



CircuiTikZ
version 1.2.3-738180c (2020/07/21)

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)

July 21, 2020

Contents

1	Introduction	7
1.1	About	7
1.2	License	7
1.3	Loading the package	7
1.4	Installing a new version of the package.	8
1.5	Requirements	8
1.6	Incompatible packages	8
1.7	Known bugs and limitation	8
1.8	Scale factors inaccuracies	9
1.9	Incompabilities between version	9
1.10	Feedback	10
1.11	Package options	11
2	Tutorials	14
2.1	Getting started with CircuiTikZ: a current shunt	14
2.2	A more complex tutorial: circuits, Romano style.	17
2.3	Tutorial: a logic circuit	22
3	The components	25
3.1	Path-style components	25
3.1.1	Anchors	25
3.1.2	Customization	26
3.1.2.1	Components size	26
3.1.2.2	Thickness of the lines	27
3.1.2.3	Shape of the components	28
3.1.3	Descriptions	28
3.2	Node-style components	29
3.2.1	Mirroring and flipping	29
3.2.2	Anchors	30
3.2.3	Descriptions	30
3.3	Styling circuits and components	31
3.3.1	Relative size	31
3.3.2	Fill color	33
3.3.3	Line thickness	34
3.3.4	Style files	34
3.3.5	Style files: how to write them	35
3.4	Grounds and supply voltages	36
3.4.1	Grounds	36
3.4.1.1	Grounds anchors	37

3.4.1.2	Grounds customization	37
3.4.2	Power supplies	37
3.4.2.1	Power supply anchors	37
3.4.2.2	Power supplies customization	37
3.5	Resistive bipoles	38
3.5.1	Potentiometers: wiper position	40
3.5.2	Generic sensors anchors	40
3.5.3	Resistive components customization	41
3.6	Capacitors and inductors: dynamical bipoles	41
3.6.1	Capacitors	41
3.6.2	Capacitive sensors anchors	42
3.6.3	Capacitors customizations	42
3.6.4	Inductors	42
3.6.5	Inductors customizations	43
3.6.6	Inductors anchors	44
3.7	Diodes and such	44
3.7.1	Tripole-like diodes	46
3.7.2	Triacs anchors	47
3.7.3	Diode customizations	47
3.8	Sources and generators	48
3.8.1	Batteries	48
3.8.2	Stationary sources	48
3.8.3	Sinusoidal sources	49
3.8.4	Controlled sources	50
3.8.5	Noise sources	51
3.8.6	Special sources	52
3.8.7	DC sources	53
3.8.8	Sources customizations	53
3.9	Instruments	54
3.9.1	Instruments customizations	55
3.9.2	Rotation-invariant elements	55
3.9.3	Instruments as node elements	56
3.9.4	Measuring voltage and currents, multiple ways	56
3.10	Mechanical Analogy	59
3.10.1	Mechanical elements customizations	59
3.11	Miscellaneous bipoles	60
3.11.1	Miscellaneous element customization	61
3.12	Multiple wires (buses)	61
3.13	Crossings	62
3.14	Arrows	63
3.14.1	Arrows size	63

3.15	Terminal shapes	64
3.15.1	BNC connector/terminal	65
3.16	Block diagram components	65
3.16.1	Blocks anchors	68
3.16.2	Blocks customization	70
3.16.2.1	Multi ports	70
3.16.2.2	Labels and custom two-port boxes	70
3.16.2.3	Box option	71
3.16.2.4	Dash optional parts	71
3.17	Transistors	71
3.17.1	Standard bipolar transistors	71
3.17.2	Multi-terminal bipolar transistors	72
3.17.3	Field-effect transistors	73
3.17.4	Transistor texts (labels)	75
3.17.5	Transistors customization	75
3.17.5.1	Size.	75
3.17.5.2	Arrows.	75
3.17.5.3	Body diodes and similar things.	76
3.17.5.4	Schottky transistors.	76
3.17.5.5	IGBT outer base	77
3.17.5.6	Base/Gate terminal.	77
3.17.5.7	Bulk terminals.	78
3.17.6	Multiple terminal transistors customization	79
3.17.7	Transistors anchors	79
3.17.8	Transistor paths	82
3.18	Electronic Tubes	83
3.18.1	Tubes customization	85
3.19	RF components	87
3.19.1	RF elements customization	88
3.19.2	Microstrip customization	88
3.20	Electro-Mechanical Devices	89
3.20.1	Electro-Mechanical Devices anchors	89
3.21	Double bipoles (transformers)	90
3.21.1	Double dipoles anchors	91
3.21.2	Double dipoles customization	92
3.21.3	Styling transformer's coils independently	93
3.22	Amplifiers	95
3.22.1	Amplifiers anchors	96
3.22.2	Amplifiers customization	98
3.22.2.1	European-style amplifier customization	100
3.22.3	Designing your own amplifier	101

