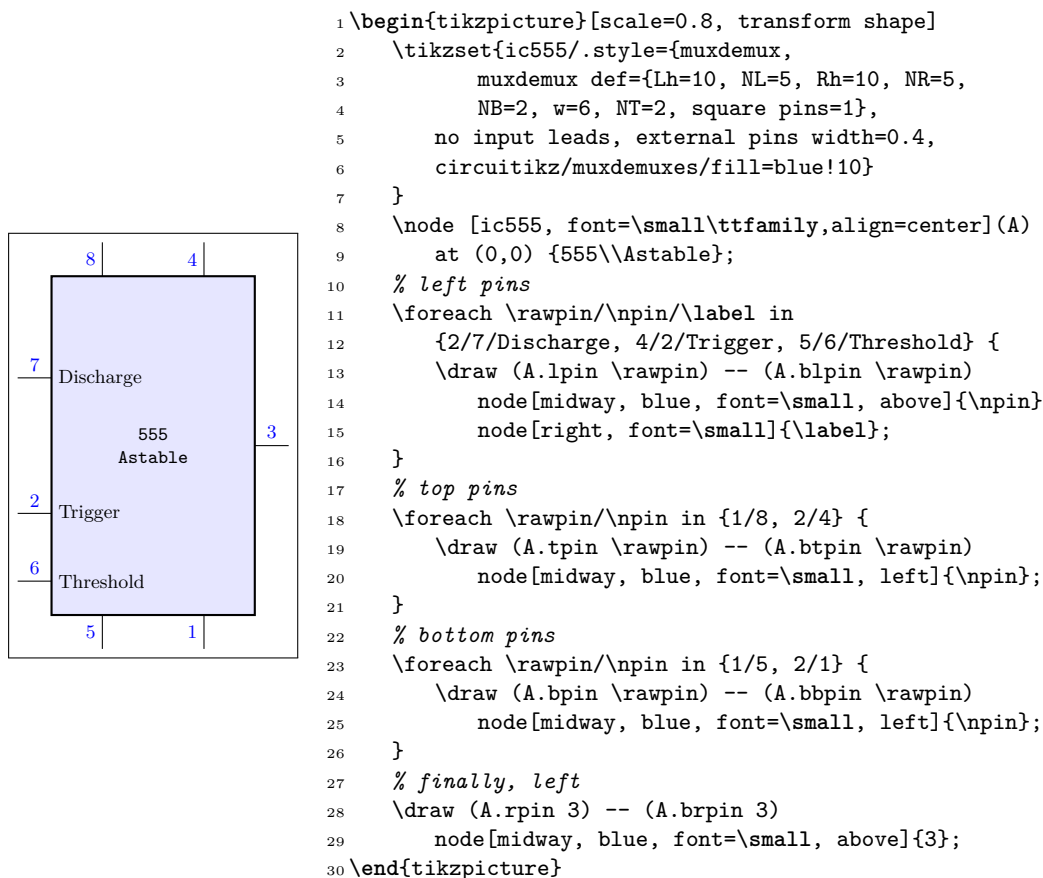
4.24.5 Mux-Demux special usage

You can use these shapes to draw a lot of symbols that are unavailable; using a bit of L^AT_EX command trickery you can use them quite naturally too.... Examples with personalized amplifier shapes are listed in section 4.20.3.

As an additional example, this was used before the introduction of the `double tgate` symbol in 1.2.4 (see 4.22.6.4):

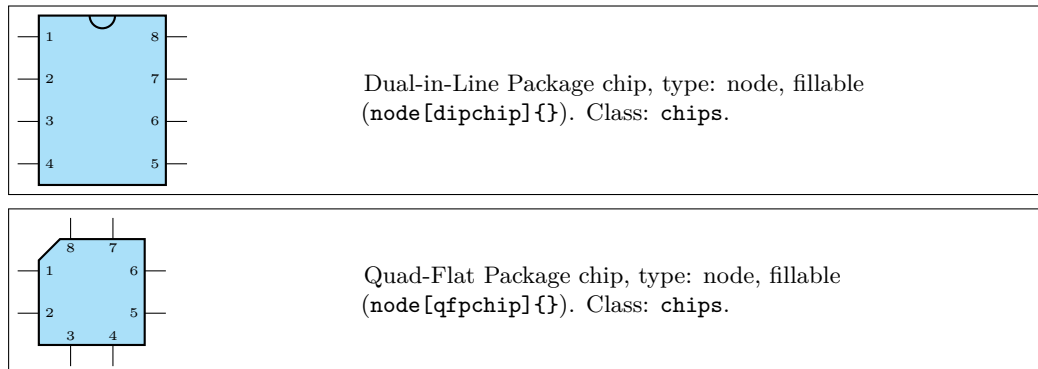


Finally, you can play with them to create chips that have generic numbers of pins on the four sides, as in the following example (asked on [TeX.Stackexchange](https://tex.stackexchange.com)):



4.25 Chips (integrated circuits)

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



4.25.1 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.

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).

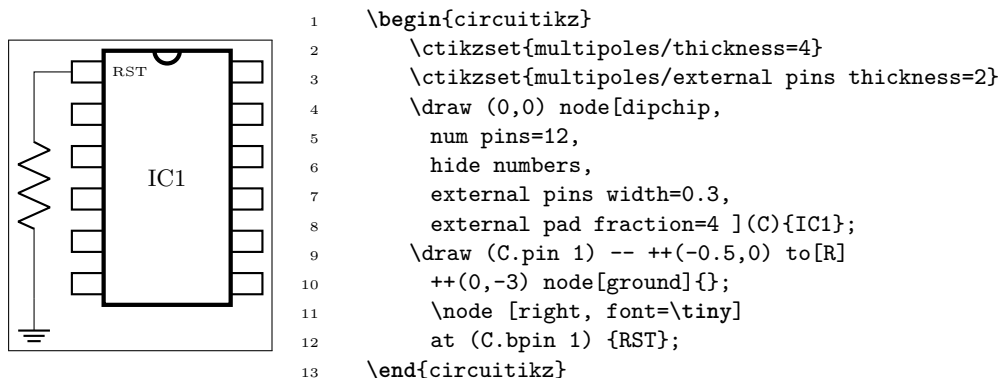
You can customize the DIP chip with the key `multipoles/dipchip/width` (with a default of 1.2) and the key `multipoles/dipchip/pin spacing` (default 0.4) that are expressed in fraction of basic lengths (see section 3.1.4). The height of the chip will be equal to half the numbers of pins multiplied by 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 number of pins is settable with the key `num pins`. **Please notice** that the number of pins **must** be *even* for `dipchips` and *multiple of 4* for `qfpchips`, otherwise havoc will ensue.

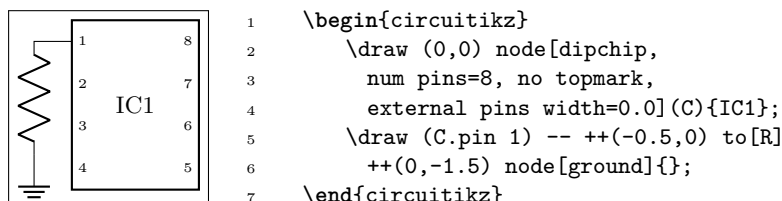
The pins of the chip can be “hidden” (that is, just a spot in the border, optionally marked with a number) or “stick out” with a thin lead by setting `multipoles/external pins width` greater than 0 (default value is 0.2, so you’ll have leads as shown above). Moreover, you can transform the thin lead into a pad by setting the key `multipoles/external pad fraction` to something different from 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.

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).

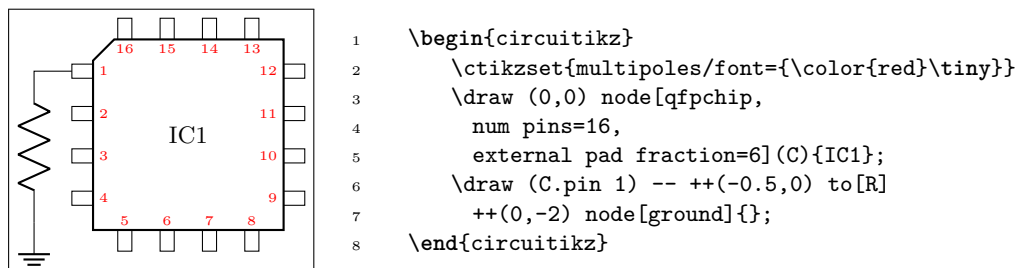


Also, you can suppress the drawing of the pins, by using the style `no inputs leads` (that can be reverted with `input leads`). The main difference between setting `external pins width` to 0 or using `no inputs lead` is that in the first case the normal pin anchors and the border anchors will coincide, and in the second case they will not move and stay where they should have been if the leads were drawn.

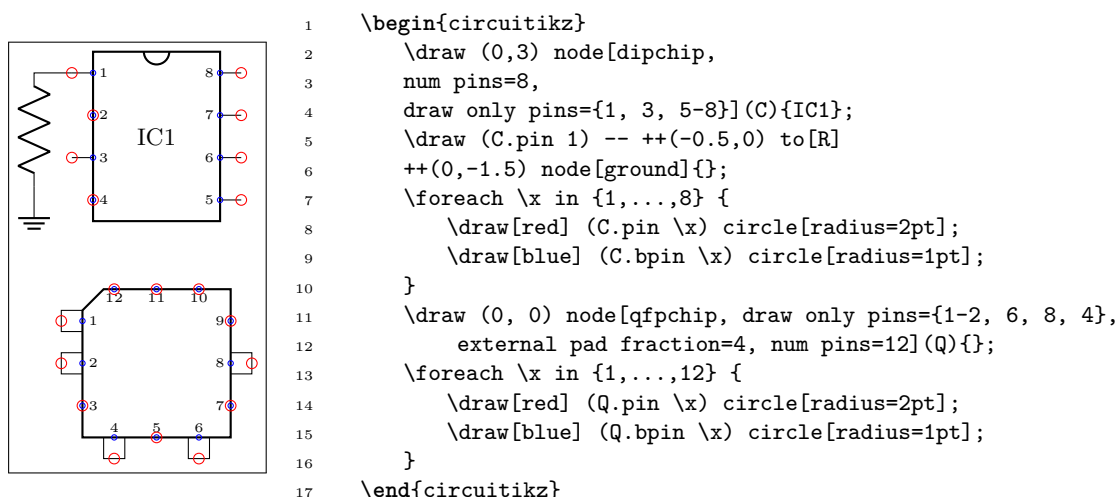
For special use you can suppress the orientation mark with the key `no topmark` (default `topmark`).



The font used for the pins is adjustable with the key `multipoles/font` (default `\tiny`)



You can draw only selected pins and leave out the rest by setting `multipoles/draw only pins`⁷⁸. This key accepts a comma-separated list of pin numbers or ranges of pin numbers (a range is given as `(start - (end))`, ends are inclusive). The numbers will not be expanded in any way, except those given as ends of ranges. A special value (and the initial one) is `all`, in which case all pins are drawn. The anchors will be adjusted, such that each `pin n` will be placed at the end of the pins which are drawn, and coincide with the `bpin n` anchors for the suppressed pins.

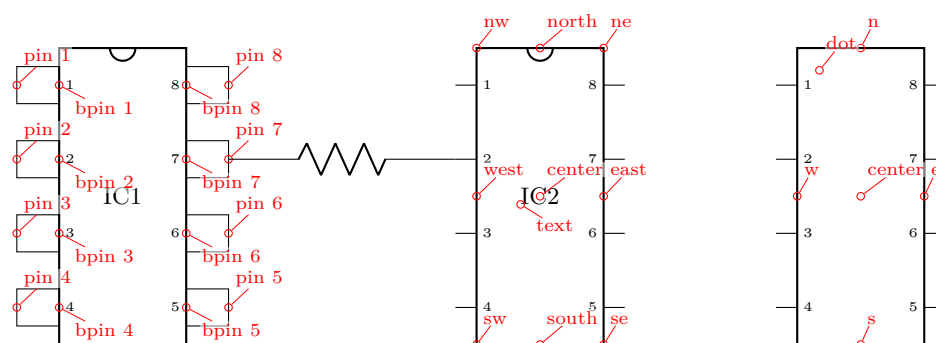


⁷⁸Added by Jonathan P. Spratte in v1.3.8

4.25.2 Chips anchors

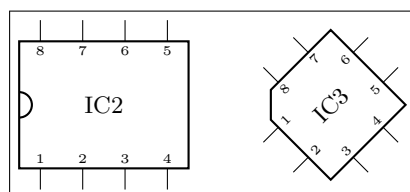
Chips have anchors on pins and global anchors for the main shape. The pin anchors to be used to connect wires to the chip are called **pin 1**, **pin 2**, ..., with just one space between **pin** and the number. Border pin anchors (**bpin 1**...) are always on the box border, and can be used to add numbers 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.



4.25.3 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.



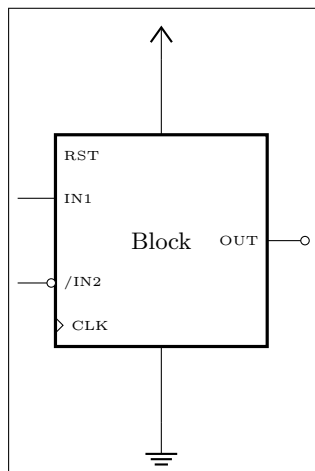
```

1 \begin{circuitikz}
2   \draw (0,0) node[dipchip,
3     rotate=90]{%
4       \rotatebox{-90}{IC2}};
5   \draw (3,0) node[qfpchip,
6     rotated numbers,
7     rotate=45]{IC3};
8 \end{circuitikz}

```

4.25.4 Chip special usage

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



```

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

```

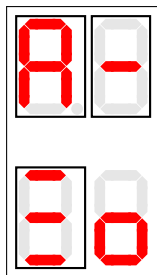
4.26 Seven segment displays



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.⁷⁹

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



```

1 \begin{circuitikz}
2   \draw (0,0) node[seven segment val=A dot off box on]{};
3   \draw (1,0) node[seven segment val=- dot none box on]{};
4   \draw (0,-2) node[seven segment bits=1001001 dot empty box on]{};
5   \draw (1,-2) node[seven segment bits=0011101 dot none box off]{};
6 \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.

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.

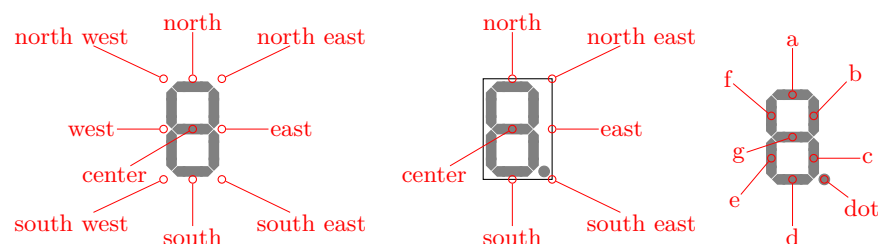
⁷⁹This component has been loosely inspired by the package `SevenSeg` by Germain Gondor, 2009, see TExample.net.

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.

4.26.1 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.



4.26.2 Seven segments customization

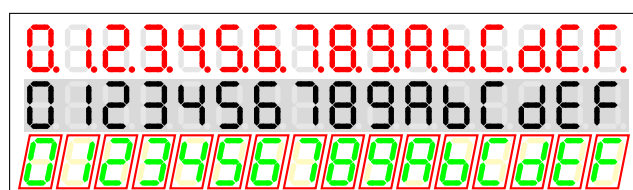
You can scale the seven segment display with the key `displays/scale`. This will scale the size of the digit, but not the absolute sizes shown below — if you want them to scale, you have to do it manually.

You can change several parameters to adjust the displays:

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

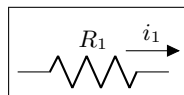
A couple of examples are shown below.

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

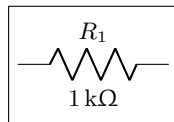


5 Labels, voltages and currents

You can add “decorations” to the path-style components; there are basically five types of them: labels, annotations, voltages, currents, and flows. Let’s see an example of all of them...



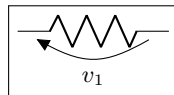
```
1 \begin{circuitikz}
2   \draw (0,0) to[R, l=$R_1$, f=$i_1$] (2,0);
3 \end{circuitikz}
```



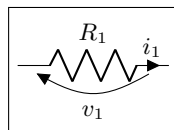
```
1 \begin{circuitikz}
2   \draw (0,0) to[R=$R_1$, a=\SI{1}{\kohm}] (2,0);
3 \end{circuitikz}
```



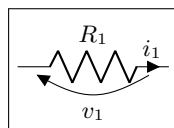
```
1 \begin{circuitikz}
2   \draw (0,0) to[R, i=$i_1$] (2,0);
3 \end{circuitikz}
```



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

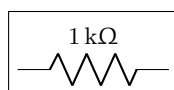


```
1 \begin{circuitikz}
2   \draw (0,0) to[R=$R_1$, i=$i_1$, v=$v_1$] (2,0);
3 \end{circuitikz}
```



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

Long names/styles for the bipoles can be used, of course, and there is a special syntax (that works only in simple cases, and only with L^AT_EX — use it with caution!) if you load the package with the ‘siunitx’ options:



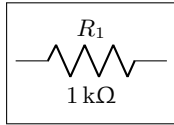
```
1 \begin{circuitikz}\draw
2   (0,0) to[resistor=1<\kilo\ohm>] (2,0);
3 \end{circuitikz}
```

5.1 Labels and Annotations

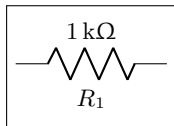
Since Version 0.7, beside the original label (l) option, there is a new option to place a second label, called annotation (a) at each bipole.

5.1.1.1 Label and annotation position

When drawing a component left-to-right, the label l is by default above the component, and the annotation a is by default below it. The position of annotations and labels can be adjusted adding the characters $_$ or $^$ to the key.

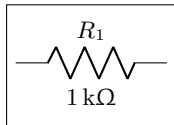


```
1 \begin{circuitikz}
2   \draw (0,0) to[R, l=$R_1$,a=1<\kilo\ohm>] (2,0);
3 \end{circuitikz}
```



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

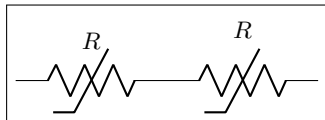
For passive components, you can use `component type=text` as a shortcut for `component type, l=text`:



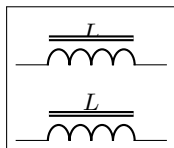
```
1 \begin{circuitikz}
2   \draw (0,0) to[R=$R_1$,a=1<\kilo\ohm>] (2,0);
3 \end{circuitikz}
```

Notice though that in active component (sources of either voltage or current) the shortcut will set the voltage (v) or current (i) property.

5.1.1.1.1 Adjust label and annotation position. Normally the package will guess a good position for the label or annotation; if you do not like it, you can add⁸⁰ (or remove, with negative values) distance using the `\ctikzset` keys `label distance` and `annotation distance`.



```
1 \begin{circuitikz}
2   \draw (0,0) to[sR, l=$R$, label distance=-4pt] (2,0)
3     to [sR, l=$R$] (4,0);
4 \end{circuitikz}
```

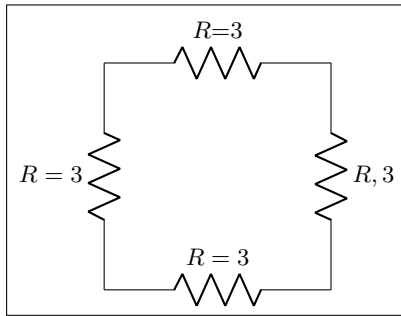


```
1 \begin{circuitikz}[american]
2   \ctikzset{bipoles/inductors/core distance=4pt}
3   \draw (0,1) to[L=$L$, name=myL] ++(2,0);
4   \draw[thick, double] (myL.core west) -- (myL.core east);
5   \draw (0,0) to[L=$L$, name=myL, label distance=2pt] ++(2,0);
6   \draw[thick, double] (myL.core west) -- (myL.core east);
7 \end{circuitikz}
```

5.1.2 Special symbols in labels and annotations.

When TikZ processes the options, there will be problems if the label (or annotation, voltage, or current) contains one of the characters $=$ (equal) or $,$ (comma) — because the parser search for those two characters to delimit the arguments, giving unexpected errors and wrong output. These two characters can be protected from the option parser using an extra set of braces.

⁸⁰Since version 1.3.3

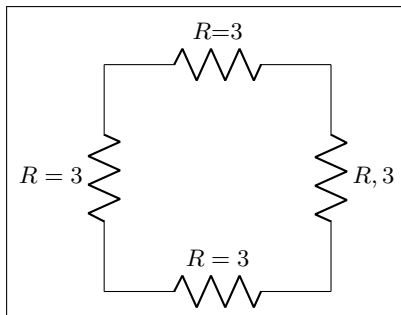


```

1 \begin{circuitikz}
2   % the following will fail:
3   % \draw (0,0) to[R, l=$R=3$] (3,0);
4   \draw (0,0) to[R, l={$R=3$}] (3,0);
5   \draw (0,0) to[R={$R=3$}] (0,3);
6   \draw (3,3) to[R={$R,3$}] (3,0);
7   % this works, but it has wrong spacing
8   \draw (0,3) to[R, l=$R{=3}$] (3,3);
9 \end{circuitikz}

```

Caveat: up to version 1.2.7, due to the way in which CircuitikZ used to process the options, even that was not sufficient, so you must protect that tokens even more, for example using an `\mbox` command, or redefining the characters with a `\TeX \def`:



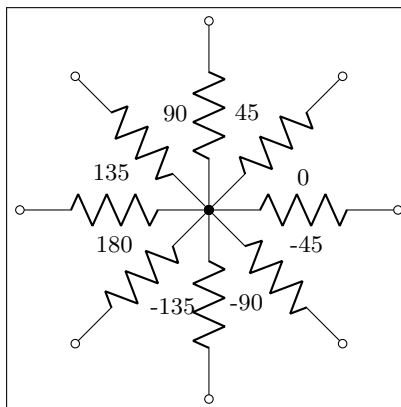
```

1 \begin{circuitikz}
2   \def\eq{=}
3   % the following will fail up to 1.2.7:
4   % \draw (0,0) to[R, l={$R=3$}] (3,0);
5   \draw (0,0) to[R, l=\mbox{$R=3$}] (3,0);
6   \draw (0,0) to[R, l=$R\eq3$] (0,3);
7   \draw (3,3) to[R, l=\mbox{$R,3$}] (3,0);
8   % this works, but it has wrong spacing
9   \draw (0,3) to[R, l=$R{=3}$] (3,3);
10 \end{circuitikz}

```

5.1.3 Labels and annotation orientation.

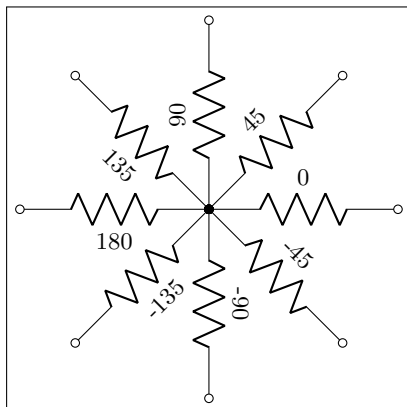
The default orientation of labels is controlled by the options `smartlabels`, `rotatelabels` and `straightlabels` (or the corresponding `label/align` keys). Here are examples to see the differences:



```

1 \begin{circuitikz}
2 \ctikzset{label/align = straight}
3 \def\DIR{0,45,90,135,180,-90,-45,-135}
4 \foreach \i in \DIR {
5   \draw (0,0) to[R=\i, *-o] (\i:2.5);
6 }
7 \end{circuitikz}

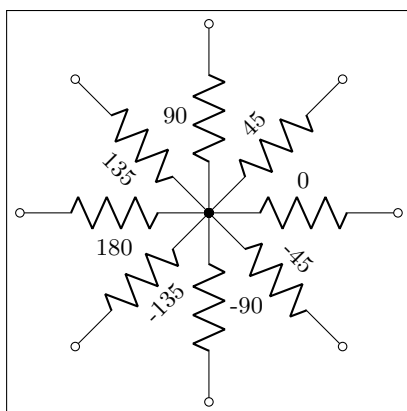
```



```

1 \begin{circuitikz}
2 \ctikzset{label/align = rotate}
3 \def\DIR{0,45,90,135,180,-90,-45,-135}
4 \foreach \i in \DIR {
5   \draw (0,0) to[R=\i, *-o] (\i:2.5);
6 }
7 \end{circuitikz}

```



```

1 \begin{circuitikz}
2 \ctikzset{label/align = smart}
3 \def\DIR{0,45,90,135,180,-90,-45,-135}
4 \foreach \i in \DIR {
5   \draw (0,0) to[R=\i, *-o] (\i:2.5);
6 }
7 \end{circuitikz}

```

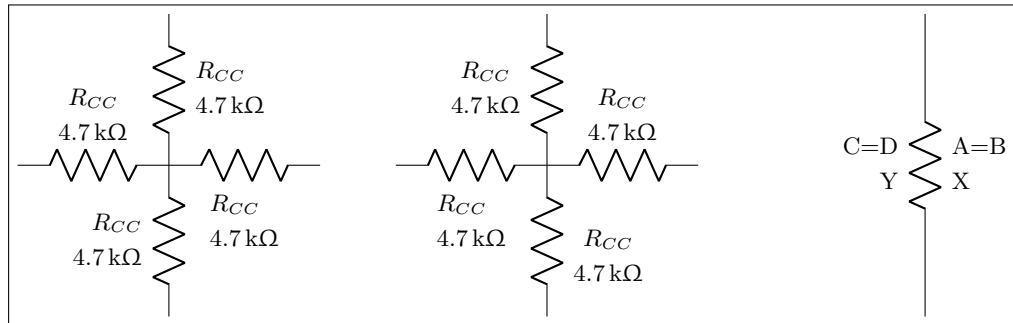
5.1.4 Stacked (two lines) labels.

When using `circuitikz` in LaTeX, you 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 (left, center, right) and `l2 valign` to control the vertical one (bottom, center, top). The default values for alignments are thought for vertical components (where the stacked labels are more natural), in other positions you have to force them.

Notice that you **can't use** the compact `<...>` notation for `siunitx` with stacked labels. Before `v1.3.6` the label was ignored, but that has been converted into an error.

Since `v1.3.6` you have the same possibility with the `annotation` (just use `a2=...`, `a2_=...`, `a2 valign` and so on. Notice that the default position for stacked annotation is `v2 halign=1`.

The `l2` and `a2` will only work in LaTeX because they use a `tabular` environment in their implementation. For plain TeX and ConTeXt you have to use `l` and `a` and build the stack of labels externally.



```

1 \begin{circuitikz}[american]
2 %
3 % default for l2 is: l2 halign=l, l2 valign=c. DO NOT USE the <...> notation
4 %
5 \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, , l2 valign=t] ++(2,0);
6 \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, , ] ++(0,2);
7 \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, l2 halign=c, l2 valign=b] ++(-2,0);
8 \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, l2 halign=r, l2 valign=c] ++(0,-2);
9 \draw (5,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, l2 halign=c, l2 valign=b] ++(2,0);
10 \draw (5,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, l2 halign=c, ] ++(0,2);
11 \draw (5,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, , l2 valign=t] ++(-2,0);
12 \draw (5,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, l2 halign=c, l2 valign=t] ++(0,-2);
13 \draw (10,2) to[R, l2={A=B} and X, a2={C=D} and Y] ++(0,-4);
14 \end{circuitikz}

```

For extra options about labels and annotations, please refer to section [5.6](#)

5.2 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 labeling 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 the direction of the electric field, like in **nooldvoltagedirection**, but batteries are fixed;

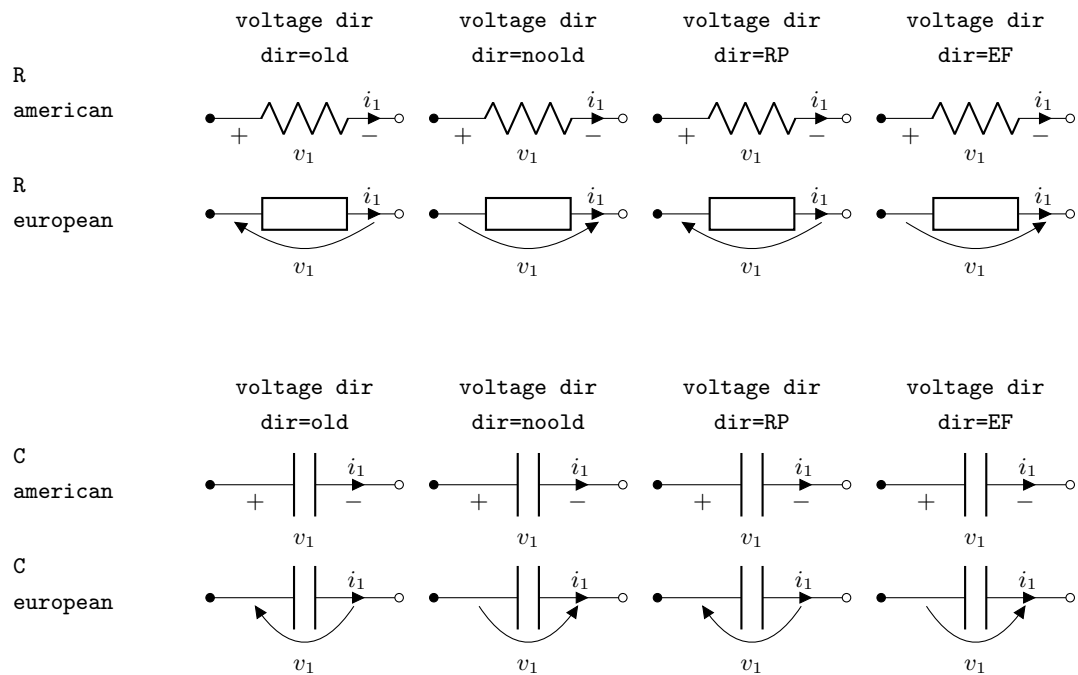
Notice that the four styles are designed to be used at the environment level: that is, you should use them at the start of your environment as in `\begin{circuitikz}[voltage dir=old]` ... and not as a key for single components, in which case the behavior is not guaranteed.

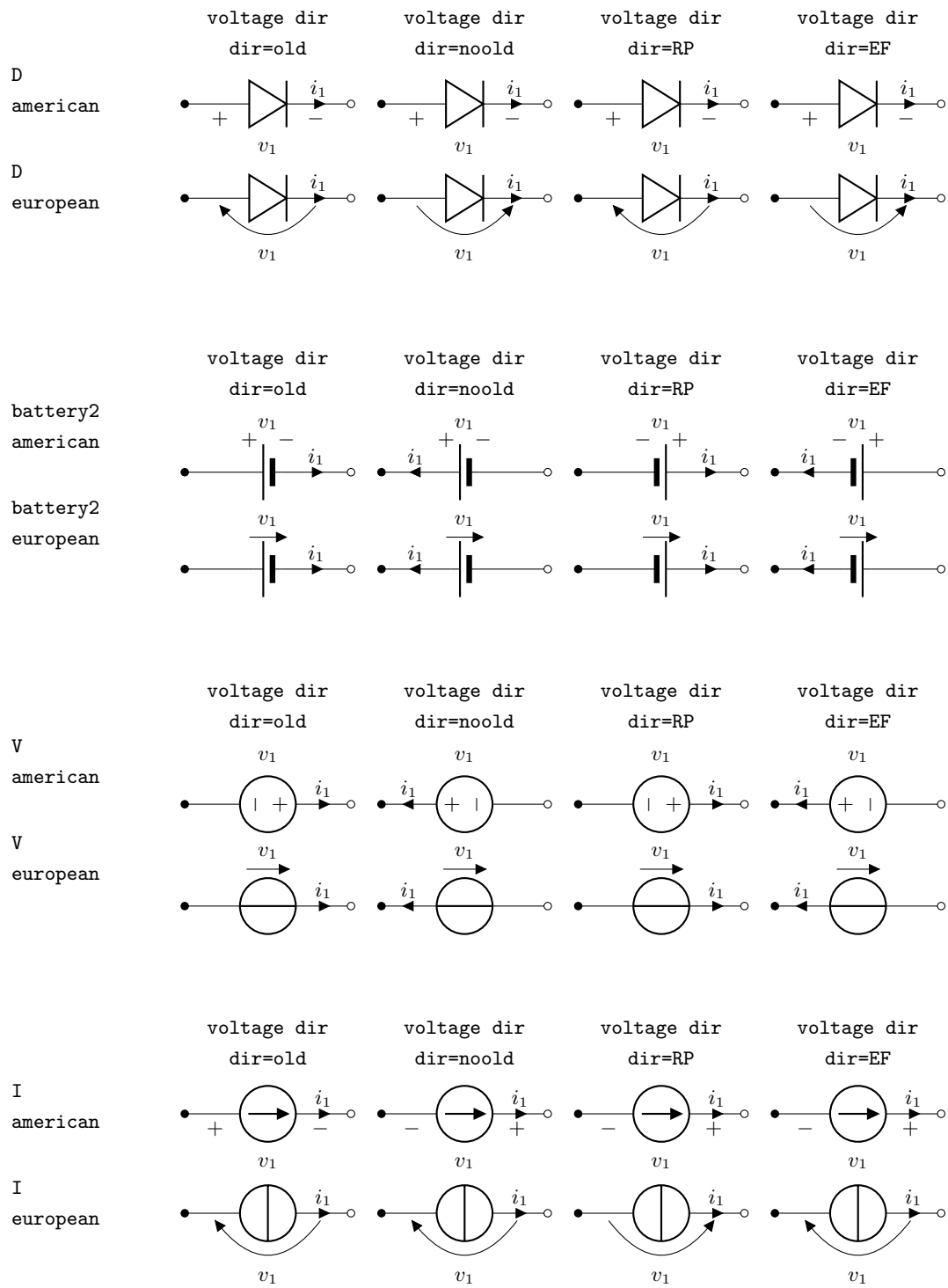
The standard direction of currents, flows and voltages are changed by these options; notice that the default drops in case of passive and active elements is normally different. Take care that in the case of **noold** and **EFvoltages** also the currents can switch directions. It is much easier to understand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

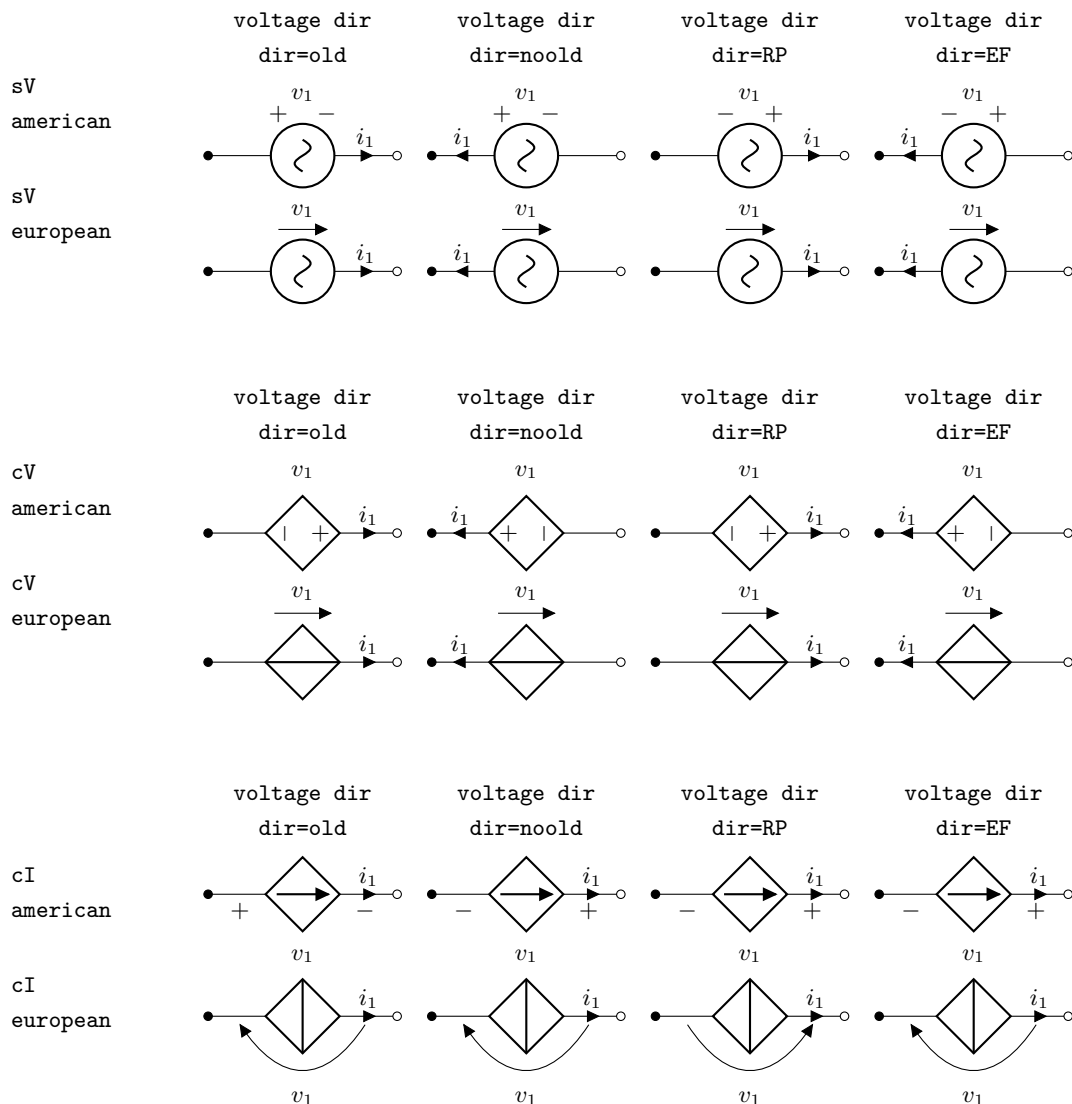
```

1 \foreach\element in {R, C, D, battery2, V, I, sV, cV, cI}{%
2   \noindent\ttfamily
3   \begin{tabular}{p{2cm}}
4     \element \\\ american \\\[15pt]
5     \element \\\ european \\\
6   \end{tabular}
7   \foreach\mode in {old, noold, RP, EF} {
8     \begin{tabular}{@{}l@{}}
9       \multicolumn{1}{c}{voltage dir} \\\
10      \multicolumn{1}{c}{dir=\mode} \\\[4pt]
11      \begin{tikzpicture}[
12        american, voltage dir=\mode,
13      ]
14        \draw (0,0) to[\element, *-o, v=$v_1$, i=$i_1$, ] (2.5,0);
15      \end{tikzpicture}\\\
16      \begin{tikzpicture}[
17        european, voltage dir=\mode,
18      ]
19        \draw (0,0) to[\element, *-o, v=$v_1$, i=$i_1$, ] (2.5,0);
20      \end{tikzpicture}
21    \end{tabular}
22    \medskip
23  }
24  \par
25 }

```





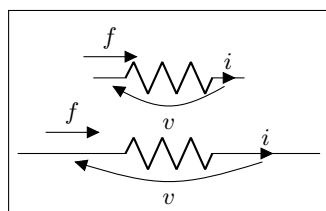


Obviously, you normally use just one between current and flows, but anyway you can change the direction of the voltages, currents and flows using the complete keys `i_>`, `i^<`, `i>_`, `i>^`, as shown in the following examples.

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

5.2.1 Common properties of voltages and currents

Currents, voltages and flows (see later) are positioned along, or across, the part of the wires that connect the inner component to the rest of the circuit. So, changing the length of the connection (the coordinates that embrace the `to[...]` command) will change the position of the components.

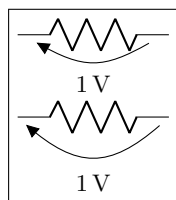


```

1 \begin{circuitikz}
2   \draw (-1,1) to[R, v=$v$, i=$i$, f>^=$f$] (1,1);
3   \draw (-2,0) to[R, v=$v$, i=$i$, f>^=$f$] (2,0);
4 \end{circuitikz}

```

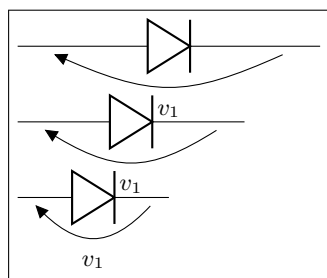
However, you can override the properties `voltage/distance from node` (default 0.5: how distant from the initial and final points of the path the arrow starts and ends or the plus and minus symbols are drawn) and `voltage/bump b` (how high the bump of the arrow is — how curved it is, default 1.5), and also `voltage/european label distance` (how distant from the normal position the voltage label will be, default 1.4) on a per-component basis, in order to fine-tune the voltages:



```
1\tikz \draw (0,0) to[R, v=1<\volt>] (2,0); \par
2\ctikzset{voltage/distance from node=.1}
3\ctikzset{voltage/bump b=2.5}
4\tikz \draw (0,0) to[R, v=1<\volt>] (2,0);
```

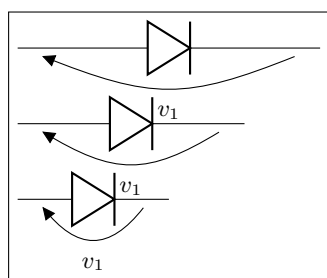
You can also use a global `ctikzset` on the key `voltage/distance from node` (and similar) that will act as a default value. Notice however that the specific component value **overrides** the global one, and several components have pre-defined overrides, so they will ignore the default value. The components that have out of the box predefined overrides for `distance from node` are `generic`, `ageneric`, `fullgeneric` and `memristor` (set to 0.4), and the ones that have it for `bump b` are `generic`, `ageneric`, `fullgeneric`, `memristor`, `tline`, `varistor`, `photoresistor`, `thermistor`, `thermistorntc`, `thermistorptc`, `ccapacitor`, `emptyzdiode`, `fullzdiode`, `emptythyristor`, `fullthyristor`, `emptytriac` and `fulltriac`, with several values (you can look at them in the file `pgfcirc.defines.tex`)

Notice also that normally `distance from node` is a relative displacement, computed on the node-component wire. So that this will put the start and stop point 1/4 of the way between node and component:



```
1\begin{circuitikz}
2\ctikzset{voltage/distance from node=0.25}
3\draw (0, 2) to[D, v=$v_1$] ++(4,0);
4\draw (0, 1) to[D, v=$v_1$] ++(3,0);
5\draw (0, 0) to[D, v=$v_1$] ++(2,0);
6\end{circuitikz}
```

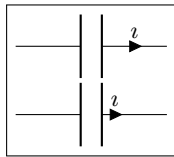
The value of `distance from node` can also be an absolute distance; in that case is measured from the start of the connection toward the component on the left (and symmetrically on the right), so this will put the start and end point to 0.25 cm from the start of the node:



```
1\begin{circuitikz}
2\ctikzset{voltage/distance from node=0.25cm}
3\draw (0, 2) to[D, v=$v_1$] ++(4,0);
4\draw (0, 1) to[D, v=$v_1$] ++(3,0);
5\draw (0, 0) to[D, v=$v_1$] ++(2,0);
6\end{circuitikz}
```

There is currently no way to specify the position at a fixed distance from the component (as opposed as from the node).

The same concept as `distance from node` applies to the key `current/distance` for the position of the current's arrow (and to `flow/distance` for the flow arrow position):

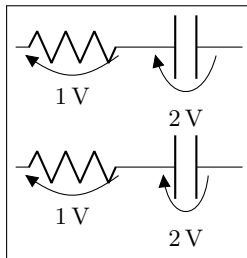


```

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

```

If you want to change those parameters by defining a component-specific key you have to use the internal name of the component (in the component list, is the `nodename` without the terminal “`shape`” part):



```

1\tikz \draw (0,0) to[R, v=1<\volt>] (1.5,0)
2      to[C, v=2<\volt>] (3,0); \par
3\ctikzset{bipoles/capacitor/voltage/distance from node/.initial
4          =.7}
5\tikz \draw (0,0) to[R, v=1<\volt>] (1.5,0)
6      to[C, v=2<\volt>] (3,0); \par

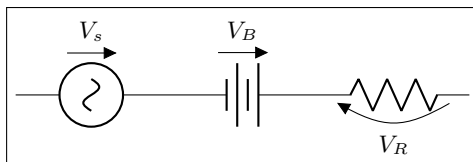
```

Note the `.initial`; you have to create such key the first time you use it. These kinds of adjustments are not guaranteed to work in future upgrades, though; if you have to create a key you are somehow touching the internal structure of the package; it’s much safer to create a style.

One common request is to change the style of the arrows (both head and line) of these elements. Voltages, currents and flows are part of the same path of the component, so this is not possible in simple way; you have to draw your own with TikZ commands using the facilities explained in section 5.8.

5.2.2 Special treatment for generators

The “active” elements (sources and batteries, mainly) are treated differently from passive elements, in the sense that the default current and voltage direction and position could be different⁸¹ following the chosen global voltage direction strategy (see section 5.2). If they change or not depend on both the element and the chosen `voltage dir` option.

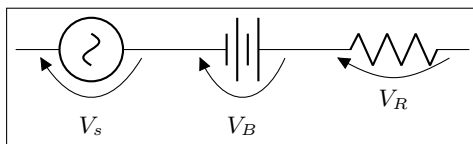


```

1\begin{tikzpicture}[]
2  \draw (0,0) to[sV, v=$V\_s$] ++(2,0)
3      to[battery, v=$V\_B$] ++(2,0)
4      to[R, v=$V\_R$] ++(2,0);
5\end{tikzpicture}

```

The consistency between symbols drawings and the default voltage and current directions are designed to work well *when this default is enabled*. If you want, though, you can override this behavior by “switching off” the source status of the component by setting the property `bipole/is voltage` to `false`:



```

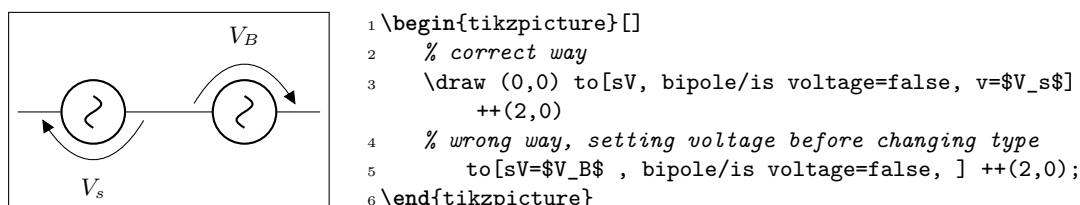
1\begin{tikzpicture}[]
2  \draw (0,0) to[sV, bipole/is voltage=false,
3      v=$V\_s$] ++(2,0)
4      to[battery, bipole/is voltage=false,
5      v=$V\_B$] ++(2,0)
6      to[R, v=$V\_R$] ++(2,0);
7\end{tikzpicture}

```

⁸¹This, in hindsight, has been a bad feature — and I’m partly responsible for it. But removing it would create *too small* variations in circuits, easily to go unnoticed, so it stays: nobody wants *wrong* circuits just by recompiling.

When you do this, **be careful** that (as you can see) the direction of the plain `v=...` option will change (please notice that this does not mean that it is incorrect, given that the voltage and current direction are arbitrary; in the case above, if the battery is a 3 V one, $V_B = -3$ V with the `RPvoltages` conventions).

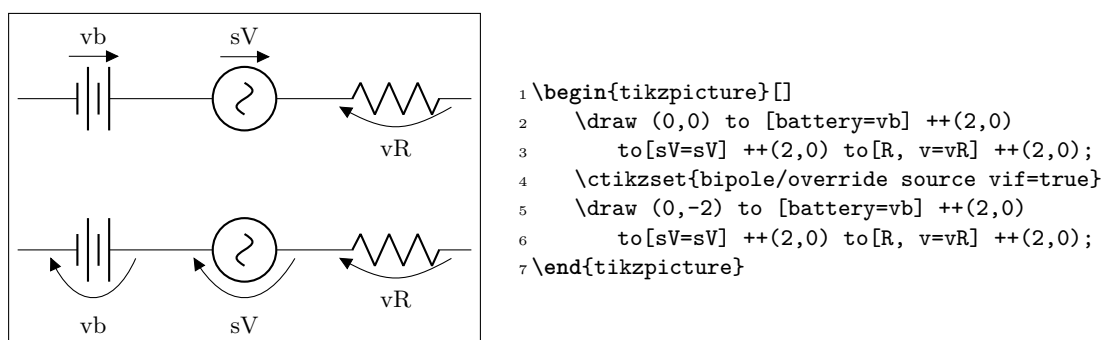
Also, notice that there is an ordering problem in the `to[...]` options: you have to switch the `is voltage` property off **before** setting the voltage, otherwise you will have a mix of the source-type and passive positioning:



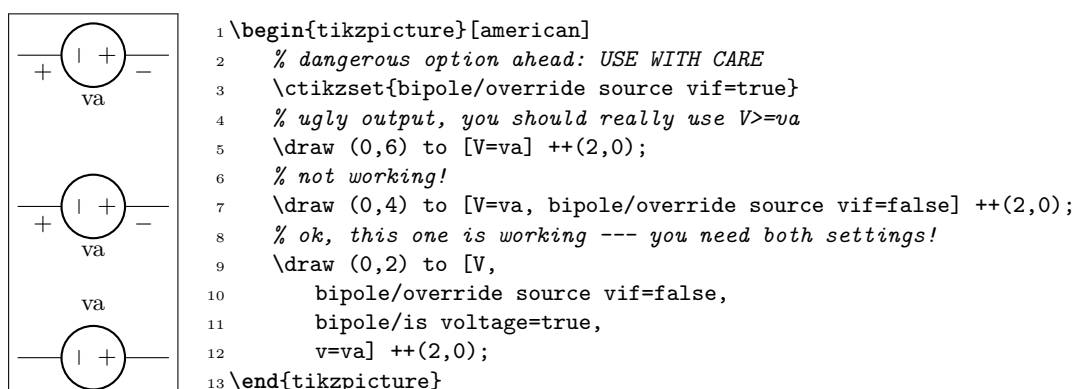
In the first `to[]` command, the voltage is set before changing the type (assigning a value to the name of the element is understood as a `v=...` command for voltage sources).

A similar switch is present for current generators, called `bipoles/.is current`, acting in a very similar way.

If you would prefer to switch to the `is voltage=false`, `is current=false` behavior by default, you can (since v1.4.4⁸²) by setting the option `bipole/override source vif` to `true`. This is *highly* experimental, so use with care.

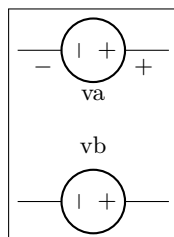


Notice that the option `override source vif` is “stronger” than the normal `is voltage`; so to locally re-set the behavior for just one source, you need to disable that *before* using a voltage designator.



Clearly, if you find yourself using the last component often, it is better to define a style, which will save you a lot of typing and help readability:

⁸²Suggested by user [@judober](#) on GitHub.



```

1 \tikzset{myV/.style={V, bipole/override source vif=false,
2   bipole/is voltage=true, v={#1}}}%
3 \begin{tikzpicture}[american]
4   % dangerous option ahead: USE WITH CARE
5   \ctikzset{bipole/override source vif=true}
6   \draw (0,0) to [V>=va] ++(2,0);
7   \draw (0,-2) to[myV=vb] ++(2,0);
8 \end{tikzpicture}

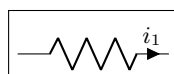
```

On the other way around, you could use styles to set `is voltage=false` only on the components you use and without using the global switch — which is the recommended way of doing it.

5.3 Currents

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

Basically, `^` and `_` control if the label is above or below the line (above and below **do** depend on the direction of the component path), and `<` and `>` the direction of the arrow; swapping them (from for example from `i^>` to `i>^`) will switch the side of the component where the symbol is drawn. See the following examples:



```

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

```



```

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

```



```

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

```



```

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

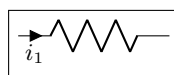
```



```

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

```



```

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

```



```

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

```

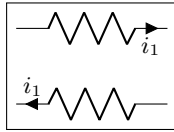


```

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

```

Also notice that the direction of the path is important:

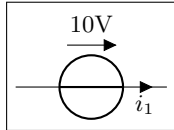


```

1 \begin{circuitikz}
2   \draw (2,1) to[R, i<=$i_1$] (0,1);
3   \draw (0,0) to[R, i<=$i_1$] (2,0);
4 \end{circuitikz}

```

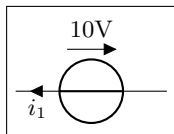
Default directions can change if the component is active or passive,⁸³ following the chosen global voltage direction strategy (see section 5.2).



```

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

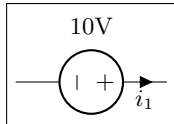
```



```

1 \begin{circuitikz}[voltage dir=EF]
2   \draw (0,0) to[V=10V, i_=$i_1$] (2,0);
3 \end{circuitikz}

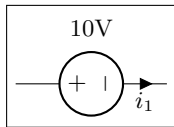
```



```

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

```

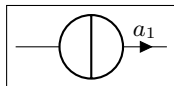


```

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

```

Current generators with the direct label (the one obtained by, for example, `I = something`) will treat it as a current:

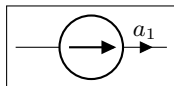


```

1 \begin{circuitikz}
2   \draw (0,0) to[I=$a_1$] (2,0);
3 \end{circuitikz}

```

If you use the option `americancurrent` or using the style `[american currents]` you can change the style of current generators.



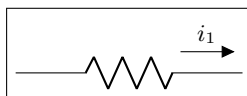
```

1 \begin{circuitikz}[american currents]
2   \draw (0,0) to[I=$a_1$] (2,0);
3 \end{circuitikz}

```

5.4 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.

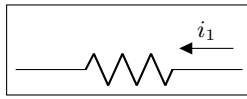


```

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

```

⁸³This is better explained in section 5.2.2



```

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

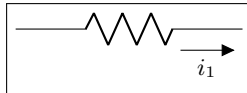
```



```

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

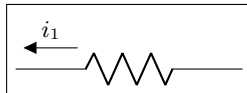
```



```

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

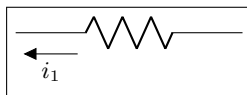
```



```

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

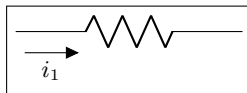
```



```

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

```



```

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

```

5.5 Voltages

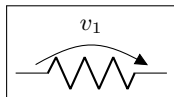
See the introduction at Currents and Voltages (section 5.2, page 186) for the default direction of the voltage and currents.

Voltages come in four different styles: European (with curved or straight arrows) and American (with signs that can stay near the wire or raised at the label level).

Direction and position of the symbols are controlled in the same way as for the currents (see section 5.3) with the `_<>` symbols.

5.5.1 European style

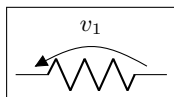
The default, with curved arrows. Use option `europeanvoltage` or style `[european voltages]`, or setting (even locally) `voltage=european`.



```

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

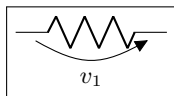
```



```

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

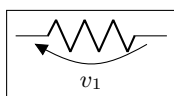
```



```

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

```

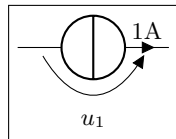


```

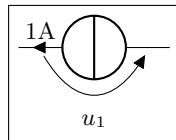
1 \begin{circuitikz}[european voltages]
2   \draw (0,0) to[R, v_<=$v_1$] (2,0);
3 \end{circuitikz}

```

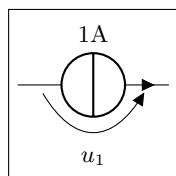
The default direction for active elements can change, depending on the global `voltage dir` setting, so be careful.



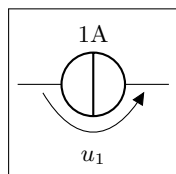
```
1 \begin{circuitikz}
2   \draw (0,0) to[I=1A, v_=$u_1$] (2,0);
3 \end{circuitikz}
```



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

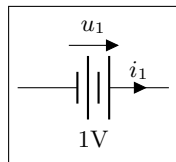


```
1 \begin{circuitikz}
2   \draw (0,0) to[I=$\sim$,l=1A, v_=$u_1$] (2,0);
3 \end{circuitikz}
```

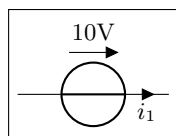


```
1 \begin{circuitikz}
2   \draw (0,0) to[I,l=1A, v_=$u_1$] (2,0);
3 \end{circuitikz}
```

Moreover, for historical reasons, voltage generators have differently looking arrows (they are straight even in curved European style).

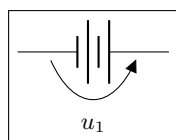


```
1 \begin{circuitikz}
2   \draw (0,0) to[battery,l=1V, v=$u_1$, i=$i_1$] (2,0);
3 \end{circuitikz}
```



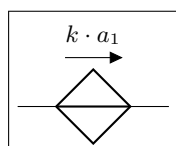
```
1 \begin{circuitikz}
2   \draw (0,0) to[V=10V, i_=$i_1$] (2,0);
3 \end{circuitikz}
```

You can change this last thing by forcing “off” the status of “voltage generator” of the component; but now the normal (passive) rule will apply, so, again, be careful and read section 5.2.2.



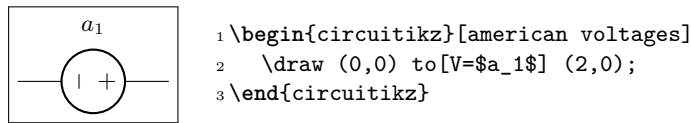
```
1 \begin{circuitikz}
2   \draw (0,0) to[battery, bipole/is voltage=false,
3     v>=$u_1$,] (2,0);
4 \end{circuitikz}
```

As for the currents, the direct label of voltage sources is passed as a voltage:



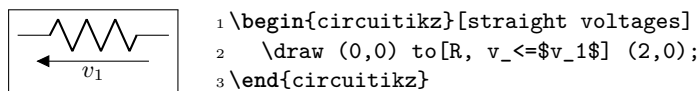
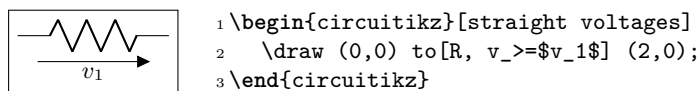
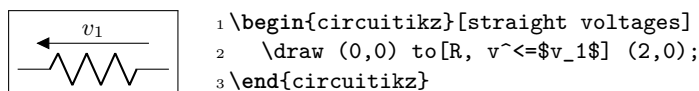
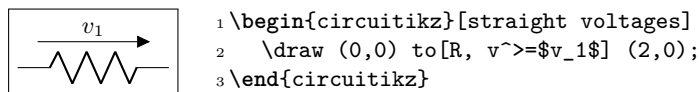
```
1 \begin{circuitikz}
2   \draw (0,0) to[cV=$k\cdot a_1$] (2,0);
3 \end{circuitikz}
```

The following results from using the option `americanvoltage` or the style `[american voltages]`.

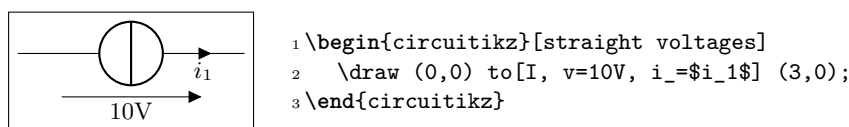
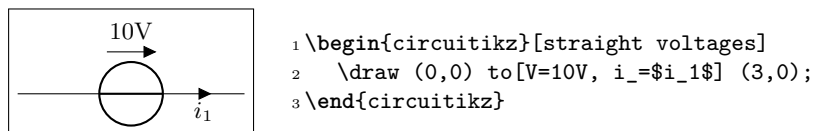


5.5.2 Straight European style

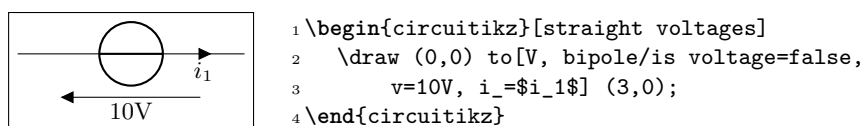
Using straight arrows. Use option `straightvoltages` or style `[straight voltages]`, or setting (even locally) `voltage=straight`.



Again, voltage generators are treated differently:

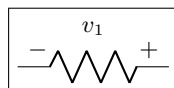


And you can override that with `bipole/is voltage` keeping into account that the default direction will be the one of passive components (see 5.2.2):

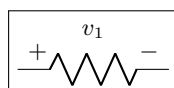


5.5.3 American style

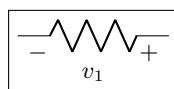
Use option `americanvoltage` or set `[american voltages]` or use the option `voltage=american`.



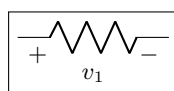
```
1 \begin{circuitikz}[american voltages]
2   \draw (0,0) to[R, v^>=$v_1$] (2,0);
3 \end{circuitikz}
```



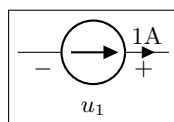
```
1 \begin{circuitikz}[american voltages]
2   \draw (0,0) to[R, v^<=$v_1$] (2,0);
3 \end{circuitikz}
```



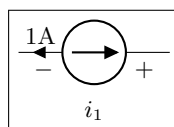
```
1 \begin{circuitikz}[american voltages]
2   \draw (0,0) to[R, v_>=$v_1$] (2,0);
3 \end{circuitikz}
```



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



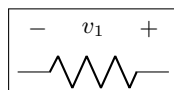
```
1 \begin{circuitikz}[american]
2   \draw (0,0) to[I=1A, v_=$u_1$] (2,0);
3 \end{circuitikz}
```



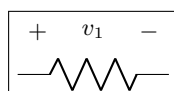
```
1 \begin{circuitikz}[american]
2   \draw (0,0) to[I<=1A, v_=$i_1$] (2,0);
3 \end{circuitikz}
```

5.5.4 Raised American style

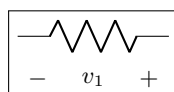
Since version 1.2.1, “raised” American voltages are available; to use them, set the style `[raised voltages]` or use the option `voltage=raised`. This is a version of the American-style voltage where the signs are raised to the level of the label. The label is centered between the two signs, and the position of the signs is calculated by supposing that the label itself will be pretty simple; if you have very big labels you will need to adjust the position with `voltage shift` and/or the `voltage/distance from node` properties (see section 5.2.1).



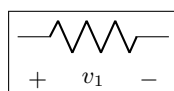
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,0) to[R, v^>=$v_1$] (2,0);
3 \end{circuitikz}
```



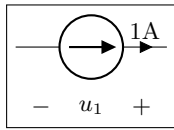
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,0) to[R, v^<=$v_1$] (2,0);
3 \end{circuitikz}
```



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



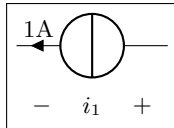
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,0) to[R, v_<=$v_1$] (2,0);
3 \end{circuitikz}
```



```

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

```



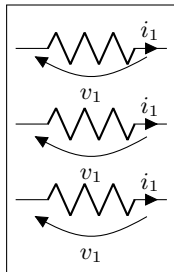
```

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

```

5.5.5 Voltage position

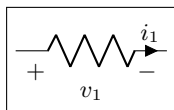
It is possible to move the arrows and the plus or minus signs away from the component with the key `voltages shift` (default value is 0, which gives the standard position):



```

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

```

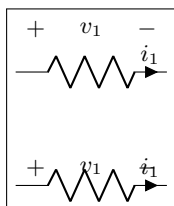


```

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

```

Negative values do work as expected:

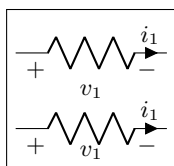


```

1 \begin{circuitikz}[raised voltages]
2   \draw (0,1.5) to[R, v^=$v_1$, i=$i_1$] ++(2,0);
3   \draw (0,0) to[R, v^=$v_1$, i=$i_1$,
4     voltage shift=-1.0] ++(2,0);
5 \end{circuitikz}

```

You can fine-tune the position of the + and - symbols and the label in independent way using `voltage/shift` (default 0.0 for the former and `voltage/american label distance` (the distance of the label from the lines of the symbols, default 1.4) for the latter.

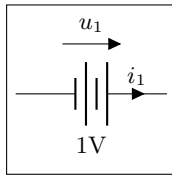


```

1 \begin{circuitikz}[american voltages]
2   \draw (0,1) to[R, v=$v_1$, i=$i_1$] ++(2,0);
3   % normally 1.4, make it tighter
4   \ctikzset{voltage/american label distance=0.5}
5   \draw (0,0) to[R, v=$v_1$, i=$i_1$] ++(2,0);
6 \end{circuitikz}

```

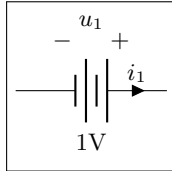
Notes that `american voltage` also affects batteries.



```

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

```

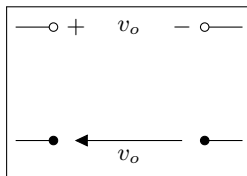


```

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

```

Additionally, the **open** component is treated differently; the voltage is placed in the middle of the open space⁸⁴:



```

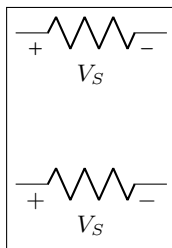
1 \begin{circuitikz}[american voltages]
2   \draw (0,1.5) -- ++(0.5,0)
3     to[open, v=$v_o$, o-o] ++(2,0) -- ++(0.5,0);
4   \draw (0,0) -- ++(0.5,0)
5     to[open, v=$v_o$, voltage=straight, *-] ++(2,0)
6     -- ++(0.5,0);
7 \end{circuitikz}

```

If you want or need to maintain the old behavior for **open** voltage, you can set the key **open voltage position** to **legacy** (the default is the new behavior, which corresponds to the value **center**).

5.5.6 American voltages customization

Since 0.9.0, you can change the font⁸⁵ 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:

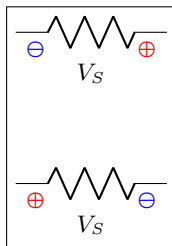


```

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

```

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



```

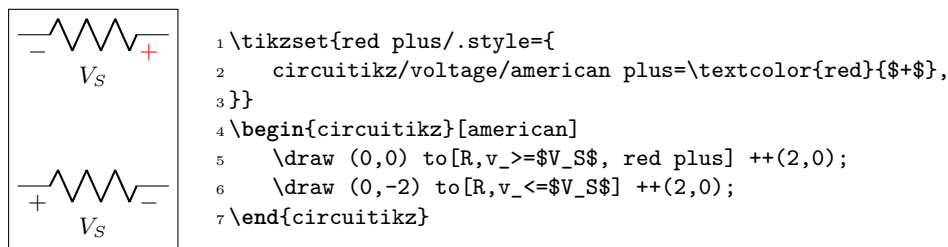
1 \begin{circuitikz}[american]
2   \ctikzset{voltage/american font=\scriptsize\boldmath}
3   \ctikzset{voltage/american plus=\textcolor{red}{+}\oplus}
4   \ctikzset{voltage/american minus=\textcolor{blue}{-}\ominus}
5   \draw (0,0) to[R,v_>=$V_S$] ++(2,0);
6   \draw (0,-2) to[R,v_<=$V_S$] ++(2,0);
7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

```

⁸⁴Since v1.1.2, thank to an [issue opened by user rhandley on GitHub](#).

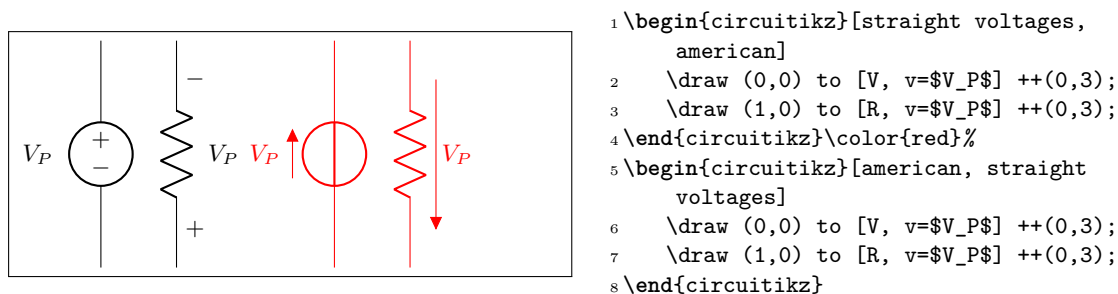
⁸⁵There was a bug before, noticed by the user [dzereb on tex.stackexchange.com](#) which made the symbols using different fonts in a basically random way. In the same page, user [campa](#) found the problem. Thanks!

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



5.5.7 Combining different styles

Due to an historical hiccup, you need to be careful if you want to mix styles, like for example having **american** styled components and straight voltages (which are basically **european** style, at least in CircuitikZ). The problem is that the order of style parameters can change the output⁸⁶ as you can see in the following example, where in the red case the voltage generator shape reverted to the **european** one.



This is arguably a bug, but fixing it (separating the voltage generator shapes from the voltage style) would break havoc with older circuits, so this will not be fixed for now.

5.6 Changing the style of labels, voltages, and other text ornaments

Since version 0.9.5, it is possible to change the style of bipole text ornaments (labels, annotations, voltages etc) by using the appropriate styles or keys. The basic style applied to the text is defined in the `/tikz/circuitikz` key directory and applied to every node that contains the text; you can also change them locally by using the `tikz` direct keys in local scopes.

For example, you can make all annotations small by using:

```
\ctikzset{bipole annotation style/.style={font=\small}}
```

And/or change (override) the setting in one specific bipole using:

```
...to[bipole annotation style={color=red}, R, a={Red note}]...
```

where the annotation will be in normal font (it has been reset!) and red, or append to the style:

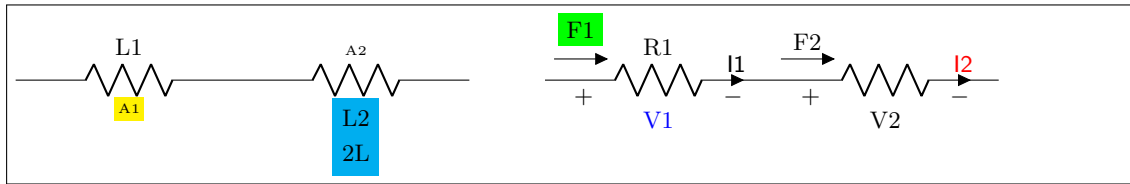
```
...to[bipole annotation append style={color=red}, R, a={Red small note}]...
```

Caveat: you have to put the style changing key at the start of the `to` arguments to have any effect⁸⁷.

The available styles and commands are `bipole label style`, `bipole annotation style`, `bipole voltage style`, `bipole current style`, and `bipole flow style`. The following example shows a bit of everything.

⁸⁶thanks to Stack Exchange user [Mads P Olesen](#) for noticing.

⁸⁷No, I do not know why. Hints and fixes are welcome.



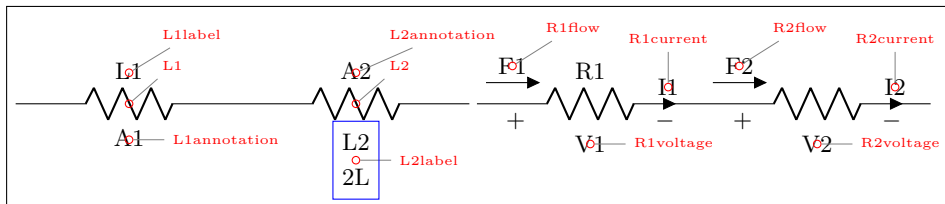
```

1 \begin{circuitikz}[american]
2   \ctikzset{bipole annotation style/.style={font=\tiny}}
3   \ctikzset{bipole current style/.style={font=\small\sffamily}}
4   \draw (0,0) to [bipole annotation append style={fill=yellow}, R=L1, a=A1] ++(3,0)
5     to [bipole label style={fill=cyan}, R, l2=L2 and 2L, a^=A2] ++(3,0);
6   \draw (7,0) to [bipole voltage style={color=blue},
7     bipole flow style={fill=green, outer sep=5pt},
8     R=R1, v=V1, i=I1, f>^=F1] ++(3,0)
9     to [bipole current append style={color=red}, R, v<=V2, i^=I2, f>^=F2] ++(3,0);
10 \end{circuitikz}

```

5.7 Accessing labels text nodes

Since 0.9.5, you can access all the labels nodes⁸⁸ using special node names. So, if you use **name** to give a name to the bipole node, you can also access the following nodes: **namelabel** (notice: no space nor any other symbol between **name** and **label**!), **nameannotation**, **namevoltage**, **namecurrent** and **nameflow**. Notice that the node names are available only if the bipole has an anchor or an annotation, of course.



```

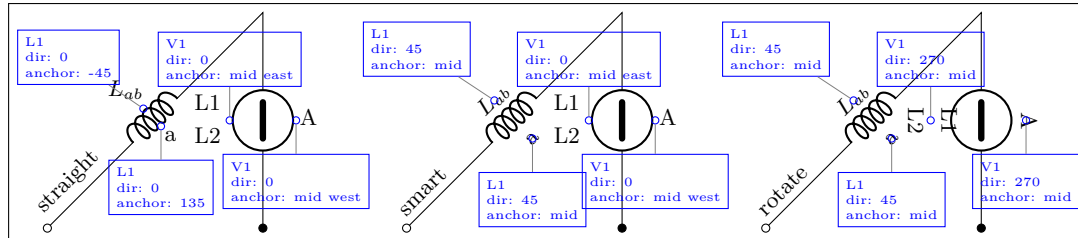
1 \newcommand{\marknode}[2][45]{%
2   \node[circle, draw, red, inner sep=1pt,
3     pin={[\red, font=\tiny]#1:#2}] at (#2.center) {};
4 }
5 \begin{circuitikz}[american]
6   \draw (0,0) to [R=L1, a=A1, name=L1] ++(3,0)
7     to [R, l2=L2 and 2L, a^=A2, name=L2] ++(3,0);
8   \marknode{L1} \marknode{L1label} \marknode[0]{L1annotation}
9   \marknode{L2} \marknode[0]{L2label} \marknode{L2annotation}
10  \draw[blue] (L2label.south west) rectangle (L2label.north east);
11  \draw (6.1,0) to [R=R1, v=V1, i=I1, f>^=F1, name=R1] ++(3,0)
12    to [R, v<=V2, i^=I2, f>^=F2, name=R2] ++(3,0);
13  \marknode[0]{R1voltage} \marknode[0]{R2voltage} \marknode[90]{R1current}
14  \marknode[90]{R2current} \marknode{R1flow} \marknode{R2flow}
15 \end{circuitikz}

```

If you want to have more access to the label positioning algorithm, since 1.2.5 you can access the label rotation using the command `\ctikzgetdirection{nodename}` (where node name is for example `L1label` or `L2annotation`), and the anchor used for positioning the node as `\ctikzgetanchor{component label}{type}`, where *component label* is, for example, `L1` and *type* is either *label* or *annotation* (notice that the syntax is slightly different, for implementation reasons). Those values are available only if the dipole declares a `l` or `a` keys; if you want them without any label you need to declare a blank one (like

⁸⁸The access to labels and annotations was present before, but not documented.

for example $1=\sim$). The following example gives an idea of the values of those macros for the three types of label positioning strategies.



```

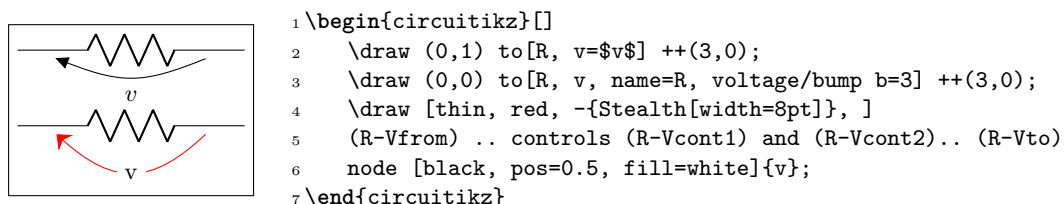
1 \newcommand{\marklabann}[3][45]{% [angle] {node label} {type: label or annotation}
2 \node[circle, draw, blue, inner sep=1pt,
3 pin={\draw, blue, font=\tiny, align=left}#1:#2 \ \ dir: \ctikzgetdirection{#2#3} \ \
4 anchor: \ctikzgetanchor{#2}{#3}}] at (#2#3.\ctikzgetanchor{#2}{#3}) {};}
5 \begin{tikzpicture}[scale=0.95, transform shape]
6 \foreach \style/\xdelta in {straight/0, smart/5, rotate/10} {
7 \begin{scope}[xshift=\xdelta cm]
8 \ctikzset{label/align = \style}
9 \draw (0,0) node[above right, rotate=45]{\style}
10 to[L, o-, l=$L_{ab}$, v, name=L1, a=a] ++(3,3)
11 to[ceV, -, v, name=V1, l2=L1 and L2, a^=A] ++(0,-3);
12 \marklabann[135]{L1}{label}
13 \marklabann[-90]{L1}{annotation}
14 \marklabann[90]{V1}{label}
15 \marklabann[-90]{V1}{annotation}
16 \end{scope}}
17 \end{tikzpicture}

```

5.8 Advanced voltages, currents and flows

Since version 1.2.1⁸⁹, it is possible to access the anchors of the “ornaments” — voltage, current and flows, together with some additional information that makes it possible to personalize them. Normally, voltages and flow and currents are drawn into the path of the bipoles, so that it is not possible, for example, to change the line type or color of the arrows, or the type of arrows⁹⁰. Access to the anchors allows you to do all these things, and more.

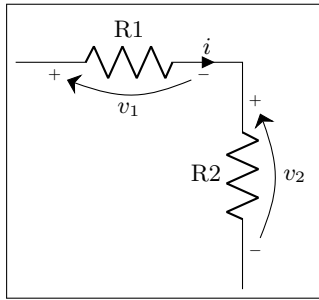
For example, you can do something like this:



Or, for example, to have a different voltage style; normally you would define a macro:

⁸⁹some options have been added in v1.4.1

⁹⁰in regular voltages, the arrows are not real TikZ arrows, but the auxiliary arrow shapes of CircuiTikZ

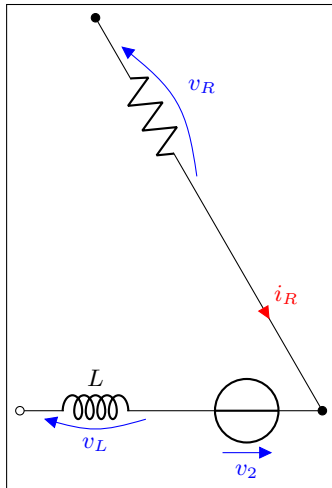


```

1 \begin{circuitikz}[voltage shift=0.5]
2   \def\eurVPM#1#2{% node, label
3     \draw [thin, --{Stealth[width=8pt]}], shorten >=5pt,
4     shorten <=5pt] (#1-Vfrom) node[font=\tiny]{\$-\$}
5     .. controls (#1-Vcont1) and (#1-Vcont2)..
6     (#1-Vto) node[font=\tiny]{\$+\$}
7     node[pos=0.5,anchor=\ctikzgetanchor{#1}{Vlab}]{#2};}
8   \draw (0,0) to[R=R1, v, i=i] ++(3,0)
9     to[R, l=R2, v^, name=R2] ++(0,-3);
10  \eurVPM{R1}{\$v_1\$} \eurVPM{R2}{\$v_2\$}
11 \end{circuitikz}

```

Since v1.4.1 you can also keep the voltage, current and flow labels and suppress the output of the symbols (arrows or plus/minus depending on the style) with the keys `no v symbols`, `no i symbols`, `no f symbols` (there are also the corresponding `v symbols`, `i symbols` and `f symbols` in case you want to switch the behavior off/on globally). This for example simplify an often requested feature, like having all the current in one color and the voltages in another one, which is not possible natively because the arrows are part of the same path. One possible implementation of that is the following one:



```

1 \newcommand{\iarronly}[1]{% name
2   \node [curarrow, color=red, anchor=center,
3     rotate=\ctikzgetdirection{#1-Iarrow}] at (#1-Ipos) {};
4 }
5 \newcommand{\varronly}[1]{% name
6   \draw [color=blue] (#1-Vfrom) .. controls (#1-Vcont1)
7     and (#1-Vcont2).. (#1-Vto) node [curarrow,
8     sloped, anchor=tip, allow upside down,pos=1]{};
9 }
10 \begin{circuitikz}[]
11   \ctikzset{!vi/.style={no v symbols, no i symbols}}
12   \ctikzset{bipole voltage style/.style={color=blue},
13     bipole current style/.style={color=red}}
14   \draw (120:6) to[R, *, name=R, v^=$v_R$, !vi]
15     (120:3) to[short, i=$i_R$, name=SR, !vi] (0,0);
16   \draw (180:4) to[L, o-, l=$L$, name=L2, v=$v_L$, !vi]
17     (180:2) to[V, -*, name=V2, v_=$v_2$, !vi] (0:0);
18   \iarronly{SR}\varronly{R}\varronly{L2}\varronly{V2}
19 \end{circuitikz}

```

5.8.1 Activating the anchors

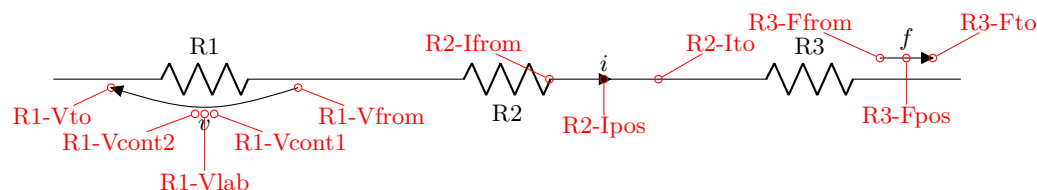
You will have access to the anchors for voltages, currents and flows when, in the bipole, you have both a `v`, `i`, `f` specification (one or more of them) **and** a `name` key, to give the bipole a name. Otherwise, the anchors and the associated functions are not defined. To suppress the normal output of the `v`, `i`, `f` keys, you can use such keys without any argument, like in the previous example; notice that the `_` and `^` modifiers work as expected.

The following line of resistors has been drawn with the following commands; it is used to show the name of the available anchors.

```

1 \draw (0,0) to[R=R1, v=$v$, name=R1] ++(4,0)
2   to[R, l=R2, i=$i$, name=R2] ++(4,0)
3   to[R=R3, f=$f$, name=R3] ++(4,0);

```



The meaning of the anchors is the following:

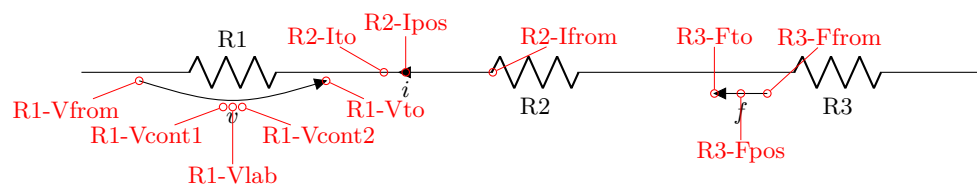
- **Vfrom** and **Vto** are the main points where the voltage information is given: start and end point of the arrow, or position of the + or – sign. This is the same for the **Ffrom** or **Fto** anchors for flows; for inline currents, the corresponding **Ifrom** and **Ito** mark the wire segment where the arrowhead is positioned (at the specified **current/distance** fraction). The direction of the arrow is available using the auxiliary macro `\ctikzgetdirection` (see below).
- **Vcont1** and **Vcont2** are the control points for the curved arrow (see the examples above); in the case of straight arrows or american-style voltages, they are set at the midpoint between **Vfrom** and **Vto**.
- **Vlab** is where the text label for the voltage is normally positioned. The anchor used for such label is available using the auxiliary macro `\ctikzgetanchor` (see below)
- **Ipos** and **Fpos** are the position for the arrowhead or the small flow arrow (which is a `curarrow` or `flowarrow` node normally) is positioned, respectively. The label is then added to the correct side of it using the anchor available via `\ctikzgetanchor` (see below, 5.8.2). In this case, the exact position of the label is not available if you do not position the element, for this there is no **Flab** or **Ilab** coordinate; you have to use the **Fpos** and **Ipos** coordinate with the corresponding **Ilab** and **Flab** anchors.

Changing the options of the elements will change the anchors accordingly:

```

1 \ctikzset{current/distance=0.2}
2 \draw (0,0) to[R=R1, v=>v$, name=R1] ++(4,0)
3     to[R, l_=R2, i<_=$i$, name=R2] ++(4,0)
4     to[R, l_=R3, f<_=$f$, name=R3] ++(4,0);

```

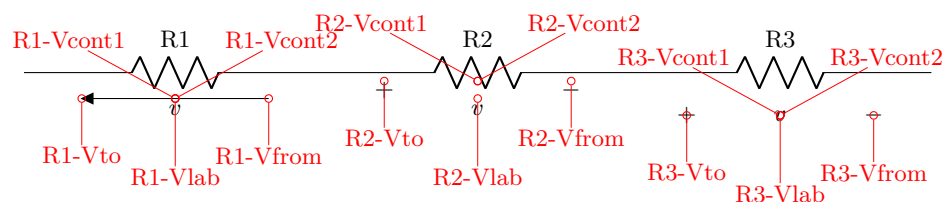


Obviously, the anchors follow the voltage style you choose:

```

1 \draw (0,0) to[R=R1, v=>v$, name=R1, voltage=straight] ++(4,0)
2     to[R=R2, v=>v$, name=R2, voltage=american] ++(4,0)
3     to[R=R3, v=>v$, name=R3, voltage=raised] ++(4,0);

```



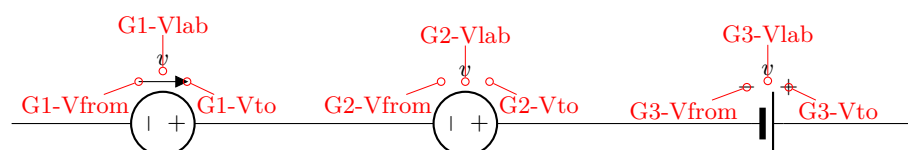
Notice the position of the control points, as well as the fact that the anchor available with `\ctikzgetanchor` is applied to **Vfrom** and **Vto** symbols, too.

Finally, as ever, generators are treated differently, but you have all your anchors too.

```

1 \ctikzset{american}
2 \draw (0,0) to[V=v$, name=G1, voltage=european] ++(4,0)
3     to[V=v$, v=>v$, name=G2, voltage=american] ++(4,0)
4     to[battery2, v=>v$, name=G3, voltage=raised] ++(4,0);

```

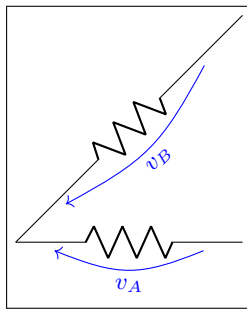


5.8.2 Auxiliary information

When the anchors are activated, there are additional macros that you can use:

- `\ctikzgetanchor{<name>}{<anchor>}`: *name* is the name of the bipole, and *anchor* can be `Vlab`, `Fpos` or `Ipos`. This macro expands to the normal anchor position (something like `north`, `south` or `west`). Notice that if you have not activated the corresponding anchor, the content of this macro is not specified. It could be equivalent to `\relax` (basically, empty) or contains the anchor of a bipole with the same name from another drawing — it's a global macro like the coordinates.
- `\ctikzgetdirection{<name>}`: a number which is the direction of the *named* bipole.
- `\ctikzgetdirection{<name>-Iarrow}`: a number which is the direction of the current arrow requested for the *named* bipole; using `<name>-Farrow` you get the same information for flow arrows.

For example, you could like the voltage label oriented with the bipole:



```

1 \begin{circuitikz}[]
2   \def\myvv#1#2{%
3     \draw [thin, blue, ->,]
4       (#1-Vfrom) .. controls (#1-Vcont1) and (#1-Vcont2).. (#1-Vto)
5     node [pos=0.5, below,
6           rotate=\ctikzgetdirection{#1}] at (#1-Vlab) {#2}; }
7   \draw (0,0) to[R, v, name=A] ++(3,0);
8   \draw (0,0) to[R, v, name=B] ++(3,3);
9   \myvv{A}{v_A}\myvv{B}{v_B}
10 \end{circuitikz}

```

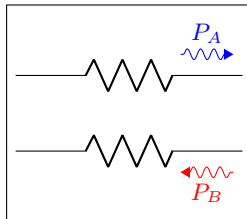
Or you could use the anchor to substitute the flow with a fancy one and still position the label automatically; suppose you have the following definition in your preamble (see TikZ manual, “Path decorations”):

```

1 % requires \usetikzlibrary{decorations, decorations.pathmorphing}
2 \tikzset{%
3   lray/.style={decorate, decoration={
4     snake, amplitude=2pt, pre length=1pt, post length=2pt, segment length=5pt},
5     -Triangle,
6   }}

```

You can then define a kind of “power flow” style:



```

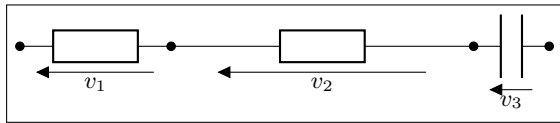
1 \begin{circuitikz}[]
2   \newcommand\myff[3][blue]{% [opt: color] node label
3     \draw [lray, #1, ] (#2-Ffrom) -- (#2-Fto)
4       node [anchor=\ctikzgetanchor{#2}{Flab}, inner sep=4pt]
5         at (#2-Fpos) {#3};}
6   \draw (0,1) to[R, f, name=A] ++(3,0);
7   \draw (0,0) to[R, f_<, name=B] ++(3,0);
8   \myff{A}{P_A}\myff[red]{B}{P_B}
9 \end{circuitikz}

```

5.8.3 Fixed voltage arrows: an example of advanced voltage usage

An interesting application of the advanced voltage is to have fixed length straight voltage arrows.⁹¹ The normal voltage arrows length depends not on the component length but on the node distance (this is the behavior since when the voltages were first introduced, so it can't be changed).

⁹¹This was suggested by users Franklin and Zarko in [a question on tex.stackexchange.com](https://tex.stackexchange.com)



```

1 \begin{circuitikz}[european,]
2   \ctikzset{voltage=straight}
3   \draw (0,0) to[R,v=$v_1$,*-*] ++(2,0) to[R, v<=$v_2$] ++(4,0) to[C, *-, v=$v_3$] ++(1,0);
4 \end{circuitikz}

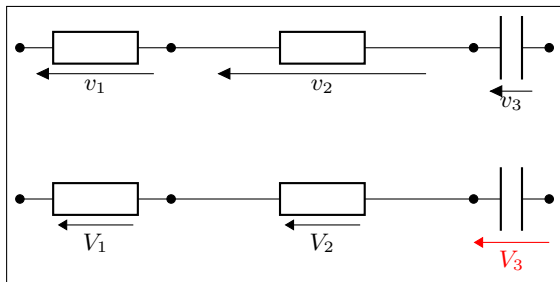
```

Using the advanced voltage interface mechanism, you can for example design voltages that are of fixed lengths; in the example below the new **xparse** method for defining commands is used, so that we can have a couple of different optional arguments:

```

1 \NewDocumentCommand{\fixedvlen}{0{0.5cm} m m 0{}}{% [semilength]{node}{label}[extra options]
2   % get the center of the standard arrow
3   \coordinate (#2-Vcenter) at ($(#2-Vfrom)!0.5!(#2-Vto)$);
4   % draw an arrow of a fixed size around that center and on the same line
5   \draw[-Triangle, #4] ($(#2-Vcenter)!#1!(#2-Vfrom)$) -- ($(#2-Vcenter)!#1!(#2-Vto)$);
6   % position the label as in the normal voltages
7   \node[anchor=\ctikzgetanchor{#2}{Vlab}, #4] at (#2-Vlab) {#3};
8 }

```



```

1 \begin{circuitikz}[european,]
2   \ctikzset{voltage=straight}
3   \draw (0,2) to[R,v=$v_1$,*-*] ++(2,0) to[R, v<=$v_2$] ++(4,0) to[C, *-, v=$v_3$] ++(1,0);
4   \draw (0,0) to[R,v=,name=v1,*-*] ++(2,0) to[R, v<=, name=v2] ++(4,0) to[C, *-, v, name=v3] ++(1,0);
5   \fixedvlen{v1}{V_1$}
6   \fixedvlen{v2}{V_2$}
7   \fixedvlen{v3}{V_3$}[red]
8 \end{circuitikz}

```

Notice that with a coherent naming you can use a **\foreach** loop for the last three lines.

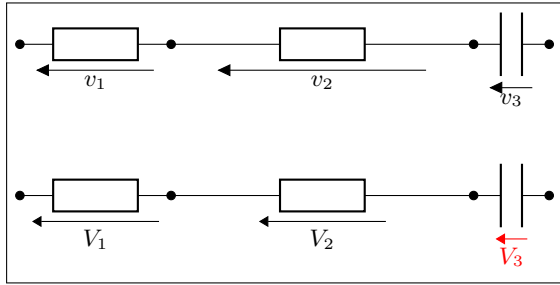
You can also notice that the arrow is not exactly the same as other arrows in the circuit; if you want them to be exactly the same, you can use a trick to get the default CircuiTiKZ arrow size — please look at [this answer by Romano on tex.stackexchange.com](https://tex.stackexchange.com/a/444444).

Another possibility is to have the arrow length based on the length of the component; for example you can use this code:

```

1 \NewDocumentCommand{\compvlen}{0{1.5} m m 0{}}{% [relative length]{node}{label}[extra options]
2   % get the center of the standard arrow
3   \coordinate (#2-Vcenter) at ($(#2-Vfrom)!0.5!(#2-Vto)$);
4   % draw an arrow of a size proportional to the component length
5   % around that center and on the same line
6   % the component length is calculated using the let...in with the left and right anchors
7   % and multiplied by the relative length
8   \draw[-Triangle, #4] let \p1=(#2.left), \p2=(#2.right), \n1={0.5*#1*vecclen(\x2-\x1,\y2-\y1)}
9   in ($(#2-Vcenter)!#1!(#2-Vfrom)$) -- ($(#2-Vcenter)!#1!(#2-Vto)$);
10  % position the label as in the normal voltages
11  \node[anchor=\ctikzgetanchor{#2}{Vlab}, #4] at (#2-Vlab) {#3};
12 }

```

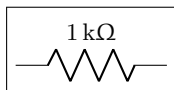
```

1 \begin{circuitikz}[european,]
2   \ctikzset{voltage=straight}
3   \draw (0,2) to[R,v=$v_1$,*-*] ++(2,0) to[R, v<=$v_2$] ++(4,0) to[C, *-, v=$v_3$] ++(1,0);
4   \draw (0,0) to[R,v=,name=v1,*-*] ++(2,0) to[R, v<=, name=v2] ++(4,0) to[C, *-, v, name=v3] ++(1,0);
5   \compvlen{v1}{V_1$}
6   \compvlen{v2}{V_2$}
7   \compvlen{v3}{V_3$}[red]
8 \end{circuitikz}

```

5.9 Integration with siunitx

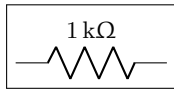
If the option `siunitx` is active⁹², then the following are equivalent (this will **not** work in ConTeXt, it has been disabled in upstream ConTeXt, in favor of [its own units module](#)):



```

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

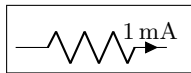
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, l=$\SI{1}{\kilo\ohm}$] (2,0);
3 \end{circuitikz}

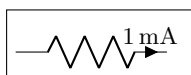
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, i=1<\milli\ampere>] (2,0);
3 \end{circuitikz}

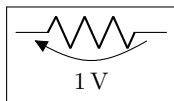
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, i=$\SI{1}{\milli\ampere}$] (2,0);
3 \end{circuitikz}

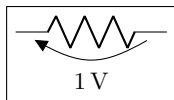
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, v=1<\volt>] (2,0);
3 \end{circuitikz}

```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, v=$\SI{1}{\volt}$] (2,0);
3 \end{circuitikz}

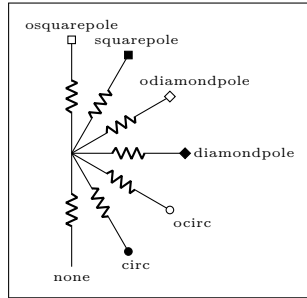
```

⁹²This option is still experimental — personally (Romano) I would advise using the normal `\SI{}{}` syntax, or the `\qty{}{}` one for `siunitx v3` and newer.

6 Using bipoles in circuits

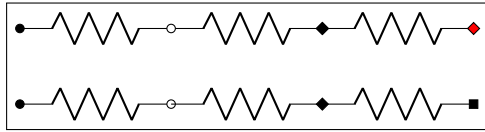
6.1 Nodes (also called poles)

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`⁹³ (see section 4.12).



```
1 \begin{circuitikz}
2   \ctikzset{bipoles/length=.5cm, nodes width=0.1}%small
   components, big nodes
3   \foreach \a/\p [evaluate=\a as \b using (\a+180)] in
4     {-90/none, -60/circ, -30/ocirc, 0/diamondpole, 30/
       odiamondpole, 60/squarepole, 90/osquarepole}
5     \draw (0,0) to[R, bipole nodes={none}{\p}] ++(\a:1.5)
       node[font=\tiny, anchor=\b]{\p};
6 \end{circuitikz}
```

These bipole nodes are added after the path is drawn, as every node in TikZ — 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.



```
1 \begin{circuitikz}
2   \draw (0,0) to[R, *-o] ++(2,0) to[R, -d] ++(2,0)
3     to[R, bipole nodes={diamondpole}{odiamondpole, fill=red}] ++(2,0);
4   \draw (0,-1) to[R, *-o] ++(2,0) ;
5   \draw (2,-1) to[R, -d] ++(2,0) to[R, bipole nodes={none}{squarepole}] ++(2,0);
6 \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:



```
1 \begin{circuitikz}
2   \ctikzset{-s/.style = {bipole nodes={none}{osquarepole, fill=red}}}
3   \draw (0,0) to[R, -s] ++(2,0);
4 \end{circuitikz}
```

There are several predefined shorthand as the above; in the following pages you can see all of them.



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

⁹³You 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”.



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



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



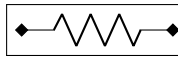
```
1\begin{circuitikz}
2  \draw (0,0) to[R, -*-] (2,0);
3\end{circuitikz}
```



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



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



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



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



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



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



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



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



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



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



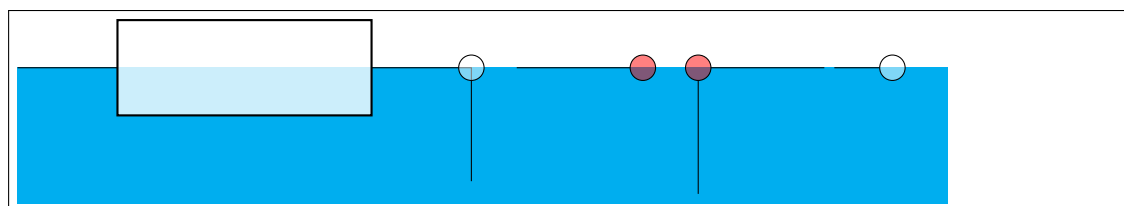
```
1\begin{circuitikz}
2  \draw (0,0) to[R, d-*] (2,0);
3\end{circuitikz}
```

6.1.1 Transparent poles

“Open-poles” terminals (`ocirc`, `odiamondpole`, and `osquarepole`) are normally filled with the background color at full opacity. This is because, for simplicity of operation, the nodes are placed *after* the wires are drawn and have to “white-out” the underlying lines.

Anyway, *if you know what you are doing*, you can change it with the key `poles/open fill opacity` (with `\ctikzset`) or the style `open poles opacity`. Notice that you will have artifacts if you don’t use the border anchors of the poles to connect wires, and you need to do that by hand.

Notice that in poles, the opacity is *always* selected with these keys, and it overrides the opacity of the draw commands (when not set explicitly is as if it is set to 1.0, i.e., full opaque). This is because you normally do not want unfilled poles!



```

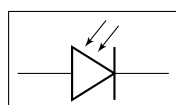
1 \begin{circuitikz}[scale=3, transform shape]
2   \fill[cyan] (0,0) rectangle (4.1,-0.6);
3   \ctikzset{open poles fill=red}
4   \tikzset{open poles opacity=0.5}
5   % automatic positioning when opacity is not 1.0 creates artifacts
6   % note that the global fill opacity affects the "generic shape", but not the poles!
7   % the fill color of the poles, instead, goes with the component
8   \draw[fill opacity=0.8] (0,0) to[generic, fill=white, -o] ++(2,0) ---++(0,-0.5);
9   % \draw (0,0) to[generic, fill=white, -o] ++(2,0) ---++(0,-0.5);
10  % you have to use manual positioning
11  \draw (2.2,0) -- ++(0.5,0) node[ocirc, anchor=180, fill opacity=0.5]{};
12  \draw (3,0) node[ocirc, fill opacity=0.5](B){} (B.0) ---++(0.5,0) (B.-90)
13    ---++(0,-0.5);
14  % maybe really useful only for terminals going out of the circuit...
15  % notice that in node commands you can specify the opacity directly
16  \draw (3.6,0) -- ++(0.2,0) node[ocirc, fill=white, fill opacity=0.5, anchor=180]{};
17 \end{circuitikz}

```

You also have the similar keys for the “full” poles (albeit they are probably not useful at all).

6.2 Mirroring and Inverting

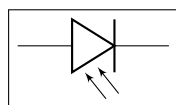
Bipole paths can also be mirrored and inverted (or reverted) to change the drawing direction.



```

1 \begin{circuitikz}
2   \draw (0,0) to[pD] (2,0);
3 \end{circuitikz}

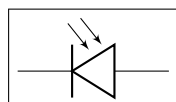
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[pD, mirror] (2,0);
3 \end{circuitikz}

```

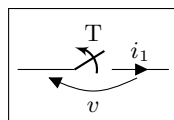


```

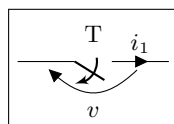
1 \begin{circuitikz}
2   \draw (0,0) to[pD, invert] (2,0);
3 \end{circuitikz}

```

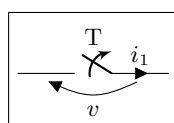
Placing labels, currents and voltages also works, please note, that mirroring and inverting does not influence the positioning of labels and voltages. Labels are by default above/right of the bipole and voltages below/left, respectively.



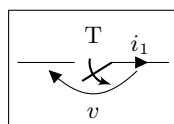
```
1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}
```



```
1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T, mirror, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}
```

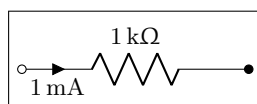


```
1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T, invert, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}
```

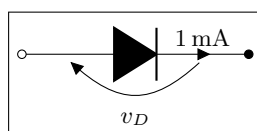


```
1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T,mirror,invert, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}
```

6.3 Putting them together



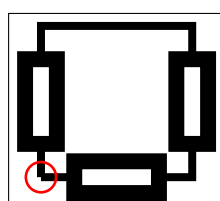
```
1 \begin{circuitikz}
2   \draw (0,0) to[R=1<\kilo\ohm>,
3     i>_1<\milli\ampere>, o-*] (3,0);
4 \end{circuitikz}
```



```
1 \begin{circuitikz}
2   \draw (0,0) to[D*, v=$v_D$,
3     i=1<\milli\ampere>, o-*] (3,0);
4 \end{circuitikz}
```

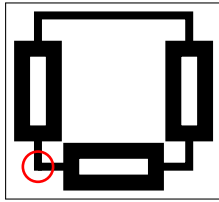
6.4 Line joins between Path Components

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!):



```
1 \begin{tikzpicture}[line width=3pt,european]
2   \draw (0,0) to[R]++(2,0)to[R]++(0,2)
3     --++(-2,0)to[R]++(0,-2);
4   \draw[red,line width=1pt] circle(2mm);
5 \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:



```

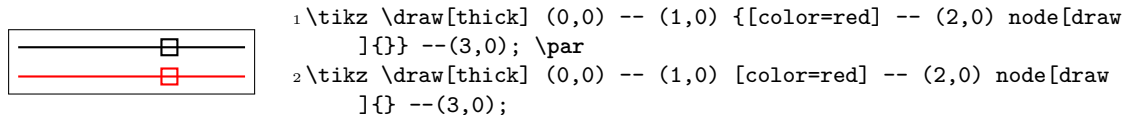
1 \begin{tikzpicture}[line width=3pt,european]
2 \draw (0,0) to[R]++(2,0)to[R]++(0,2)
3   --++(-2,0)to[R,-.]++(0,-2);
4 \draw[red,line width=1pt] circle(2mm);
5 \end{tikzpicture}

```

7 Colors

Color support in CircuiTikZ has been quite limited up to version 1.5.1; from that one onward there has been an effort to make components' behavior more intuitive.

Part of the problem is how colors in paths are treated by TikZ itself; you can see part of the discussion on [this issue](#) and in [this question on TeX.SX](#) — many thanks to @muzimuzhi for helping there. Basically, nodes are drawn *after* the path is completed, and color is applied to the path at the end. Look at this code (pure TikZ, no CircuiTikZ here):



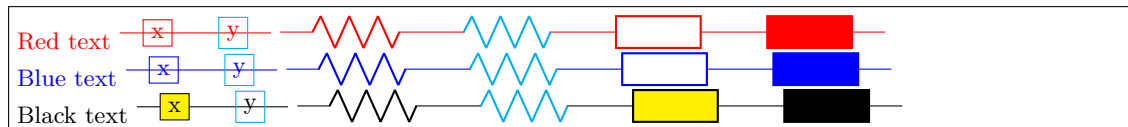
So the path is drawn with the last “effective” (in current group) color. The deferred behavior of the color properties is very difficult to track for CircuiTikZ, especially when the shorthand `color-name` (e.g., `red` instead of `color=red`) is used. CircuiTikZ will try to keep track of the colors specified by `color=...` and `fill=...`, but if you use the implicit way (`\draw[red]...`) it often fails.

If you're adventurous, you can try to add

```
\usepackage{regexpatch}\ctikzPatchImplicitColor
```

after loading CircuiTikZ, and it will try to patch the default commands to keep track of the “current color”; if it fails will give a warning like `patch failed, use only explicit color=...`. This is, unfortunately, not compatible with package `xpatch` (and much others, which load `xpatch`).⁹⁴

Before 1.5.0, CircuiTikZ used black as the default color. Now it tries to follow the current color, as TikZ does normally; but notice that there is a difference with the fill strategy:



```

1\color{red}
2Red text
3\tikz \draw (0,0) -- node[draw] {x} (1,0) -- node[draw=cyan] {y} (2,0);
4\tikz \draw (0,0) to[R] (2,0) to[R,color=cyan] (4,0) to [generic] (6,0) to [fullgeneric
] (8,0);
5
6\color{blue}
7Blue text
8\tikz \draw (0,0) -- node[draw] {x} (1,0) -- node[draw=cyan] {y} (2,0);
9\tikz \draw (0,0) to[R] (2,0) to[R,color=cyan] (4,0) to [generic] (6,0) to [fullgeneric
] (8,0);
10
11\color{black}
12Black text
13\tikz \draw[fill=yellow] (0,0) -- node[draw, fill] {x} (1,0) -- node[draw=cyan] {y}
(2,0);
14\tikz \draw[fill=yellow] (0,0) to[R] (2,0) to[R,color=cyan] (4,0) to [generic] (6,0) to
[fullgeneric] (8,0);

```

⁹⁴version 1.5.0 loaded it unconditionally for LaTeX; please do not use it

CircuitikZ components that are fillable will inherit the `fill` property of the path (it is almost impossible to do otherwise) as if the `fill` flag was present. “Full”-type elements (for example, full diodes or similar) are filled with the draw color; elements with intrinsic labels (i.e., labels that are part of the shape, like signs on amplifiers and pin numbers in chips) are drawn with the “draw” colors.

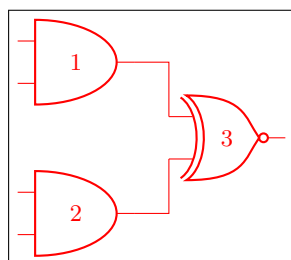
Basically, you should have no problem if:

1. You stick to use styles (see 3.3.2) for filling your components, or using a direct `fill=...` option in the component node or `to` option;
2. do not try to change the color mid-path; sometimes it works (see the examples below), but it’s better to avoid it (using different paths is better);
3. when coloring whole circuits, it’s better to use the option `color=...` in your global picture options or in the `\draw` command (not just the color name as a shorthand);
4. forget about transparency.

Nevertheless, if you really need to do strange things with colors you can read on; you can do almost everything but there are several glitches to take into account.

7.1 Shape colors

The color of the components is stored in the key `\circuitikzbasekey/color`. CircuitikZ tries to follow the color set in TikZ, although sometimes it fails. The following circuit will fail to draw the circuit in red if the patching of the inner commands of TikZ fails, like for example in ConTeXt.

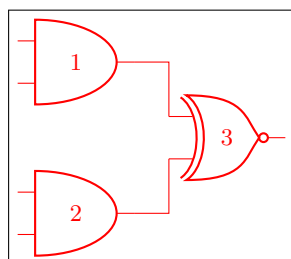


```

1 \begin{circuitikz} \draw[red]
2   (0,2) node[and port](myand1){1}
3   (0,0) node[and port](myand2){2}
4   (2,1) node[xnor port](myxnor){3}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 ;\end{circuitikz}

```

If you see this problem, please do not use just the color name as a style, like `[red]`, but rather assign the style `[color=red]`.

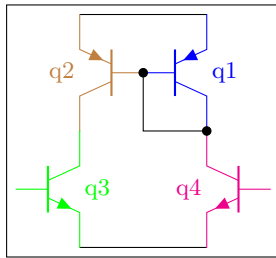


```

1 \begin{circuitikz} \draw[color=red]
2   (0,2) node[and port](myand1){1}
3   (0,0) node[and port](myand2){2}
4   (2,1) node[xnor port](myxnor){3}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 ;\end{circuitikz}

```

One can, of course, change the color directly in the component:

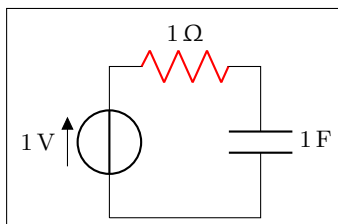


```

1 \begin{circuitikz} \draw
2   (0,0) node[pnp, color=blue](pnp2){q1}
3   (pnp2.B) node[pnp, xscale=-1, anchor=B, color=brown](pnp1){\ctikzflipx{q2}}
4   (pnp1.C) node[npn, anchor=C, color=green](npn1){q3}
5   (pnp2.C) node[npn, xscale=-1, anchor=C, color=magenta](npn2){\ctikzflipx{q4}}
6   (pnp1.E) -- (pnp2.E) (npn1.E) -- (npn2.E)
7   (pnp1.B) node[circ]{} |- (pnp2.C) node[circ]{}
8;\end{circuitikz}

```

The all-in-one stream of bipoles poses some challenges, as only the actual body of the bipole, and not the connecting lines, will be rendered in the specified color (Notice that the following example uses the special **siunitx** shorthand; you can use it only in simple cases like here, in general using the full `\SI{}` or `\qty{}` commands).

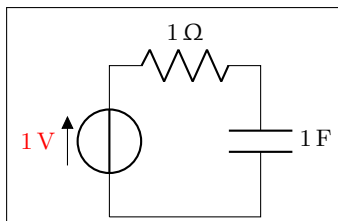


```

1 \begin{circuitikz} \draw
2   (0,0) to[V=1<\volt>] (0,2)
3       to[R=1<\ohm>, color=red] (2,2)
4       to[C=1<\farad>] (2,0) -- (0,0)
5;\end{circuitikz}

```

The postponed application of colors creates a problem if you want to use arrows for voltages, because the “arrows” are partially part of the path, partially nodes:

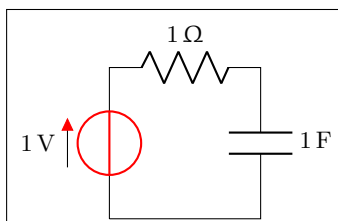


```

1 \begin{circuitikz} \draw
2   (0,0){[red] to[V=1<\volt>] (0,2) }
3       to[R=1<\ohm>] (2,2)
4       to[C=1<\farad>] (2,0) -- (0,0)
5;\end{circuitikz}

```

...and that can become quite frustrating:

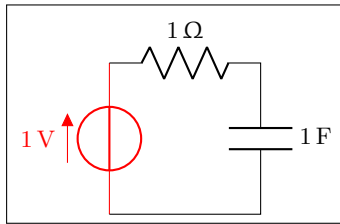


```

1 \begin{circuitikz} \draw
2   (0,0) to[V=1<\volt>, color=red] (0,2)
3       to[R=1<\ohm>] (2,2)
4       to[C=1<\farad>] (2,0) -- (0,0)
5;\end{circuitikz}

```

In those cases, the only way out is to specify different paths (and/or using advanced voltages, see [5.8](#))



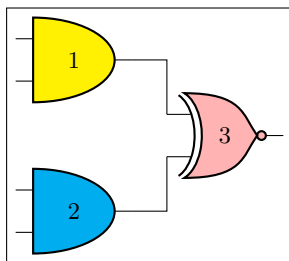
```

1 \begin{circuitikz} \draw[color=red]
2   (0,0) to[V=1<\volt>, color=red] (0,2);
3   \draw (0,2) to[R=1<\ohm>] (2,2)
4         to[C=1<\farad>] (2,0) -- (0,0)
5 ;\end{circuitikz}

```

7.2 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:

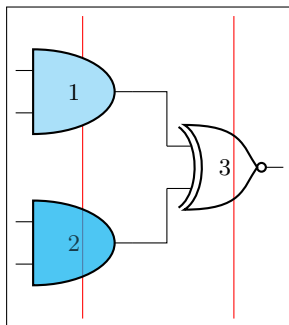


```

1 \begin{circuitikz} \draw
2   (0,2) node[and port, fill=yellow](myand1){1}
3   (0,0) node[and port, fill=cyan](myand2){2}
4   (2,1) node[xnor port,fill=red!30!white](myxnor){3}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 ;\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 should be an exceptional thing anyway. Notice that in simple case `fill opacity` works, but don't count too much on it.

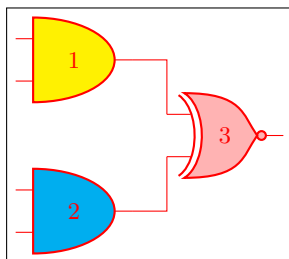


```

1 \begin{circuitikz}
2   \ctikzset{logic ports/fill=cyan!30!white}
3   \draw[red] (-0.5,3) -- (-0.5, -1);
4   \draw[red] (1.5,3) -- (1.5, -1);
5   \draw
6   (0,2) node[and port](myand1){1}
7   (0,0) node[and port, fill=cyan, fill opacity=0.7](myand2)
8         {2}
9   (2,1) node[xnor port, circuitikz/logic ports/fill=none
10          ](myxnor){3}
11   (myand1.out) -| (myxnor.in 1)
12   (myand2.out) -| (myxnor.in 2)
13 ;\end{circuitikz}

```

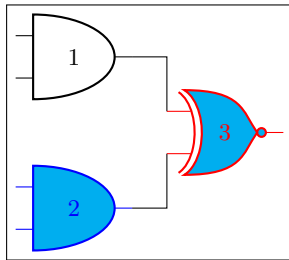
You can combine shape colors with fill colors, too, but you should use the explicit `color` option style for this:



```

1 \begin{circuitikz} \draw[color=red]
2   (0,2) node[and port, fill=yellow](myand1) {1}
3   (0,0) node[and port, fill=cyan] (myand2){2}
4   (2,1) node[xnor port,fill=red!30!white](myxnor){3}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 ;\end{circuitikz}

```



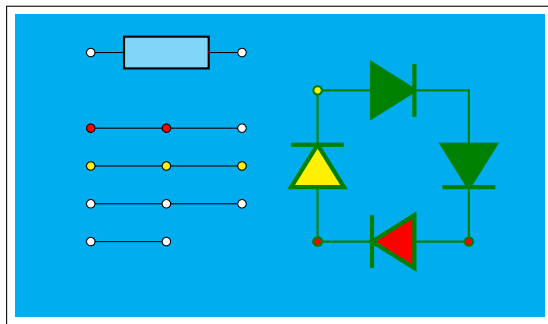
```

1 \begin{circuitikz} \draw
2   (0,2) node[and port, color=black] (myand1){1}
3   (0,0) node[and port, color=blue, fill=cyan] (myand2){2}
4   (2,1) node[xnor port, color=red, fill=cyan] (myxnor){3}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 ;\end{circuitikz}

```

7.2.1 Background colors different from white

Notice also that the connection point is always filled, and the color *tries* to follow the color of the filling of the component (but look at section 6.1.1). Moreover, if you want to pass fill transparency down to path-style components, you *have* to put it into the options of the `\draw` command.



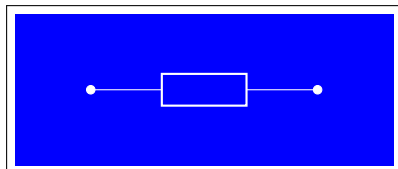
```

1 \begin{circuitikz}
2   \fill[cyan] (0,3.0) rectangle (7,7);
3   \draw [fill opacity=0.5] (1,6.5) to[generic, fill=white,o-o] ++(2,0);
4   \draw (1,5.5) to[short, fill=red, o-o] ++(1,0) to[short, -o] ++(1,0);
5   \draw[fill=yellow] (1,5) to[short, o-o] ++(1,0) to[short, -o] ++(1,0);
6   \draw (1,4.5) to[short, o-o] ++(1,0) to[short, -o] ++(1,0);
7   \draw (1,4) node[ocirc]{} -- ++(1,0) node[ocirc]{};
8   \draw [thick, color=green!50!black] (4,4) to [D,o-o,fill=yellow] ++(0,2) to[D*, fill
9     =yellow] ++(2,0) to[D*,fill=yellow] ++(0,-2) to[D, fill=red, o-o] ++(-2,0);
10 \end{circuitikz}

```

As you can see, the “black” components (as `D*`) follow the color of the line, not the fill.

Note, however, that if you choose a colored background, for example with the `\pagecolor{}` command or with other tricks, the nodes will be by default still filled with white.

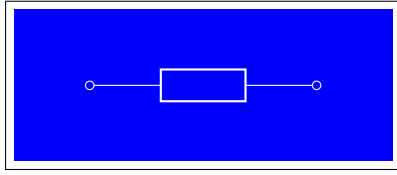


```

1 \begin{circuitikz}[european]
2   \fill[color=blue] (-1,-1) rectangle (4,1);
3   \draw[color=white] (0,0) to[R, o-o] ++(3,0);
4 \end{circuitikz}

```

You have two solutions for this. You can redefine the `o-o` (and the similar commands `-o`, `o-`, `*-o` and so on) with a blue filled “open” pole:

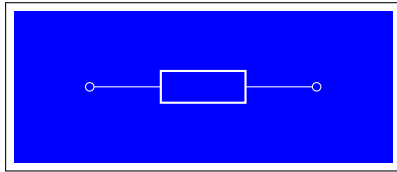


```

1\tikzset{bcirc/.style={shape=ocirc, fill=blue}}
2\ctikzset{o-o/.style ={
3    \circuitikzbasekey/bipole/nodes/left=bcirc,
4    \circuitikzbasekey/bipole/nodes/right=bcirc}}
5\begin{circuitikz}[european]
6    \fill[color=blue] (-1,-1) rectangle (4,1);
7    \draw[color=white] (0,0) to[R, o-o] ++(3,0);
8\end{circuitikz}

```

Also, since v1.2.3, you can set the key `open poles fill` (default: white which works for `ocirc`, `odiamondpole` and `osquarepole`):



```

1\begin{circuitikz}[european]
2    \tikzset{open poles fill=blue}
3    \fill[color=blue] (-1,-1) rectangle (4,1);
4    \draw[color=white] (0,0) to[R, o-o] ++(3,0);
5\end{circuitikz}

```

8 FAQ: Frequently asked questions

8.1 Using named nodes in circuits

Q: When I use a node to name a connection in the circuit, I have gaps in the wires! I am sure it used to work!

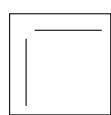
A: This is explained in 1.9. The fast answer is that in a hurry, use the 1.1.2 fallback point with:

```
\usepackage{circuitikz-1.1.2}
```

in your preamble.

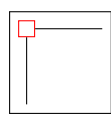
But really, your circuit definition is buggy, so the best thing to do is fix that; if you want to name a point in your circuit, you should use a **coordinate**, not a **node**.⁹⁵ Here is a small tutorial on *why* you should change your circuit.

Nodes, in TikZ, normally have a non-zero size even when they are empty; moreover, connections are supposed to join the border of nodes. Please study the following (pure TikZ, not CircuiTikZ):



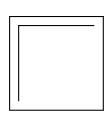
```
1 \begin{tikzpicture}
2   \path (1,1) node (A){}; % empty node at (1,1)
3   \draw (1,0) -- (A) -- (2,1); % surprise!
4 \end{tikzpicture}
```

The gap is there because the node has a non-zero size (more in detail, its **inner sep** is by default different from zero). You can see it easily if you draw the node shape:



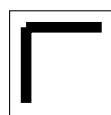
```
1 \begin{tikzpicture}
2   \path (1,1) node [draw=red] (A){};
3   \draw (1,0) -- (A) -- (2,1);
4 \end{tikzpicture}
```

The problem is that when you want to name a coordinate, in the sense of a dimensionless point, you should use a **coordinate**, **not** a **node**!



```
1 \begin{tikzpicture}
2   \path (1,1) coordinate (A); % give a name to (1,1)
3   \draw (1,0) -- (A) -- (2,1); % now it's ok!
4 \end{tikzpicture}
```

Now, before version 1.2.1 (and since around 0.6), CircuiTikZ was detecting when a connection was between nodes and sort-of added a **node.center** movement to the path. That in turn generated the need of hacks to draw the correct joining of lines, because that kind of movement broke the continuity of the path, like in this example:



```
1 \begin{tikzpicture}[line width=4pt]
2   \path (1,1) node (A){};
3   \draw (1,0) -- (A.center) (A) (A.center) -- (2,1);
4 \end{tikzpicture}
```

You can see more examples and more reasonings on GitHub; start from the [issue detecting the join problem](#), then [look at the merged fix](#); you can follow several issues and discussions from there, but for example there are circuits that can't be drawn with the “hack” in, [like this one](#).

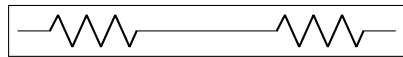
So finally it was decided⁹⁶ to remove the change, to simplify the code and to make the package more maintainable.

⁹⁵Yes, I understand from where the confusion arise — in circuit theory they are called nodes.

⁹⁶well, Romano decided, so you can blame him. *I do not think that workarounds to correct malformed circuits are really maintainable; just see the bunch of code removed by the patch!* — Romano.

8.2 Using dashed (or colored) wires in circuits

Q: How can I make part of the wires dashed (or colored)? This does not work:



```
1 \begin{circuitikz}
2   \draw (0,0) to[R] ++(2,0)
3     to[short, dashed, red] ++(1,0)
4     to [R] ++(2,0); % surprise!
5 \end{circuitikz}
```

Nor this one, which is even stranger:



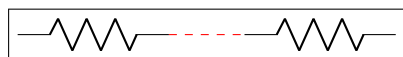
```
1 \begin{circuitikz}
2   \draw (0,0) to[R] ++(2,0)
3     [dashed, red] -- ++(1,0)
4     to [R] ++(2,0); % surprise!
5 \end{circuitikz}
```

A: This is an effect on how TikZ builds and draws path. As explained in the TikZ manual,⁹⁷ most path options are globally valid for the whole path; color and dash/dot is one of this. You have two options in this case. The first one is to use two paths.



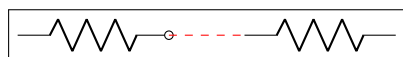
```
1 \begin{circuitikz}
2   \draw (0,0) to[R] ++(2,0) coordinate(a);
3   \draw [dashed, red] (a) -- ++(1,0) coordinate(b);
4   \draw (b) to [R] ++(2,0);
5 \end{circuitikz}
```

The other one is to use **edge** operations⁹⁸; be sure to read about it on the TikZ manual⁹⁹ — but basically this is similar to the **to** operation but it builds another path (added at the end of the current path, like nodes are). This means that it can use different options, and that it **does not** move the path coordinates. So, for example:



```
1 \begin{circuitikz}
2   \draw (0,0) to[R] ++(2,0)
3     edge[dashed, red] ++(1,0)
4     % we have to move the path position here!
5     ++(1,0) to [R] ++(2,0);
6 \end{circuitikz}
```

The only problem with this approach is that the **edges** are added *after* the nodes, so it can create problems with nodes (look carefully!):



```
1 \begin{circuitikz}
2   \draw (0,0) to[R,-o] ++(2,0)
3     edge[dashed, red] ++(1,0)
4     ++(1,0) to [R] ++(2,0);
5 \end{circuitikz}
```

So it's better, in this case, to add the nodes manually after the path (there is no perfect solution!):

⁹⁷in 3.1.5b, section 14, “syntax for path specification”

⁹⁸I took the idea from [this answer by @LaTeXdraw-com user on TeX.SE](#), thanks!

⁹⁹in 3.1.5b, section 17.12, “connecting nodes: use the **edge** operation”

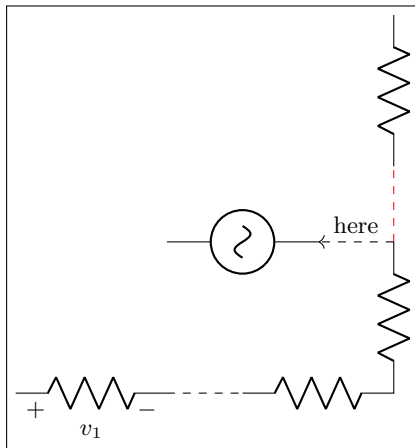


```

1 \begin{circuitikz}
2   \draw (0,0) to[R] ++(2,0) coordinate(a)
3   edge[dashed, red] ++(1,0)
4   ++(1,0) to [R] ++(2,0);
5   \node [ocirc] at (a){};
6 \end{circuitikz}

```

A more complex example can be seen (look at the comments!) in the following circuit.



```

1 \begin{circuitikz}[american]
2   \draw (0,0) to[R, v=$v_1$] ++(2,0)
3   edge[dashed] ++(1,0)
4   ++(1,0) to[R]
5   ++(2,0) to [R] ++(0,2) coordinate(a)
6   edge[red, dashed] ++(0,1)
7   % several edges start from the same position
8   edge[dashed, ->] node[above]{here} ++(-1,0)
9   % notice that the path here is still
10  % at coordinate (a)!
11  ++(0,1) to[R] ++(0,2)
12  (a) ++(-1,0) to[sV] ++(-2,0);
13 \end{circuitikz}

```

8.3 Errors when externalizing pictures

Q: When using `\tikzexternalize` I get the following error:

! Emergency stop.

A: The TikZ manual states:

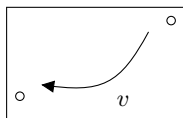
Furthermore, the library assumes that all L^AT_EX pictures are ended with `\end{tikzpicture}`.

Just substitute every occurrence of the environment `circuitikz` with `tikzpicture`. They are actually pretty much the same.

8.4 Labels, voltages and currents woes

Q: How do I draw the voltage between two nodes?

A: Between any two nodes there is an open circuit!



```

1 \begin{circuitikz} \draw
2   node[ocirc] (A) at (0,0) {}
3   node[ocirc] (B) at (2,1) {}
4   (A) to[open, v=$v$] (B)
5 ;\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, joined with a suboptimal processing by `CircuitikZ` (up to 1.2.7) of the passing of the argument of keys.

You should protect commas and equal signs like in `to[R = $\{\$R_1=12V\$ \}$]` or `to[ospst = $\{open, 3s\}$]`.

In versions up to 1.2.7, use for example `\mbox{}` or define `\def{\eq}{=}` and use `to[R = $\$R_1\eq 12V\$$]`, or try to protect commas and equal signs like `to[ospst = open{,} 3s]` or `ospst=\mbox{open, 3s}` instead; see caveat in section 5.1.

8.5 Global scaling and rotating

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.

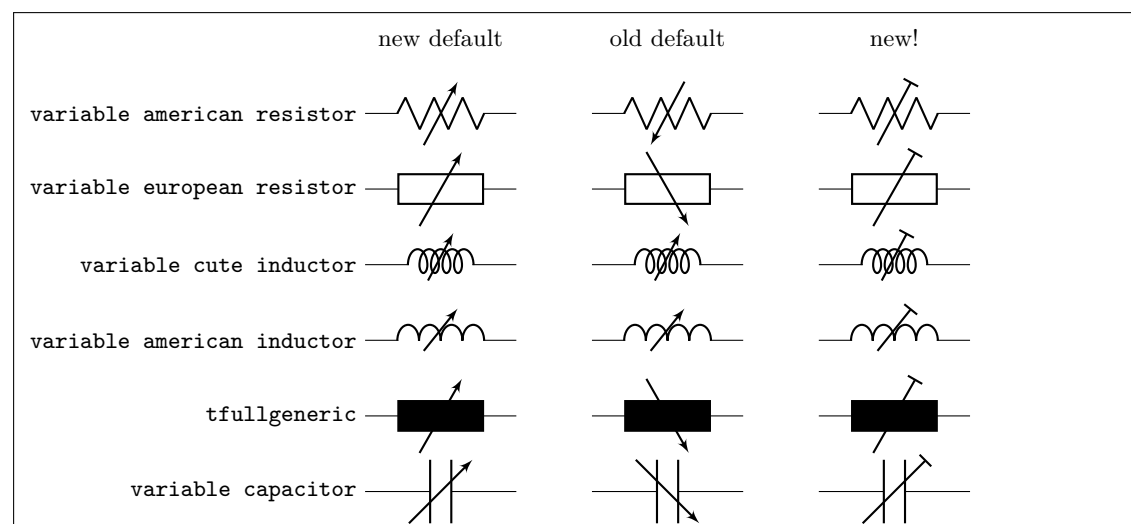
A: Yes, it's a known bug (or misfeature, or limitation, or a fact of life). See section 1.7. `CircuitikZ` is not compatible with `pics` at this point.

8.6 Tunable components

Q: The direction of the arrows in variable resistors or capacitors changed!

A: Yes, it changed in v1.3.3.

Version 1.3.3 fixes the direction of the arrows in tunable elements; before this version, they were more or less random, now the arrow goes from bottom left to top right. You have the option to go back to the old behavior with `\ctikzset{bipoles/fix tunable direction=false}`. As a compensation for the fuss, now the arrows are configurable.



```

1 \begin{circuitikz}[european]
2   \draw (1,0) node{new default} (4,0) node{old default} (7,0) node{new!};
3   \foreach [count=\i] \comp in
4     {variable american resistor, variable european resistor,
5       variable cute inductor, variable american inductor, tfullgeneric,
6       variable capacitor} {
7     \draw (0,-\i) node[left]{\texttt{\comp}} to[\comp, name=E] ++(2,0);
8     \ctikzset{bipoles/fix tunable direction=false}
9     \draw (3,-\i) to[\comp, name=E] ++(2,0);
10    \ctikzset{bipoles/fix tunable direction=true, tunable end arrow={Bar}}
11    \draw (6,-\i) to[\comp, name=E] ++(2,0);
12  }
13 \end{circuitikz}

```


9 Defining new components

Per me si va ne la città dolente,
per me si va ne l'eterno dolore,
per me si va tra la perduta gente.

...

Lasciate ogne speranza, voi ch'intrate.¹⁰⁰

Big fat warning: this material is reserved for T_EX-hackers; do not delve into this if you have no familiarity with (at least) a bit of core T_EX programming and to the basic TikZ layer. You have been warned.

9.1 Suggested setup

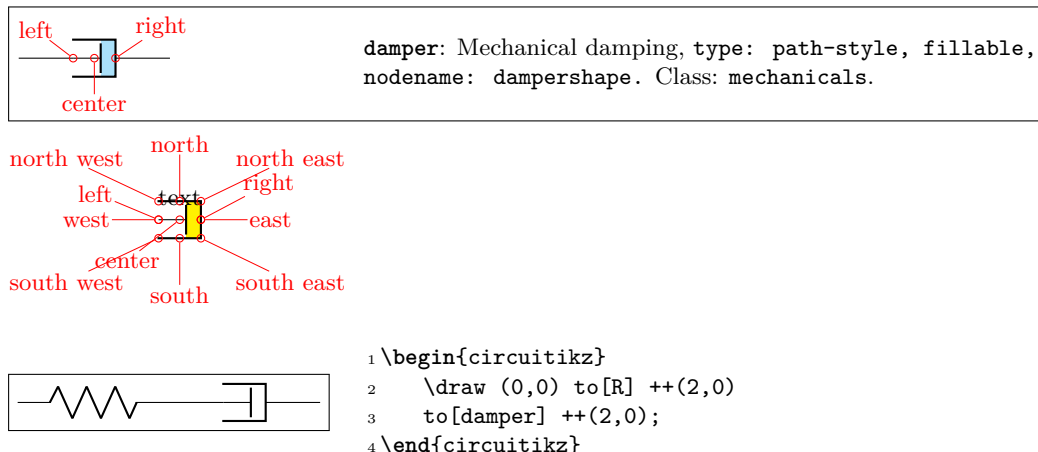
Notice: the source code has been reorganized after release 1.2.7; if you are bound to use an older version check the corresponding manual.

The suggested way to start working on a new component is to use the utilities of the CircuiTikZ manual for checking and testing your device. Basically, find (or download) the source code of the last version of CircuiTikZ and find the file `ctikzmanutils.sty`; copy it in your directory and prepare a file like this:

```
1 \documentclass[a4paper, titlepage]{article}
2 \usepackage{a4wide} %smaller borders
3 \usepackage[utf8]{inputenc} %not needed since LaTeX 2019
4 \usepackage[T1]{fontenc}
5 \parindent=0pt
6 \parskip=4pt plus 6pt minus 2pt
7 \usepackage[siunitx, RPvoltages]{circuitikzgit}
8 \usepackage{ctikzmanutils}
9 \makeatletter
10 %% Test things here
11 % defines
12
13 % components
14
15 % paths
16 \makeatother
17
18 \begin{document}
19
20 \circuitdescbip*{damper}{Mechanical damping}{}(left/135/0.2, right/45/0.2, center
    /-90/0.3)
21
22 \geolrcoord{dampershape, fill=yellow}
23
24 \begin{LTXexample}[varwidth]
25 \begin{circuitikz}
26     \draw (0,0) to[R] ++(2,0)
27         to[damper] ++(2,0);
28 \end{circuitikz}
29 \end{LTXexample}
30 \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):

¹⁰⁰<https://classicsincontext.wordpress.com/2010/02/28/canto-iii-per-me-si-va-ne-la-citta-dolente/>



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.

9.2 Path-style component

Let's define for example a path style component, like the one suggested by the user @alex on [T_EX stack-exchange site](#). The component will be a mix of the `damper` and the `spring` components already present.

The definitions of the components are in the files `pgfcircsomething.tex`; they are more or less distributed by the number of terminals, but there are exceptions (for example, switches are in `bipoles`, even if several of them are tripoles or more...`grep` is your friend here).

To define the new component we will look into (in this case) `pgfcircbipoles.tex`; at the start of the block where the components are defined, you can find the relevant definitions (sometime some of the definitions are in `pgfcirc.defines.tex`, for historical or dependencies reasons). 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 into the we find

```

1 \ctikzset{bipoles/spring/height/.initial=.5}
2 \ctikzset{bipoles/spring/width/.initial=.5}
3 \ctikzset{bipoles/damper/height/.initial=.35}
4 \ctikzset{bipoles/damper/length/.initial=.3}
5 \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).

After that we will copy, for example, the definition of the damper into our code, just changing the name:

```

1 %% mechanical resistor - damper
2 \pgfcircdeclarebipolescaled{mechanicals}
3 {} % extra anchors
4 {\ctikzvalof{bipoles/damper/height}} % depth (under the path line)
5 {viscoe} % name
6 {\ctikzvalof{bipoles/damper/height}} % height (above the path line)
7 {\ctikzvalof{bipoles/damper/width}} % width
8 {
9   \pgfpathrectanglecorners{\pgfpoint{\ctikzvalof{bipoles/damper/length}}\
    pgf@circ@res@right}{\pgfpoint{\pgf@circ@res@down}}{\pgfpoint{\pgf@circ@res@right}}{\
    pgf@circ@res@up}}

```

```

10 \pgfcirc@maybefill
11
12 % line into the damper
13 \pgfpathmoveto{\pgfpoint{\pgfcirc@res@left}{\pgfcirc@res@zero}}
14 \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\pgfcirc@res@right}
15 {\pgfcirc@res@zero}}
16 \pgfusepath{stroke}
17
18 % damper box
19 \pgfcirc@setlinewidth{bipoles}{\pgfstartlinewidth}
20 \pgfpathmoveto{\pgfpoint{\pgfcirc@res@left}{\pgfcirc@res@down}}
21 \pgfpathlineto{\pgfpoint{\pgfcirc@res@right}{\pgfcirc@res@down}}
22 \pgfpathlineto{\pgfpoint{\pgfcirc@res@right}{\pgfcirc@res@up}}
23 \pgfpathlineto{\pgfpoint{\pgfcirc@res@left}{\pgfcirc@res@up}}
24
25 \pgfsetrectcap
26 \pgfsetmiterjoin
27 \pgfusepath{stroke}
28
29 % damper vertical element
30 \pgfpathmoveto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\pgfcirc@res@right}
31 {.8\pgfcirc@res@down}}
32 \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\pgfcirc@res@right}
33 {.8\pgfcirc@res@up}}
34 \pgfsetbuttcap
35 \pgfusepath{stroke}
36
37 }

```

This `\pgfcircdeclarebipolescaled` 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: in this case we are reusing the ones of the `damper` shape. Moreover, the element is assigned to the class `mechanicals` for styling.

To be coherent with the styling, you should use (when needed) the length `\pgfcirc@scaled@Rlen` as the “basic” length for drawing, using the fill functions (they are defined at the start of the file `pgfcirc.defines.tex`) to fill and stroke — so that the operation will follow the style parameters and, finally, use the macro `\pgfcirc@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 `\pgfcirc@res@left`) are already scaled for your use. You can use these lengths (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 (the definition of the commands are normally in `pgfcircpath.tex`). In this case the component is simple — look at the definitions if you need to do more complex things.

```

1 \pgfcirc@activate@bipole@simple{1}{viscoe}

```

In the definition above, the `{1}` parameter means that using the component like `to[viscoe=A]` will be equivalent to `to[viscoe, l=A]`; you can also use `v` or `i` or `f` if your component needs it. Now you can show it with:

```

1 \circuitdescbip*{viscoe}{Mechanical viscoelastic element}{}(left/135/0.2, right/45/0.2,
2 center/-90/0.3)
3
4 \geolrcoord{viscoeshape, fill=yellow}
5 \begin{LTXexample}[varwidth]
6 \begin{circuitikz}
7 \draw (0,0) to[spring] ++(2,0)

```

```

8   to[viscoe] ++(2,0);
9 \end{circuitikz}
10 \end{LTXexample}

```

Obviously, at first 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 in the main argument of the `\pgfcircdeclarebipole`, things will be set up so that the lengths `\pgf@circ@res@right` and `\pgf@circ@res@up` are the x - y coordinates of the upper right corner, and `\pgf@circ@res@left` and `\pgf@circ@res@down` are the x - y coordinates of the lower left corner of your shape. The `center` coordinate is usually at $(0pt, 0pt)$.

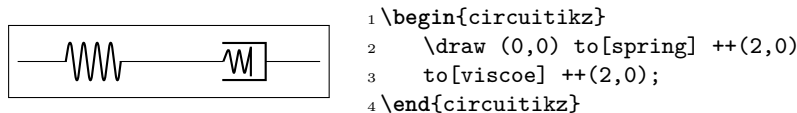
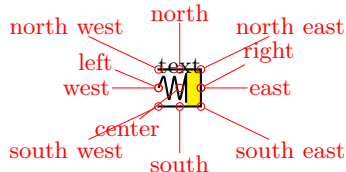
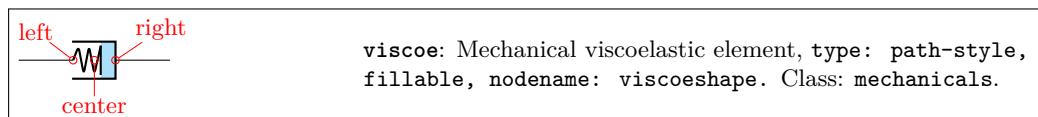
Looking at the implementation of the `spring` element, one possibility is changing the lines between lines 12 and 16 with:

```

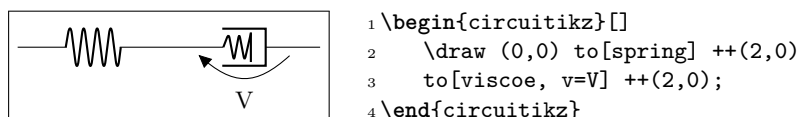
1   % spring into the damper
2   \pgfscope
3     \pgfpathmoveto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@zero}}
4     \pgf@circ@setlinewidth{bipoles}{\pgfstartlinewidth}
5     \pgfsetcornersarced{\pgfpoint{.25\pgf@circ@res@up}{.25\pgf@circ@res@up}}
6     \pgfpathlineto{\pgfpoint{.75\pgf@circ@res@left}{.75\pgf@circ@res@up}}
7     \pgfpathlineto{\pgfpoint{.5\pgf@circ@res@left}{-.75\pgf@circ@res@up}}
8     \pgfpathlineto{\pgfpoint{.25\pgf@circ@res@left}{.75\pgf@circ@res@up}}
9     \pgfpathlineto{\pgfpoint{0pt}{-.75\pgf@circ@res@up}}
10    \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\pgf@circ@res@right}
11                    }{.75\pgf@circ@res@up}}
12    \pgfusepath{stroke}
13  \endpgfscope

```

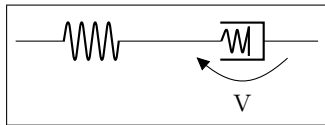
which leads to:



Now you can check if the voltage labels are correct for your new component:



If you think they are too tight or too loose, you can use a (developer-only) key to adjust the distance:



```

1 \begin{circuitikz}
2   \ctikzset{bipoles/viscoe/voltage/additional shift/.
      initial=1}
3   \draw (0,0) to[spring] ++(2,0)
4     to[viscoe, v=V] ++(2,0);
5 \end{circuitikz}

```

Notice that by default the key `bipoles/mybipole/voltage/additional shift` is not defined, so if you want to use it you must create it before (this is the meaning of the `.initial` here).

Now you can save all the code between the `\makeatletter` and `\makeatother` in a file and `\input{}` it for using your special component, or submit the component to the project (see below).

As a final note, notice that the `viscoe` element is already added to the standard package.

9.3 Node-style component

Adding a node-style component is much more straightforward. Just define it by following examples in, for example, `pgfcircctripoles.tex` or the other files; be careful that you should define all the geographical anchors of the shape if you want that the TikZ 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

```

1   \savedmacro{\ctikzclass}{\edef\ctikzclass{transistors}}
2   \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):

```

1   \savedanchor\northeast{% upper right
2     \pgfmathsetlength{\pgf@circ@scaled@Rlen}{\ctikzvalof{\ctikzclass/scale}\pgf@circ@Rlen}
3     \pgf@y=\pgf@circ@scaled@Rlen
4     \pgf@y=0.5\pgf@y
5     \pgf@x=0.3\pgf@y
6   }

```

in all the `savedanchors`.

Then, to draw your component, you should start with¹⁰¹:

```

1   \pgf@circ@draw@component{%
2     \pgf@circ@scaled@Rlen=\scaledRlen
3     ...
4   }

```

and then use `\pgf@circ@scaled@Rlen` (or the anchors) as the default length while you draw it.

The special command `\pgf@circ@draw@component` will issue a `\behindforegroundpath` command, and take care of calling the start and end hooks for the component. Notice that, given the use of `\behindforegroundpath`, you must take care to use the path you define here! The path itself is protected by a `pgfscope` (and so also by a `TEX` group), so local definitions will be reset after exiting.

¹⁰¹Since v1.5.0; component defined with this mechanism will not be compatible with older CircuitikZ.

9.3.1 The internal hook system

Since version v1.5.0, before starting the drawing of any component, CircuiTikZ will check for the existence of three different hooks, in the following order (suppose that the shape name is *myname*, and it has a class *myclass*):

- `\ctikz@hook@start@draw@component@myname`
- `\ctikz@hook@start@draw@class@myclass`
- `\ctikz@hook@start@draw@default`

The first one that is defined in the current (or outer) scope is used, and the following ones are not used. These hooks can be used to set drawing parameters, or to reset them to a known state: TikZ normally inherit most of the drawing option, but that can lead to surprises (like unexpected arrows, etc.).

In the same way, before leaving `\pgf@circ@draw@component`, a set of similar hooks (with `end` instead of `start`) is tried, with the same logic.

The only predefined hook is `\ctikz@hook@start@draw@default`, which is set to the equivalent of:

```
1 \pgfsetshortenstart{+0pt}\pgfsetshortenend{+0pt}\pgfsetarrows{-}%  
2 \def\pgf@circ@reset@rounded{\pgfsetcornersarced{\pgfpointorigin}}%
```

which means that, by default, arrows parameters are reset to the default (no shorten, no arrows) and that corners are not rounded. If you want to override them, just define the appropriate hook for your component/class and the generic one will not be called.

No `...@end@draw@...` hook is defined by default.

9.3.2 Finishing your work

Once you have a satisfactory element, you should

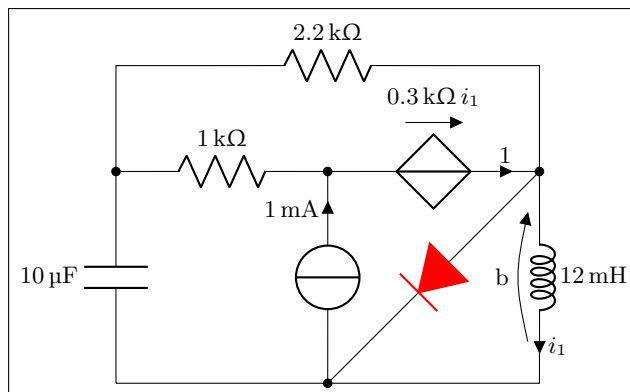
- Clean up your code;
- write a piece of documentation explaining its use, with an example;
- Propose the element for inclusion in the GitHub page of the project (you will have to license this as explained in that page, of course).

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.

10 Examples

Here a series of examples, contributed by several people, is shown with their code.

10.1 A red diode

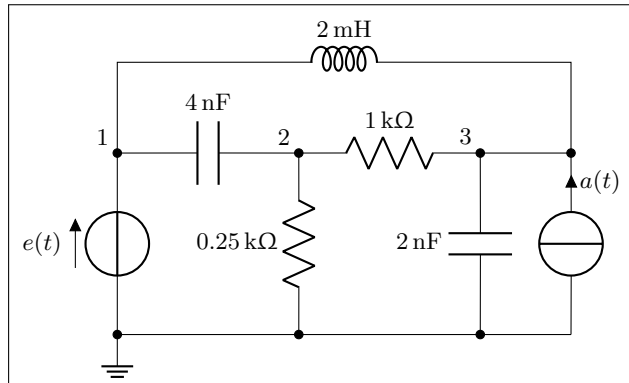


```

1 \begin{circuitikz}[scale=1.4]\draw
2 (0,0) to[C, l=10<\micro\farad>] (0,2) -- (0,3)
3     to[R, l=2.2<\kilo\ohm>] (4,3) -- (4,2)
4     to[L, l=12<\milli\henry>, i=$i_1$,v=b] (4,0) -- (0,0)
5 (4,2) to[D*, color=red] (2,0)
6 (0,2) to[R, l=1<\kilo\ohm>, *-] (2,2)
7     to[cV, i=1, -*-, v=$\SI{.3}{\kilo\ohm}\,, i_1$] (4,2)
8 (2,0) to[I, i=1<\milli\ampere>, -*-] (2,2)
9 \end{circuitikz}

```

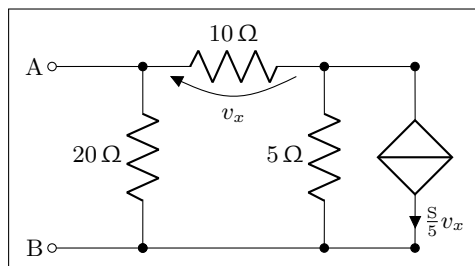
10.2 Using the (experimental) siunitx syntax



```

1 \begin{circuitikz}[scale=1.2]\draw
2   (0,0) node[ground] {}
3     to[V=$e(t)$, *-] (0,2) to[C=4<\nano\farad>] (2,2)
4     to[R, l_=.25<\kilo\ohm>, *-] (2,0)
5   (2,2) to[R=1<\kilo\ohm>] (4,2)
6     to[C, l_=2<\nano\farad>, *-] (4,0)
7   (5,0) to[I, i_=$a(t)$, *-] (5,2) -- (4,2)
8   (0,0) -- (5,0)
9   (0,2) -- (0,3) to[L, l=2<\milli\henry>] (5,3) -- (5,2)
10
11 {[anchor=south east] (0,2) node {1} (2,2) node {2} (4,2) node {3}}
12 ;
13 \end{circuitikz}

```

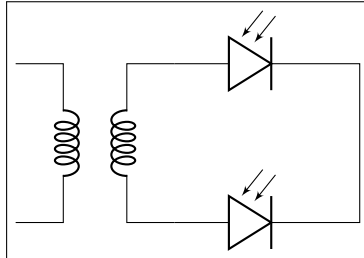


```

1 \begin{circuitikz}[scale=1.2]\draw
2   (0,0) node[anchor=east] {B}
3     to[short, o-*] (1,0)
4     to[R=20<\ohm>, *-] (1,2)
5     to[R=10<\ohm>, v=$v_x$] (3,2) -- (4,2)
6     to[cI=$\frac{\si{siemens}}{5} v_x$, *-] (4,0) -- (3,0)
7     to[R=5<\ohm>, *-] (3,2)
8   (3,0) -- (1,0)
9   (1,2) to[short, -o] (0,2) node[anchor=east] {A}
10 ;\end{circuitikz}

```


10.3 Photodiodes

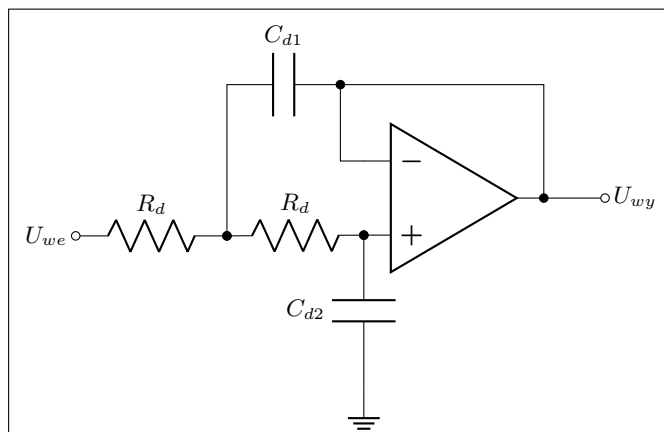


```

1 \begin{circuitikz}[scale=1]\draw
2   (0,0) node[transformer] (T) {}
3   (T.B2) to[pD] ($(T.B2)+(2,0)$) -| (3.5, -1)
4   (T.B1) to[pD] ($(T.B1)+(2,0)$) -| (3.5, -1)
5 ;\end{circuitikz}

```

10.4 A Sallen-Key cell

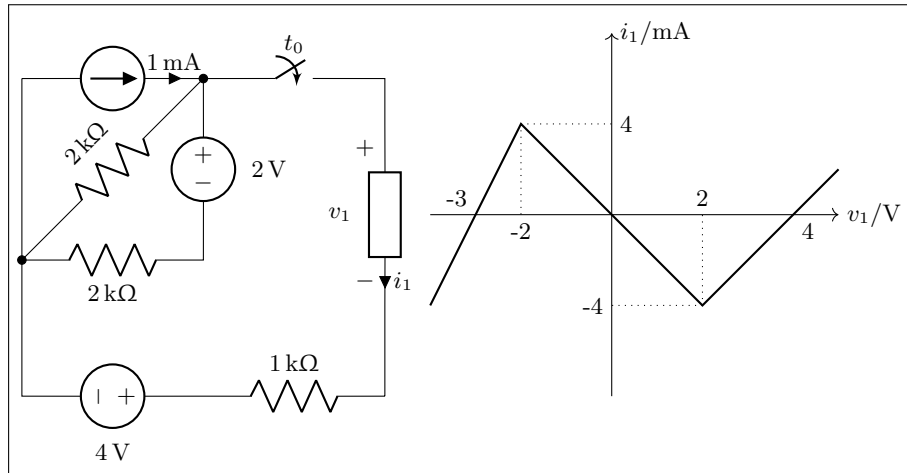


```

1 \begin{circuitikz}[scale=1]\draw
2   (5,.5) node [op amp] (opamp) {}
3   (0,0) node [left] {$U_{we}$} to [R, l=$R_d$, o-] (2,0)
4   to [R, l=$R_d$, *-] (opamp.+)
5   to [C, l=$C_{d2}$, *-] ($(opamp.+) + (0,-2)$) node [ground] {}
6   (opamp.out) |- (3.5,2) to [C, l=$C_{d1}$, *-] (2,2) to [short] (2,0)
7   (opamp.-) -| (3.5,2)
8   (opamp.out) to [short, *-o] (7,.5) node [right] {$U_{wy}$}
9 ;\end{circuitikz}

```

10.5 Mixing circuits and graphs

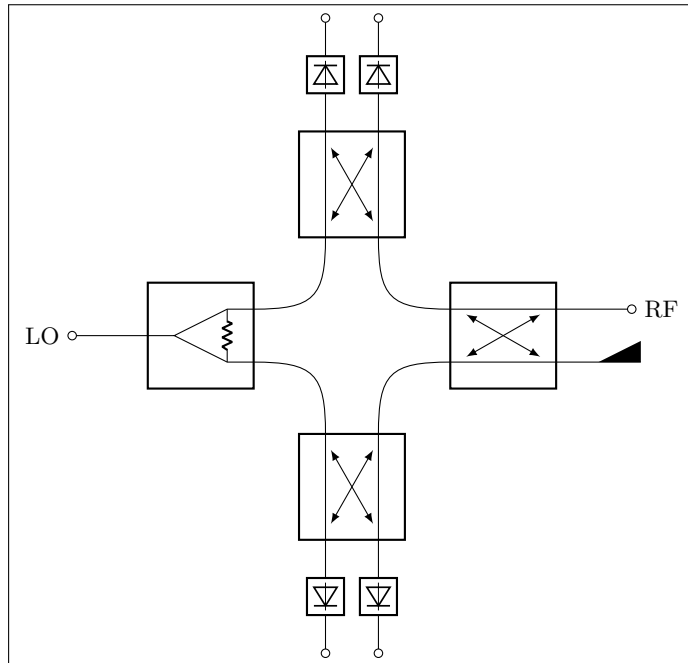


```

1 \begin{circuitikz}[scale=1.2, american]\draw
2   (0,2) to[I=1<\milli\ampere>] (2,2)
3     to[R, l_2=2<\kilo\ohm>, *-] (0,0)
4     to[R, l_2=2<\kilo\ohm>] (2,0)
5     to[V, v_2=2<\volt>] (2,2)
6     to[cspst, l=$t_0$] (4,2) -- (4,1.5)
7     to [generic, i=$i_1$, v=$v_1$] (4,-.5) -- (4,-1.5)
8   (0,2) -- (0,-1.5) to[V, v_4=4<\volt>] (2,-1.5)
9     to [R, l=1<\kilo\ohm>] (4,-1.5);
10
11 \begin{scope}[xshift=6.5cm, yshift=.5cm]
12   \draw [->] (-2,0) -- (2.5,0) node[anchor=west] {$v_1/\text{V}$};
13   \draw [->] (0,-2) -- (0,2) node[anchor=west] {$i_1/\text{mA}$};
14   \draw (-1,0) node[anchor=north] {-2} (1,0) node[anchor=south] {2}
15         (0,1) node[anchor=west] {4} (0,-1) node[anchor=east] {-4}
16         (2,0) node[anchor=north west] {4}
17         (-1.5,0) node[anchor=south east] {-3};
18   \draw [thick] (-2,-1) -- (-1,1) -- (1,-1) -- (2,0) -- (2.5,.5);
19   \draw [dotted] (-1,1) -- (-1,0) (1,-1) -- (1,0)
20             (-1,1) -- (0,1) (1,-1) -- (0,-1);
21 \end{scope}
22 \end{circuitikz}

```

10.6 RF circuit

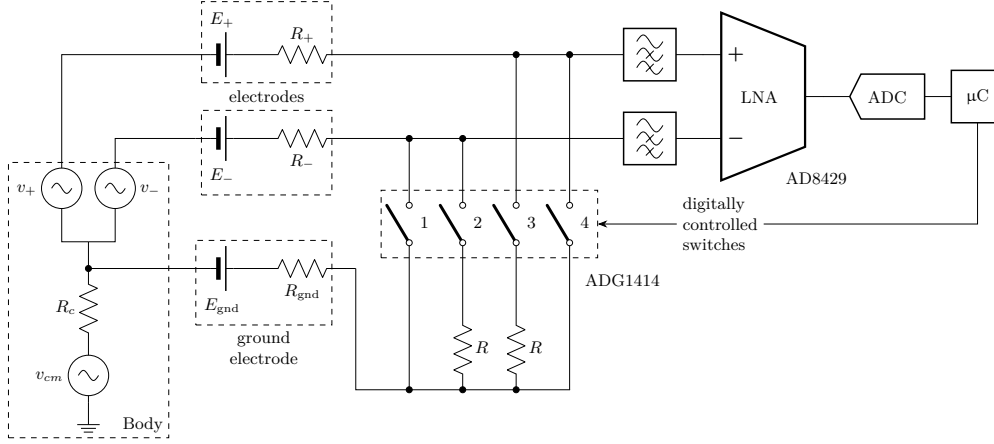


```

1  \begin{circuitikz}[scale=1]
2      \ctikzset{bipoles/detector/width=.35}
3      \ctikzset{quadpoles/coupler/width=1}
4      \ctikzset{quadpoles/coupler/height=1}
5      \ctikzset{tripoles/wilkinson/width=1}
6      \ctikzset{tripoles/wilkinson/height=1}
7      %\draw[help lines,red,thin,dotted] (0,-5) grid (5,5);
8      \draw
9      (-2,0) node[wilkinson](w1){}
10     (2,0) node[coupler] (c1) {}
11     (0,2) node[coupler,rotate=90] (c2) {}
12     (0,-2) node[coupler,rotate=90] (c3) {}
13     (w1.out1) .. controls ++(0.8,0) and ++(0,0.8) .. (c3.port3)
14     (w1.out2) .. controls ++(0.8,0) and ++(0,-0.8) .. (c2.port4)
15     (c1.port1) .. controls ++(-0.8,0) and ++(0,0.8) .. (c3.port2)
16     (c1.port4) .. controls ++(-0.8,0) and ++(0,-0.8) .. (c2.port1)
17     (w1.in) to[short,-o] ++(-1,0)
18     (w1.in) node[left=30] {LO}
19     (c1.port2) node[match,yscale=1] {}
20     (c1.port3) to[short,-o] ++(1,0)
21     (c1.port3) node[right=30] {RF}
22     (c2.port3) to[detector,-o] ++(0,1.5)
23     (c2.port2) to[detector,-o] ++(0,1.5)
24     (c3.port1) to[detector,-o] ++(0,-1.5)
25     (c3.port4) to[detector,-o] ++(0,-1.5)
26     ;
27 \end{circuitikz}

```

10.7 A styled low noise input stage



```

1 \ctikzloadstyle{romano}
2 \scalebox{0.707}{%
3 \begin{circuitikz}[american, romano circuit style]
4   \ctikzset{bipoles/cuteswitch/thickness=0.5}
5   \draw (0,0) node[ground](GND0){} to[sV, l=$v_{cm}$] ++(0,1)
6   to [R, l=$R_c$, -*] ++(0,1.5) coordinate(vcm) --++(0,0.5) coordinate(diffc);
7   \draw (diffc) -| ++(-0.5, 0.5) to[sV, l=$v_+$, name=vplus] ++(0,1) --++(0,2)
8   -- ++(2.5,0) coordinate(skin+ a) to[battery2, l=$E_+$, name=eplus] ++(1,0)
9   to[R=$R_+$, name=rplus] ++(2,0) coordinate(skin+ b) -- ++(0.5,0)
10  -- ++(4,0) coordinate(hpin+) to[highpass] ++(2,0)
11  node[inst amp, anchor=+, noinv input up,
12  circuitikz/amplifiers/scale=1.6,
13  circuitikz/tripoles/inst amp/width=1](LNA){LNA}
14  (LNA.out);
15  \coordinate (skin- a) at (LNA.- -| skin+ a);
16  \draw (diffc) -| ++(0.5,0.5) to[sV, l=$v_-$, name=vminus] ++(0, 1) |- (skin- a);
17  \draw (skin- a) to[battery2, l=$E_-$, name=eminus] ++(1,0)
18  to[R, l=$R_-$, name=rminus] ++(2,0) coordinate(skin- b) -- ++(2.5,0)
19  -- (skin- b -| hpin+) to[highpass] (LNA.-);
20  \coordinate (gnd a) at (vcm -| skin+ a);
21  \draw (vcm) -- (gnd a) to[battery2, l=$E_{\mathrm{gnd}}$, name=egnd] ++(1,0)
22  to[R, l=$R_{\mathrm{gnd}}$, name=rgnd] ++(2,0) coordinate(gnd b);
23  % switch set
24  \def\swdown{-3.2}
25  \draw (skin- b) ++(1,0) coordinate(sw1) to[cosw, invert, mirror, l=1, *-, name=s1]
26  ++(0,\swdown) to[short, -*] ++(0, -1.5);
27  \draw (sw1) ++(1,0) coordinate(sw2) to[cosw, invert, mirror, l=2, *-, name=s2]
28  to[R=$R$, -*] ++(0, -1.5);
29  \draw (sw2|-skin+ b) ++(1,0) coordinate(sw3) to[short, *-, name=s3] (sw3|-sw2) to[cosw,
30  invert, mirror, l=3,] ++(0,\swdown) to[R=$R$, -*] ++(0, -1.5);
31  \draw (sw3) ++(1,0) coordinate(sw4) to[short, *-, name=s4] (sw4|-sw2) to[cosw, invert, mirror,
32  l=4, name=s4] ++(0,\swdown) to[short] ++(0, -1.5) coordinate(endsw);
33  \draw (gnd b) |- (endsw) node[rectjoinfill]{};
34  % boxes
35  \node [rectangle, draw, dashed, fit=(GND0) (vplus) (vpluslabel) (vminuslabel)](body)
36  {};
37  \node [anchor=south east, align=center] at (body.south east) {Body} ;
38  \node [rectangle, draw, dashed, fit=(rplus) (eplus) (epluslabel) (rpluslabel)](top)
39  {};
40  \node [rectangle, draw, dashed, fit=(eminus) (rminus) (eminuslabel) (rminuslabel)](
41  bot){};

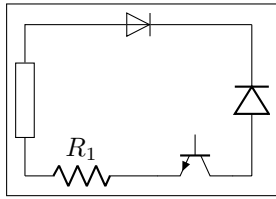
```

```

35 \node [anchor=center, align=center] at ($(top.south)!0.5!(bot.north)$) {electrodes}
    ;
36 \node [rectangle, draw, dashed, fit=(egnd) (rgnd) (egndlabel) (rgndlabel)](gnd){};
37 \node [below, align=center] at (gnd.south) {ground\\ electrode} ;
38 \node [rectangle, draw, dashed, fit=(s1) (s4label), inner ysep=8pt](switches){};
39 % ADC and micro
40 \draw (LNA.out) -- ++(0.5,0) node[msport,circuitikz/RF/scale=2](ADC){ADC};
41 \draw (ADC.right) -- ++(0.5,0) node[twoportshape, anchor=left, t=$\upmu$C](uC){};
42 \draw (uC.south) -- (uC.south |- switches.east) -- ++(-4,0)
43 node[align=left, anchor=east](DCS){\small digitally\\ controlled\\ switches};
44 \draw[-Stealth] (DCS.west) -- (switches.east);
45 % components
46 \node [anchor=north west] at ([xshift=-10pt, yshift=-5pt]switches.south east) {ADG
    1414};
47 \node [anchor=north west] at ([yshift=-5pt]LNA.refv down) {AD8429};
48 \end{circuitikz}
49 } % scalebox

```

10.8 An example with the compatibility option

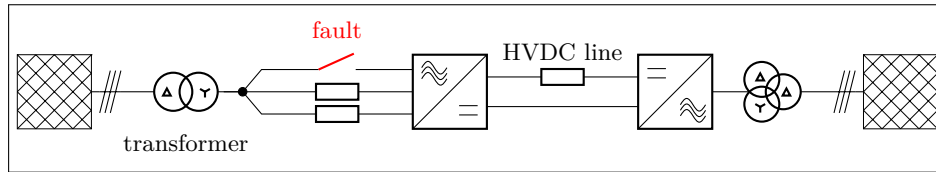


```

1 \documentclass{standalone}
2
3 \usepackage{tikz}
4 \usetikzlibrary{circuits.ee.IEC}
5 \usetikzlibrary{positioning}
6
7 \usepackage[compatibility]{circuitikzgit}
8 \ctikzset{bipoles/length=.9cm}
9
10 \begin{document}
11 \begin{tikzpicture}[circuit ee IEC]
12 \draw (0,0) to [resistor={name=R}] (0,2)
13 to [diode={name=D}] (3,2);
14 \draw (0,0) to [*R=$R_1$] (1.5,0) to [*Tnpn] (3,0)
15 to [*D] (3,2);
16 \end{tikzpicture}
17 \end{document}

```

10.9 3-phases block schematic



```

1 \begin{circuitikz}[smallR/.style={european resistor, resistors/scale=0.5}]
2   \draw (0,0) node[tacdcshape, anchor=ac mid in](acdc){} to[smallR] ++(-2,0)
3     -- coordinate(point) node[circ]({}) ++(-.5,0);
4   \draw (acdc.ac up in)
5     to[nos, invert, mirror, name=switch,color=red] ++(-2,0)
6     -- (point);
7   \draw (acdc.ac down in) to[smallR] ++(-2,0)
8     -- (point)
9     to[oosourcetrans,prim=wye,sec=delta,l=transformer] ++(-1.5,0)
10    to[tmultiwire] ++(-.5,0)
11    node[gridnode, anchor=right]{};
12   \node[above=.3cm,color=red] at (switch) {fault};
13   \draw (acdc.dc up out) to[smallR,l=HVDC line] ++(2,0 )
14     node[tdcacshape, anchor=dc up in](dcac){};
15   \draw (acdc.dc down out) -- (dcac.dc down in);
16   \draw (dcac.right)
17     to[oosource,prim=delta,sec=delta,tert=wye,invert] ++(1.5,0)
18     to[tmultiwire] ++(.5,0) node[gridnode,anchor=left]{};
19 \end{circuitikz}

```

11 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 1.6.3 (unreleased)
 - added documentation on how to contact the border of the source symbols (suggested by [user @Tipounk on GitHub](#))
- Version 1.6.2 (2023-05-13)

Several more styling options for elements (body diodes, transformers, crossing), a clock wedge shape for logical circuits, and documentation updates for ConTeXt, mainly noticing the (upstream) elimination of the thin `siunitx` layer compatibility macros.

 - there is no `siunitx` support for ConTeXt, point to the `units` package
 - `context` compatibility can have glitches: please see [this issue](#)
 - Add styling of `transform core` lines (suggested by [user @myzinsky on GitHub](#))
 - Add `scale` to the bodydiode options (suggested by [user @sputeanus on GitHub](#))
 - Add styling of crossing vertical line (suggested by [user @lkjell on GitHub](#))
 - Add `clockwedge` shape (suggested by [user @Mario1159 on GitHub](#))
- Version 1.6.1 (2023-02-11)

New components: solder jumpers; a couple of small but very useful inversion markers for logical circuits, especially targeted at the mux-demux family; a new inline microphone; a much more versatile hemt; a better legacy `tline`. More tweaks to converters blocks, and a lot of typo/grammar fixes in the manual.

 - Add configurable dashes to the dc symbols in converter blocks (suggested by [user @dbstf on GitHub](#))
 - Add solder jumpers (by Romano)
 - Add a shape to mark european-style inversion (suggested by [user yashpalgoyal1304 on GitHub](#)), adjust European-style logic port triangle inversion symbols to match
 - Add a tail-less mic (suggested by [Dr. Mathhias Jung](#)) and an option to change the thickness of the microphone's bar
 - Enhance the `hemt` shape with a GaN-hemt as example (suggested by [user @epsilon-phi on GitHub](#))
 - Add anchors and a “bare” option to `tline` (suggested by [Dr. Mathhias Jung](#))
 - subcircuits are no more experimental
 - Correction of several typo/grammar errors in the documentation by [quark67](#)
- Version 1.6.0 (2022-12-10)

The big change is the refactoring (and enhancement) of the block's code. In addition, double gate MOSes, several fixes all over the map, and quite a lot of anchors were added into the mix.

 - Big change (mostly backward compatible, minus a couple of bug fixes) to the block's code.
 - * Now `vco` can be `boxed`
 - * enabled more short-name geographical anchors
 - * generic blocks can be made rectangular
 - * mid-way lateral anchors for all blocks, as well as up/down
 - * renamed converters anchors (old ones retained for backward compatibility)
 - * new `ac/ac` blocks, both single- and three-phase
 - Added double gate MOS transistors (by Romano Giannetti)
 - Fix deformed shape for legacy TL component ([issue on GitHub](#))

- Added several anchors on variable components, suggested by [Dr Matthias Jung](#)
 - Added `genericssplitter` component (by [frankplow](#))
 - Fix - reshape `splitter` using `/tripoles/splitter/width` and `/tripoles/splitter/height` rather than `/tripoles/wilkinson/width` and `/tripoles/wilkinson/height`.
- Version 1.5.5 (2022-11-12)
New features for optoelectronic devices: a new component, arrow styling, and anchors.
 - Added styling of arrows on opto devices, thanks to a suggestion by [Dr Matthias Jung](#)
 - Added Light-Dependent resistor shape (by Romano)
 - Added `arrows` anchors to the opto-components
 - Documentation updates (rotating and flipping for path components)
- Version 1.5.4 (2022-09-09)
New components and enhancement for old ones in this version.
 - Added jumpers, inspired by a question [on TeX.stackexchange](#)
 - Added generic double bipoles, inspired by user [@erwinderboer](#) [on GitHub](#)
 - Added styling for the transistor bodydiode, suggested by user [Alex Ghilas](#) [on TeX.stackexchange](#)
 - Additions to the manual (how to remove pins on amplifiers)
- Version 1.5.3 (2022-07-02)
Minor release: fixes to the manual, and a new component (Shockley diodes).
 - Merging changes to fix the language in the manual (thanks to Charles B. Cameron, user [@cameroncb1](#) [on GitHub](#))
 - Added Shockley diode (suggested by [\[@dauph\]\(https://tex.stackexchange.com/questions/646039/creating-a-shockley-diode-in-circuitikz\)](#))
- Version 1.5.2 (2022-05-08)
Adding a couple of new component and a nice feature to transistors and tubes.
 - Added TVS diodes (transorb), suggested by [Anisio Rogerio Braga](#)
 - Added proximity switches, suggested by [Anisio Rogerio Braga](#)
 - Added partially drawn tube and transistor borders, suggested by [Jether Fernandes Reis](#)
- Version 1.5.1 (2022-04-26)
Bug fix release.
 - Do not load package `regexpatch` by default, thanks to [GitHub user alceu-git](#)
- Version 1.5.0 (2022-04-22)
In this version, several internal changes have been included in order to streamline and organize better the components and to change the management of color. The changes are pretty deep and subtle, so a bug or unexpected behaviour is always possible. You can use the 1.4.6 rollback point in case of trouble, but be sure to report any bug.
 - Added connectors shapes, and included the BNC into that class; thanks to [Alexander Sauter for suggesting them and helping in the design](#)
 - Added nullator and norator shapes, suggested by [user atticus-sullivan on GitHub](#)
 - Added buzzer and reversed buzzer bipoles, suggested by [user Michael.H](#)
 - Added “dot” anchors to inductances
 - Added “boxed only” option for some circular blocks, suggested by [user myzinsky](#)
 - Added DIN antenna shape, suggested by [user myzinsky](#)
 - Fixed block/input arrow connection, thanks to [Laurenz Preindl for reporting](#)

- Fixed a problem with generic tunable arrows, noticed thanks to [this question on TeX.SX](#)

Internal changes:

- Added a generic drawing function for shapes, which are now drawn always in background
 - Added a hook system to be able to change component drawing settings per-shape, per-class or globally
 - All the 250+ shapes are now “protected” by possible external arrow and arced corners parameters
 - Completely changed the management of the shapes’ color, thanks to [GitHub user muzimuzhi](#)
- Version 1.4.6 (2022-02-04)
A nasty bug fix and some hack to avoid that some global TikZ option spill into the shapes. To better solve that problem, some risky changes are due, so this release will be also a rollback point for compatibility reasons.
 - Fix bug with legacy transmission lines in **overlays** ([noticed by Benedikt Wilde](#))
 - Robustify some shapes: do not let arrows option pass to the inner drawing (see [here](#) and [here](#))
 - Add warning about global draw options in the manual
 - Fixes in documentation: hyperlink the index again, cite new recovery point, remove some legacy construct
 - Added 1.4.6 rollback point
 - Version 1.4.5 (2021-12-06)
Important fix for ConTeXt users, thanks to @TeXnician for reporting.
 - Fixed an incompatibility introduced with subcircuits that made compilation in ConTeXt fail
 - Added `\ctikzflip[x][y]` utility macros for ConTeXt too
 - Fixed stray characters in some TikZ environment
 - Version 1.4.4 (2021-10-31)
Normal maintenance release; minor bugs fixed, a new component and a new option. No Halloween component, sorry...
 - Added a laser diode component ([contributed by André Alves](#))
 - Add the **override source vif** option and better describe source’s voltage positioning in the manual
 - fix **nobase** option with IGBT family ([noticed by user hinata exc on Stack Eschange](#))
 - fix a problem with [legacy open voltage label position](#)
 - Version 1.4.3 (2021-09-06)
Minor release, mainly a single bugfix.
 - added hidden anchors of **oosource** to the manual
 - fix a bug in anchors of **oosource** (they did not respect class scaling)
 - faster **use fpu reciprocal** (thanks to Henri Menke)
 - Version 1.4.2 (2021-07-26)
This is a minor release, containing just a new component and a small set of fixes (mainly in the documentation).
 - add the **cpe** (constant phase element)
 - correct minor errors in the manual (capacitor’s fill, spaces) and the code.

- Version 1.4.1 (2021-07-14)
This version has an important bug fix for label positioning when once-relative style coordinates are used (the ones with a single `+`, like `+(1,1)`). Moreover, the possibility to have voltage, current and flow labels *without* the symbols (arrows, etc) has been added, which greatly simplify some kind of personalization of these elements.
 - Added the generic tunable macro
 - Added `no v symbols` (and also for `i` and `f`), thanks to a [head-up by user judober on GitHub](#), see also [issue 448](#)
 - Fixed [label position for `+`\(\) style coordinates](#)
- Version 1.4.0 (2021-07-06)
The main news is that *package rollback* for `circuitikz` has been implemented (LaTeX-only, of course). Additionally, a small but important change in the path (`to`) construction that should fix some warning from TikZ and give better line joins in wire corners.
 - bump version to 1.4.0
 - implement the version rollback: time travel to 0.4!
 - remove a wrong movement in the path construction (potentially dangerous)
- Version 1.3.9 (2021-06-27)
Bugfix release: `open poles opacity` was not working in most of the cases.
 - minor fixes to the manual
 - fix bug with `open poles opacity`; see [this question by Florian H.](#) for details.
- Version 1.3.8 (2021-06-15)
The big news of this release is the ability to selectively draw the pins of the integrated circuit and mux-demuxes symbols.
 - Add `draw only pins` feature to `dipchip` and `qfpchip`, thanks to [Jonathan P. Spratte](#), and a similar option to control the pins of `muxdemux`
 - Make `dipchip` and `qfpchip` respect `no input leads` option
 - Several corrections to the manual
- version 1.3.7 (2021-06-01)
Minor release, mainly documentation upgrades.
 - New options for the line thickness, rotation and size of symbols drawn in sources
 - New tutorial: drawing a circuit around an operational amplifier
 - Documentation fixes and small enhancements
- version 1.3.6 (2021-05-09)
Mainly a bugfix release; fixing a bug in the 12 stacked labels means that old constructs that were failing silently can give an error now. Sorry. To compensate, I added stacked annotation (for symmetry).
 - Added stacked annotations for symmetry with stacked labels.
 - Fixed a bug in the plotting of `inst amp ra` terminals.
 - Fixed a bug in managing stacked labels (`l2=...`); possibly it will be mildly backward-incompatible (please see the manual about incompatible changes)
- Version 1.3.5 (2021-05-02)
Power electronics devices are the main characters in this release: PUT, GTOs, a new style for thyristors, and a photovoltaic module. Additionally, an **experimental** support for subcircuits has been added; it could change in the future. Fixed a nasty bug in rotary switches “in” anchor positioning in some cases.

- Added support for creating and using sub-circuits
- Added UJT transistors and GTO devices ([suggested by JetherReis](#))
- Added (as an option) a different, more compact style for thyristor-type devices.
- Added a photovoltaic module ([suggested by André Alves](#))
- Added a DC/DC converter block for symmetry ([suggested by Pratched](#))
- Added the possibility to change the waveforms shown in the oscilloscope ([suggested by Mario Tafur](#))
- In the manual, separate the component usage chapter from the big component list
- Fix wrong rotary switch “in” anchors for switches with more than 180 degrees coverage ([see bug](#))
- Version 1.3.4 (2021-04-20)

New things, like configurable modifier thickness, ferroelectric devices, and several transistor tweaks. Importantly, a bug that hindered compatibility with the internal `TikZ circuits` library (introduced in 1.3.3) has been fixed.

 - Added separate configuration for the line thickness of resistors, capacitors, and inductors modifiers
 - Added ferroelectric capacitors and ferroelectric gate MOS/FETs ([suggested by Mayeul Cantan](#))
 - Added an option to fill the gate gap in MOSes, FETs and IGBTs with a color
 - Added a “centergap” anchor for transistors
 - Added the option “nogate” to the `hemt` symbol
 - Fixed a bug in thermistors not respecting their class line thickness
 - Fixes in the manual (copy and paste of snippets without numbers, correct old usage of `siunitx`, factor out repetitions in the preamble; [thanks to Ulrike Fischer](#).
 - Fixed a bug introduced in 1.3.3 that would reduce compatibility with the `circuits` internal library; [reported by JetherReis](#))
- Version 1.3.3 (2021-04-04)

Several usability additions in this version, and one small fix that could change the look of your circuit (without affecting correctness). Some of the arrow shapes are now configurable. Do not use this version, there is a bug with the new “label distance” key.

 - Added options to fine-tune the position of labels and annotations
 - Added options to change arrow tips on variable resistors, inductors and capacitors as well as in potentiometers
 - Added options to change arrow tips on switches
 - Added anchors to inductance to add core lines
 - Fixed the default direction of tunable arrows (with an option to go back to the old ones)
- Version 1.3.2 (2021-03-14)
 - Added the simplified (2-waves) highpass and lowpass blocks
 - Added graphene FETs ([suggested by Cees Keyer](#))
 - Added left/right anchors to transistors
 - Fixed a [bug in flip-flops](#)
- Version 1.3.1 (2021-02-20)
 - Fixed a bug in “fuse” and “afuse” fill
 - Remove the voltage direction warning. Nobody really ever cared
 - Minor fixes and enhancements to the manual

- Version 1.3.0 (2021-01-19)
 - Fixed a long-standing problem with labels and similar decoration with equal signs and commas
 - Fixed a typo in the manual (thanks to @muzimuzhi on GitHub)
 - The Mother of All Code Refactoring: no real changes (modulo errors)
 - Added a rollback point to 1.2.7
- Version 1.2.7 (2020-12-27)

Bugfix release.

 - The recent temporary changes to TikZ to v3.1.8a revealed a problem in corner cases with `circuitikz` that should be fixed (thanks to Henri Menke)
- Version 1.2.6 (2020-12-16)

The highlight of this release is the option to draw circles around transistors; moreover, a handful of new component and several bug fixes.

 - added option to have transistors with circles, suggested by user @myzinsky
 - added closed position for normally open button and the other way around (suggested by user @septatrix)
 - added a `tip` anchor for push buttons
 - added text anchor for legacy `linestub` component
 - added an option for a different style of european logic xnor port (suggested by user @Schlepptop)
 - added dynode tubes electrodes (suggested by user @ferdymercury)
 - fixed a bug in style-files (thanks to user @Alex on tex.stackexchange.com)
 - added a comment about relative coords (thanks to user @septatrix)
 - several fixes to the manual
- Version 1.2.5 (2020-10-14)

Mainly a bugfix release for `raised` voltage style.

 - added macro to access labels and annotations anchors and direction
 - fixed a bug in “raised” voltages’ positions with `invert` and/or `mirror`
- Version 1.2.4 (2020-10-04)
 - several documentation enhancement
 - added a couple of block elements: allpass filter, generic two-sides block (suggested by user @myzinsky)
 - added transmission gate (only IEEE style version) suggested by several users (@SJulianS on github, Philipp Birkel on [TeX.SX](https://www.tikz.org))
 - added a resistive splitter block symbol by @matthuszagh
 - added depletion-type `nmosd` and `pmosd` MOSFET simplified symbols
 - added depletion-type `nfetd` and `pfetd` for plain full-symbol MOSFET
- Version 1.2.3 (2020-08-07)

Several fixes and small enhancement all over the map, changes in the documentation to better explain the reasons and effect of the path-building changes of 1.2.0 and 1.2.1.

 - added a Mach-Zehnder-Modulator block symbol as node `mzm` by user @dl1chb
 - add an `open poles fill` option to simplify circuits where the background is different from white
 - restyled the FAQ and added the explanation of “gaps with `nodes`” that happens again after 1.2.1

- Fixed size of “not circle” in flip-flops to match european style `not circle` when used without the IEEE style
 - Block anchors: add border anchors for round elements and deprecate old 1, 2, 3, 4 anchors
 - Fixed some bipole border size to avoid overlapping labels; document it
- Version 1.2.2 (2020-07-15)

Bug-fix release: coordinate name leakage. The node and coordinate names are global; the internal coordinate names have been made stronger.
- Version 1.2.1 (2020-07-06)

Several changes, both internal and user-visible. These are quite risky, although they *should* be backward-compatible (if the circuit code is correct).

From the user point of view:

 - there is now a new style of voltages (“raised American”)
 - a powerful mechanism for customize voltages, current and flows has been added.

The internal changes are basically the re-implementation of the macros that draw the path elements (`to[...]`), which have been completely rewritten. Please be sure to read the possible incompatibilities in the manual (section 1.9).

 - Added access to voltages, currents and flows anchors
 - Added “raised american” voltage style
 - Rewrite of the path generation macros
 - Several small bugs fixed (no one ever used some “f^>” options...)
- Version 1.2.0 (2020-06-21)

In this release, the big change is the rewriting of the voltages output routine. Now all voltage options (american, european, and straight) take into account the shape (square border) of the component. The adjusting parameters are now (at least for passive elements) acting in similar way for all the options, too.

 - Bumped version number to 1.2 (potentially incompatible changes!)
 - Added 1.1.2 checkpoint
 - New path-style not, buffer, and Schmitt logic ports
 - New tutorial (using the “inline not” component)
 - Voltage output routine rewrite; now it takes into account the shape of the component also for “american” and “straight” voltages
 - Several fixes in the logic ports: fixed IEEE `invschmitt` name, added symmetry to the three-style shorthands for the ports, and so on
 - Fixed a gross bug in square poles anchor borders
 - Fixed size of not circles in flip-flops (based on logic ports style)
 - Fixed the order of initial options, to avoid “european” overwriting single options
- Version 1.1.2 (2020-05-17)
 - Blocks and component for three-phase networks (3-lines wire, AC/DC and DC/AC converters blocks and grid node block) added by user `@olfline` on GitHub
 - added transformer sources with optional vector groups for three-phase networks by `@olfline` on Github
 - added subsections to the examples
 - fixed position of american voltages on open circuits (suggested by user `@rhandley` on GitHub)
- Version 1.1.1 (2020-04-24)

One-line bugfix release for the IEEE ports “not” circle thickness

- Version 1.1.0 (2020-04-19)

Version bumped to 1.1 because the new logic ports are quite a big addition: now there is a new style for logic ports, conforming to IEEE recommendations.

Several minor additions all over the map too.

- added IEEE standard logic ports suggested by user Jason-s on GitHub
- added configurability to european logic port “not” output symbol, suggested by j-hap on GitHub
- added **inverter** component by user Tadashi on GitHub
- added variable outer base height for IGBT, suggested by user RA-EE on GitHub
- added configurable “+” and “-” signs on american-style voltage generators
- text on amplifiers can be positioned to the left or centered

- Version 1.0.2 (2020-03-22)

- added Schottky transistors (thanks to a suggestion by Jérôme Monclard on GitHub)
- fixed formatting of `CHANGELOG.md`

- Version 1.0.1 (2020-02-22)

Minor fixes and addition to 1.0, in time to catch the freeze for TL2020.

- add v1.0 version snapshots
- added crossed generic impedance (suggested by Radványi Patrik Tamás)
- added open barrier bipole (suggested by Radványi Patrik Tamás)
- added two flags to flip the direction of light’s arrows on LED and photodiode (suggested by karlkappe on GitHub)
- added a special key to help with precision loss in case of fractional scaling (thanks to AndreaDiPietro92 on GitHub for noticing and reporting, and to Schrödinger’s cat for finding a fix)
- fixed a nasty bug for the flat file generation for ConTeXt

- Version 1.0 (2020-02-04)

And finally... version 1.0 (2020-02-04) of **circuitikz** is released.

The main updates since version 0.8.3, which was the last release before Romano started co-maintaining the project, are the following — part coded by Romano, part by several collaborators around the internet:

- The manual has been reorganized and extended, with the addition of a tutorial part; tens of examples have been added all over the map.
- Around 74 new shapes were added. Notably, now there are chips, mux-demuxes, multi-terminal transistors, several types of switches, flip-flops, vacuum tubes, 7-segment displays, more amplifiers, and so on.
- Several existing shapes have been enhanced; for example, logic gates have a variable number of inputs, transistors are more configurable, resistors can be shaped more, and more.
- You can style your circuit, changing relative sizes, default thickness and fill color, and more details of how you like your circuit to look; the same you can do with labels (voltages, currents, names of components and so on).
- A lot of bugs have been squashed; especially the (very complex) voltage direction conundrum has been clarified and you can choose your preferred style here too.

A detailed list of changes can be seen below.

- Version 1.0.0-pre3 (not released)

- Added a Reed switch

- Put the copyright and license notices on all files and update them
- Fixed the loading of style; we should not guard against reload
- Version 1.0.0-pre2 (2020-01-23)

Really last additions toward the 1.0.0 version. The most important change is the addition of multiplexer and de-multiplexers; also added the multi-wires (bus) markers.

 - Added mux-demux shapes
 - Added the possibility to suppress the input leads in logic gates
 - Added multiple wires markers
 - Added a style to switch off the automatic rotation of instruments
 - Changed the shape of the or-type american logic ports (reversible with a flag)
- Version 1.0.0-pre1 (2019-12-22)

Last additions before the long promised 1.0! In this pre-release we feature a flip-flop library, a revamped configurability of amplifiers (and a new amplifier as a bonus) and some bug fix around the clock.

 - Added a flip-flop library
 - Added a single-input generic amplifier with the same dimension as “plain amp”
 - Added border anchors to amplifiers
 - Added the possibility (expert only!) to add transparency to poles (after a suggestion from user @matthuszagh on GitHub)
 - Make plus and minus symbol on amplifiers configurable
 - Adjusted the position of text in triangular amplifiers
 - Fixed “plain amp” not respecting “noinv input up”
 - Fixed minor incompatibility with ConTeXt and Plain TeX
- Version 0.9.7 (2019-12-01)

The important thing in this release is the new position of transistor’s labels; see the manual for details.

 - Fix the position of transistor’s text. There is an option to revert to the old behavior.
 - Added anchors for adding circuits (like snubbers) to the flyback diodes in transistors (after a suggestion from @EdAlvesSilva on GitHub).
- Version 0.9.6 (2019-11-09)

The highlights of this release are the new multiple terminals BJTs and several stylistic addition and fixes; if you like to pixel-peep, you will like the fixed transistors arrows. Additionally, the transformers are much more configurable now, the “pmos” and “nmos” elements have grown an optional bulk connection, and you can use the “flow” arrows outside of a path. Several small and less small bugs have been fixed.

 - Added multi-collectors and multi-emitter bipolar transistors
 - Added the possibility to style each one of the two coils in a transformer independently
 - Added bulk connection to normal MOSFETs and the respective anchors
 - Added “text” anchor to the flow arrows, to use them alone in a consistent way
 - Fixed flow, voltage, and current arrow positioning when “auto” is active on the path
 - Fixed transistors arrows overshooting the connection point, added a couple of anchors
 - Fixed a spelling error on op-amp key “noinv input down”
 - Fixed a problem with “quadpoles style=inner” and “transformer core” having the core lines running too near

- Version 0.9.5 (2019-10-12)

This release basically add features to better control labels, voltages and similar text “ornaments” on bipoles, plus some other minor things.

On the bug fixes side, a big incompatibility with ConTeXt has been fixed, thanks to help from [@TheTeXnician](#) and [@hmenke](#) on [github.com](#).

- Added a “midtap” anchor for coils and exposed the inner coils shapes in the transformers
- Added a “curved capacitor” with polarity coherent with “ecapacitor”
- Added the possibility to apply style and access the nodes of bipole’s text ornaments (labels, annotations, voltages, currents and flows)
- Added the possibility to move the wiper in resistive potentiometers
- Added a command to load and set a style in one go
- Fixed internal font changing commands for compatibility with ConTeXt
- Fixed hardcoded black color in “elko” and “elmech”

- 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 thickness
- 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/appearance 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 $\cdot\cdot$ 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 global 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, nignfet, nignfetd, pfet, pigfet, pigfetd (contribution of Clemens Helfmeier and Theodor Borsche)
 - njfet, pjfet (contribution of Danilo Piazzalunga)
 - pigbt, night
 - *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 TikZ 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, [93](#)
adder, [91](#)
afuse, [82](#)
ageneric, [50](#)
aGTOb, *see* agtobar
aGTOb*, *see* full agtobar
aGTOb-, *see* stroke agtobar
agtobar, [64](#), [65](#)
aGTObo, *see* empty agtobar
allpass, [93](#)
ALU, [173](#)
american and port, [154](#)
american buffer port, [154](#)
american controlled current source, [69](#)
american controlled voltage source, [69](#)
american current source, [68](#)
american gas filled surge arrester, [82](#)
american inductive sensor, *see* sL
american inductor, *see* L
american light dependent resistor, *see* ldR
american nand port, [154](#)
american nor port, [154](#)
american not port, [154](#)
american or port, [154](#)
american potentiometer, *see* pR, *see* pR
american resistive sensor, *see* sR
american resistor, *see* resistor, *see* R
american voltage source, [68](#)
american xnor port, [154](#)
american xor port, [154](#)
ammeter, [36](#), [75](#)
amp, [93](#)
antenna, [127](#)
asymmetric fuse, *see* afuse

bandpass, [92](#)
bandstop, [92](#)
bare jumper, [151](#)
bare7seg, [180](#)
bareantenna, [127](#)
bareRXantenna, [127](#)
bareTXantenna, [127](#)
barrier, [82](#)
battery, [67](#)
battery1, [67](#)
battery2, [67](#)
biD*, *see* full bidirectionaldiode
biDo, *see* empty bidirectionaldiode
bjtnpn, collectors=1, emitters=2, [101](#)
bjtnpn, collectors=2, emitters=2, bjt pins
width=0, bjt multi height=0.8, [113](#)
bjtpnp, collectors=3, emitters=2, [101](#)
bmultiwire, *see* bmultiwire, [84](#)
bnc, [89](#)
buffer, [138](#)

bulb, [83](#)
buzzer, [83](#)

C, *see* capacitor
capacitive sensor, [55](#)
capacitor, [54](#)
cC, *see* curved capacitor
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, [48](#)
circ, [88](#)
circleinv, scale=2, [172](#)
circulator, [91](#)
cisource, *see* european controlled current source
cisourceAM, *see* american controlled current
source
cisourceC, *see* cute european controlled current
source
courcesin, *see* controlled sinusoidal current
source
clockwedge, [168](#)
closed double solder jumper, [153](#)
closed jumper, [151](#)
closed solder jumper, [153](#)
closing switch, [144](#)
cogsw, *see* cute opening switch
controlled isourcesin, *see* controlled sinusoidal
current source
controlled sinusoidal current source, [70](#)
controlled sinusoidal voltage source, [70](#)
controlled vsourcesin, *see* controlled sinusoidal
voltage source
cosw, *see* cute open switch
coupler, [95](#)
coupler2, [95](#)
cpe, *see* cpe, [55](#)
crossing, [85](#)
csI, *see* controlled sinusoidal current source
cspst, *see* closing switch
csV, *see* controlled sinusoidal voltage source
curarrow, [86](#)
curved capacitor, [54](#)
cute choke, [57](#)
cute closed switch, [145](#)
cute closing switch, [145](#)
cute european controlled current source, [69](#)
cute european controlled voltage source, [69](#)
cute european current source, [68](#)
cute european voltage source, [68](#)
cute inductive sensor, *see* sL

cute inductor, *see* L
 cute open switch, 145
 cute opening switch, 145
 cute spdt down, 146
 cute spdt down arrow, 38, 146
 cute spdt mid, 146
 cute spdt mid arrow, 146
 cute spdt up, 145
 cute spdt up arrow, 146
 cvsource, *see* european controlled voltage source
 cvsourceAM, *see* american controlled voltage source
 cvsourceC, *see* cute european controlled voltage source
 cvsourcesin, *see* controlled sinusoidal voltage source

 D*, *see* full diode
 D-, *see* stroke diode
 dac, 93
 damper, 81, 224
 dcisource, 73
 dcvsources, 73
 demux, 172
 detector, 94
 diamondpole, 88
 dinantenna, 127
 diodetube, 121
 diodetube.filament, 122
 diodetube.filament.nocathode, 122
 diodetube.fullcathode, 123
 dipchip, 177
 Do, *see* empty diode
 double bipole, 132
 dsp, 93
 dynode, 125

 eC, *see* ecapacitor
 ecapacitor, 54
 ecsources, *see* empty controlled source
 eground, 48
 eground2, 48
 elko, *see* ecapacitor
 elmech, 129
 empty agtobar, 64
 empty bidirectionaldiode, 60
 empty controlled source, 69
 empty diode, 60
 empty gto, 63
 empty gtobar, 64
 empty laser diode, 60
 empty led, 60
 empty photodiode, 60
 empty put, 63
 empty Schottky diode, 60
 empty Shockley diode, 60
 empty thyristor, 63
 empty triac, 62
 empty tunnel diode, 60

 empty TVS diode, 60
 empty varcap, 60
 empty Zener diode, 60
 empty ZZener diode, 60
 en amp, 137
 esource, 71
 european and port, 156
 european buffer port, 156
 european controlled current source, 69
 european controlled voltage source, 69
 european current source, 68
 european gas filled surge arrester, 82
 european inductive sensor, *see* sL
 european inductor, *see* L
 european light dependent resistor, *see* ldR
 european nand port, 156
 european nor port, 156
 european not port, 157
 european or port, 156
 european potentiometer, *see* pR
 european resistive sensor, *see* sR
 european resistor, *see* R
 european voltage source, 68
 european xnor port, 156
 european xor port, 156

 fd inst amp, 138
 fd op amp, 137
 feC, 55
 ferrocap, *see* feC
 fft, 93
 flipflop, 167
 flipflop AB, 167
 flipflop D, 168
 flipflop JK, 168
 flipflop JK, add async SR, 169
 flipflop JK, add async SR, external pins width=0, 170
 flipflop JK, dot on notQ, 168
 flipflop myJK, 169
 flipflop SR, 168
 flipflop T, 168
 flowarrow, 86
 fourport, 94
 full agtobar, 64
 full bidirectionaldiode, 61
 full diode, 61, 62
 full gto, 63
 full gtobar, 64
 full laser diode, 61
 full led, 61
 full photodiode, 61
 full put, 63
 full Schottky diode, 61
 full Shockley diode, 61
 full thyristor, 63
 full triac, 63
 full tunnel diode, 61
 full TVS diode, 61

full varcap, [61](#)
 full Zener diode, [61](#)
 full ZZener diode, [61](#)
 fuse, [82](#)

 GaN hemt, [102](#)
 generic, [50](#)
 genericsplitter, [92](#)
 gm amp, [138](#)
 gridnode, [92](#)
 ground, [48](#)
 GTO, *see* gto, *see* gto
 gto, [63](#), [65](#)
 GTO*, *see* full gto
 GTO-, *see* stroke gto
 GTOb, *see* gtobar
 GTOb*, *see* full gtobar
 GTOb-, *see* stroke gtobar
 gtobar, [64](#), [65](#)
 GTObo, *see* empty gtobar
 GTOo, *see* empty gto
 gyrator, [131](#)

 hemt, [101](#)
 hemt, nobase, [102](#)
 highpass, [92](#)
 highpass2, [93](#)

 iec connector, [89](#)
 ieconn, *see* iec connector
 ieconnshape, [89](#)
 iecplugL, [90](#)
 iecplugR, [90](#)
 iecsocketL, [89](#)
 iecsocketR, [90](#)
 ieee double tgate, [156](#)
 ieee tgate, [156](#)
 ieeestd and port, [155](#)
 ieeestd buffer port, [155](#)
 ieeestd invschmitt port, [156](#)
 ieeestd nand port, [155](#)
 ieeestd nor port, [155](#)
 ieeestd not port, [155](#)
 ieeestd or port, [155](#)
 ieeestd schmitt port, [155](#)
 ieeestd xnor port, [155](#)
 ieeestd xor port, [155](#)
 iloop, [76](#)
 iloop2, [76](#)
 inerter, [81](#)
 inline buffer, [157](#)
 inline double tgate, [157](#)
 inline invschmitt, [157](#)
 inline not, [157](#)
 inline proximeter, [147](#)
 inline schmitt, [157](#)
 inline tgate, [157](#)
 inputarrow, [86](#)
 inst amp, [138](#)

 inst amp ra, [138](#)
 invschmitt, [154](#)
 ioosource, [71](#)
 isfet, [105](#)
 isource, *see* european current source
 isourceAM, *see* american current source
 isourceC, *see* cute european current source
 isourceN, *see* noise current source
 isourcesin, *see* sinusoidal current source

 jump crossing, [85](#)

 L, [56](#), [57](#)
 lamp, [83](#)
 lasD, *see* empty laser diode
 lasD*, *see* full laser diode
 lasD-, *see* stroke laser diode
 latch, [168](#)
 ldR, [51](#)
 leD*, *see* full led
 leD-, *see* stroke led
 leDo, *see* empty led
 left double solder jumper, [153](#)
 Lnigbt, [100](#)
 loudspeaker, [83](#)
 lowpass, [93](#)
 lowpass2, [93](#)
 Lpigbt, [100](#)
 Lpigbt, bodydiode, [100](#)

 magnetron, [125](#)
 mass, [81](#)
 match, [128](#)
 memristor, [51](#)
 mic, [83](#)
 mixer, [91](#)
 mixer, boxed, [91](#)
 Mr, *see* memristor
 mslstub, [127](#)
 msport, [127](#)
 msrstub, [127](#)
 mstline, [127](#)
 multiwire, *see* multiwire, [84](#)
 muxdemux, [172](#)
 mzm, [92](#)

 ncpb, *see* normally closed push button
 ncpbo, *see* normally closed push button open
 ncs, *see* normal closed switch
 nfet, [102](#)
 nfetd, [102](#)
 ngfet, [105](#)
 ngfetd, [48](#)
 nI, *see* noise current source
 nigbt, [100](#), [111](#)
 nigfetd, [102](#)
 nigfetd, doublegate, [103](#)
 nigfete, [102](#)
 nigfete, doublegate, [103](#)

nigfete,solderdot, 102
nigfete,solderdot, doublegate, 103
nigfetebulk, 102
nigfetebulk, doublegate, 103
njfet, 104
nmos, 101, 106
nmos, bulk, 111
nmosd, 101, 106
nmosd, bulk, 111
noise current source, 70
noise voltage source, 70
nopb, *see* push button
nopbc, *see* normally open push button closed
norator, 72
normal closed switch, 144
normal open switch, 144
normally closed push button, 144
normally closed push button open, 144
normally open push button, *see* push button
normally open push button closed, 144
nos, *see* normal open switch
notcirc, 156
npn, 38, 99
npn, bodydiode, 100
npn, schottky base, 99
npn, tr circle, 114
npn, tr circle, bodydiode, 114
npn,photo, 100
nujt, 104
nujt, nobase, 105
nullator, 72
nV, *see* noise voltage source

ocirc, 88
odiamondpole, 88
ohmmeter, 75
one bit adder, 172
oosource, 72
oosourcetrans, 72
op amp, 20, 137
open, 50
open double solder jumper, 153
open jumper, 151
open solder jumper, 153
openbarrier, 82
opening switch, 144
oscillator, 91
oscope, 76
ospst, *see* opening switch
osquarepole, 88

pD*, *see* full photodiode
pD-, *see* stroke photodiode
pDo, *see* empty photodiode
pentode, 122
pentode suppressor to cathode, 122
pfet, 102
pfetd, 103
pgfet, 105

pground, 48
phaseshifter, 93
photoresistor, *see* phR
phR, 52
piattenuator, 93
piezoelectric, 55
pigbt, 100
pigbt, nobase, 111
pigfetd, 103
pigfetd, doublegate, 104
pigfete, 103
pigfete, doublegate, 103
pigfetebulk, 103
pigfetebulk, doublegate, 103
pjfet, 104
plain amp, 38, 138
plain crossing, 85
plain mono amp, 138
pmos, 101, 106
pmos, bulk, 111
pmos,emptycircle, 111
pmos,nocircle,arrowmos, 111
pmosd, 101, 106
pmosd, bulk, 111
pnp, 99
pnp, schottky base, 100
pnp,photo, 100
pR, *see* pR, 36, 51
proximeter, 147
pujt, 104
push button, 144
PUT, *see* put, *see* put
put, 63, 65
PUT*, *see* full put
PUT-, *see* stroke put
PUTo, *see* empty put
pvmodule, 71
pvsources, 71
PZ, *see* piezoelectric

qfpchip, 177
qiprobe, 76
qprobe, 76
qvprobe, 76

R, *see* resistor, 51
rbuzzer, 83
reed, 145
resistor, 36
rground, 48
right double solder jumper, 153
rmeter, 76
rmeterwa, 76
rotaryswitch, 38, 148
rxantenna, 128

sacac, 94
sacdc, 94
sC, *see* capacitive sensor

schmitt, [154](#)
 schmitt symbol, [156](#)
 sD*, *see* full Schottky diode
 sD-, *see* stroke Schottky diode
 sdcac, [94](#)
 sdcdc, [94](#)
 sDo, *see* empty Schottky diode
 sground, [48](#)
 shD*, *see* full Shockley diode
 shDo, *see* empty Shockley diode
 short, [50](#)
 sI, *see* sinusoidal current source
 sinusoidal current source, [69](#)
 sinusoidal voltage source, [68](#)
 sL, [56](#), [57](#)
 smeter, [76](#)
 spdt, [145](#)
 splitter, [92](#)
 spring, [81](#)
 spst, *see* switch
 square voltage source, [71](#)
 squarepole, [88](#)
 squid, [82](#)
 sqV, *see* square voltage source
 sR, [51](#)
 stroke agtobar, [64](#)
 stroke diode, [61](#), [62](#)
 stroke gto, [64](#)
 stroke gtobar, [64](#)
 stroke laser diode, [62](#)
 stroke led, [62](#)
 stroke photodiode, [62](#)
 stroke put, [63](#)
 stroke Schottky diode, [62](#)
 stroke thyristor, [63](#)
 stroke tunnel diode, [62](#)
 stroke varcap, [62](#)
 stroke Zener diode, [62](#)
 stroke ZZener diode, [62](#)
 sV, *see* sinusoidal voltage source
 switch, [144](#)

 tacac, [94](#)
 tacdc, [94](#)
 tattuenuator, [93](#)
 tD*, *see* full tunnel diode
 tD-, *see* stroke tunnel diode
 tdcac, [94](#)
 tDo, *see* empty tunnel diode
 tetrode, [121](#)
 tgeneric, [50](#)
 tground, [48](#)
 thermistor, *see* thR
 thermistor ntc, *see* thRn
 thermistor ptc, *see* thRp
 thermocouple, [82](#)
 thR, [52](#)
 three-pins jumper, [152](#)
 thRn, [52](#)

thRp, [52](#)
 thyristor, [63](#), [64](#)
 TL, [128](#)
 TL, bipoles/tline/bare=true, [128](#)
 tlground, [48](#)
 tline, *see* TL, bipoles/tline/bare=true, *see* TL
 tlinestub, [128](#)
 tlmic, [83](#)
 tmultiwire, *see* tmultiwire, [84](#)
 toggle switch, [144](#)
 Tr, *see* triac, *see* triac
 Tr*, *see* full triac
 transformer, [131](#)
 transformer core, [131](#)
 transmission line, *see* TL,
 bipoles/tline/bare=true, *see* TL
 trarrow, [86](#)
 triac, [62](#), [64](#)
 triode, [121](#)
 Tro, *see* empty triac
 tV, *see* vsourcetri
 tvsD*, *see* full TVS diode
 tvsDo, *see* empty TVS diode
 twoport, [92](#)
 twoportsplit, [92](#)
 txantenna, [128](#)
 Ty, *see* thyristor
 Ty*, *see* full thyristor
 Ty-, *see* stroke thyristor
 Tyo, *see* empty thyristor

 vamp, [93](#)
 variable american inductor, *see* vL
 variable american resistor, *see* vR
 variable capacitor, [55](#)
 variable cute inductor, *see* vL
 variable european inductor, *see* vL
 variable european resistor, *see* vR
 varistor, [52](#)
 vC, *see* variable capacitor
 VC*, *see* full varcap
 VC-, *see* stroke varcap
 vcc, [49](#)
 VCo, *see* empty varcap
 vco, [92](#)
 vco,box, [92](#)
 vee, [49](#)
 viscoe, [81](#), [227](#)
 vL, [56](#), [57](#)
 voltmeter, [75](#)
 voosource, [71](#)
 vphaseshifter, [94](#)
 vpiattenuator, [93](#)
 vR, [51](#)
 vsource, *see* european voltage source
 vsourceAM, *see* american voltage source
 vsourceC, *see* cute european voltage source
 vsourceN, *see* noise voltage source
 vsourcesin, *see* sinusoidal voltage source

vsourcesquare, *see* square voltage source
vsourcetri, [71](#)
vtattenuator, [93](#)
waves, [127](#)
wedgeinv, scale=2, [172](#)
wilkinson, [91](#)
xgeneric, [50](#)

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