3.23	Switches and buttons	101
3.23.1	Traditional switches	102
3.23.2	Cute switches	103
3.23.2.1	Cute switches anchors	104
3.23.2.2	Cute switches customization	104
3.23.3	Rotary switches	105
3.23.3.1	Rotary switch anchors	106
3.23.3.2	Rotary switch customization	107
3.24	Logic gates	107
3.24.1	American Logic gates	108
3.24.2	IEEE logic gates	109
3.24.3	European Logic gates	110
3.24.4	Path-style logic ports	111
3.24.5	American ports usage	111
3.24.5.1	American logic port customization	112
3.24.5.2	American logic port anchors	113
3.24.6	IEEE logic gates usage.	115
3.24.6.1	Stacking and aligning IEEE standard gates.	117
3.24.6.2	IEEE standard ports customization	117
3.24.6.3	IEEE standard ports anchors	119
3.24.7	European logic port usage	119
3.24.7.1	European logic port customization	119
3.24.7.2	European logic port anchors	120
3.25	Flip-flops	120
3.25.1	Custom flip-flops	122
3.25.2	Flip-flops anchors	122
3.25.3	Flip-flops customization	123
3.26	Multiplexer and de-multiplexer	124
3.26.1	Mux-Demux: design your own shape	126
3.26.2	Mux-Demux customization	127
3.26.3	Mux-Demux anchors	127
3.27	Chips (integrated circuits)	128
3.27.1	DIP and QFP chips customization	128
3.27.2	Chips anchors	129
3.27.3	Chips rotation	130
3.27.4	Chip special usage	130
3.28	Seven segment displays	131
3.28.1	Seven segments anchors	131
3.28.2	Seven segments customization	132

4	Labels and similar annotations	133
4.1	Labels and Annotations	133
4.2	Currents and voltages	135
4.2.1	Common properties of voltages and currents	139
4.3	Currents	140
4.4	Flows	141
4.5	Voltages	142
4.5.1	European style	142
4.5.2	Straight European style	144
4.5.3	American style	144
4.5.4	Raised American style	145
4.5.5	Voltage position	146
4.5.6	American voltages customization	147
4.6	Changing the style of labels and text ornaments	148
4.7	Accessing labels text nodes	148
4.8	Advanced voltages, currents and flows	149
4.8.1	Activating the anchors	150
4.8.2	Auxiliary information	151
4.9	Integration with <code>siunitx</code>	152
5	Using bipoles in circuits	153
5.1	Nodes (also called poles)	153
5.1.1	Transparent poles	155
5.2	Mirroring and Inverting	155
5.3	Putting them together	156
5.4	Line joins between Path Components	156
6	Colors	157
6.1	Shape colors	157
6.2	Fill colors	159
7	FAQ	161
8	Defining new components	162
8.1	Suggested setup	162
8.2	Path-style component	163
8.3	Node-style component	166
8.3.1	Finishing your work	166

9	Examples	167
9.1	A red diode	167
9.2	Using the (experimental) <code>siunitx</code> syntax	168
9.3	Photodiodes	169
9.4	A Sallen-Key cell	169
9.5	Mixing circuits and graphs	170
9.6	RF circuit	171
9.7	A styled low noise input stage	172
9.8	An example with the <code>compatibility</code> option	173
9.9	3-phases block schematic	174
10	Changelog and Release Notes	175
	Index of the components	184

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 © 2007–2020 by Massimo Redaelli, 2013-2020 by Stefan Erhardt, 2015-2020 by Stefan Lindner, and 2018-2020 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.

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



¹ConT_EXt support was added mostly thanks to Mojca Miklavac and Aditya Mahajan.

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 sometime not so trivial. If you search for `local texmf tree` and the name of your distribution on <https://tex.stackexchange.com/> you will find a lot of hints.

Anyway, the easiest way of using whichever version of 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 `circuitikz.sty`.

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

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

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

1.5 Requirements

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

1.6 Incompatible packages

TikZ’s own `circuit` library, which is 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[*Tmos] (2,0)`, and so on (but `(0,0) node[nmos] {}`). See example at page 173.

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 language also requires 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/).

1.7 Known bugs and limitation

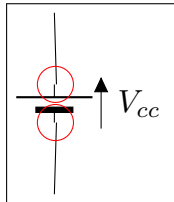
CircuiTikZ will **not work** correctly with global (in the main `circuitikz` environment, or in `scope` environments) *negative* scale parameters (`scale`, `xscale` or `yscale`), unless `transform shape` is also used, and even in this cases the behavior is not guaranteed. Neither it will work with angle-changing 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 create a lot of problem of compatibility between CircuiTikZ and the new `pic` TikZ feature, so basically don’t put components into `pics`.

1.8 Scale factors 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:



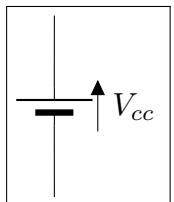
```

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 version

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

- After v1.2.1: **Important:** the routine that implement the `to[...]` component positioning has been rewritten. That should enhance the line joins in path, and it's safer, but it can potentially change 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 to be filled, not the path. That was incorrect, although sometime it was handy.

- 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: during the 1.1.0 — 1.1.2 version, 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 (you can revert to old behavior, see section 4.5.5).
- After v0.9.7: the position of the text of transistor nodes has changed; see section 3.17.4.
- 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 `nor`, `xor`, and `xnor` ports.
- After v0.9.0: voltage and current directions/sign (plus and minus signs in case of `american voltages` and arrows in case of `european voltages` have been rationalized with a couple of new options (see details in section 4.2. The default case is still the same as v0.8.3.
- Since v0.8.2: voltage and current label directions (`v<=` / `i<=`) do NOT change the orientation of the drawn source shape anymore. Use the `invert` option to rotate the shape of the source. Furthermore, from this version on, the current label (`i=`) at current sources can be used independent of the regular label (`l=`).
- Since v0.7?: The label behaviour at mirrored bipoles has changes, this fixes the voltage drawing, but perhaps you have to adjust your label positions.
- Since v0.5.1: The parts `pfet`, `pigfete`, `pigfetebulk` and `pigfetd` are now mirrored by default. Please adjust your `yscale`-option to correct this.
- Since v0.5: New voltage counting direction, there exists an option to use the old behaviour.

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

- you can use an older version locally using the git-version and picking the correct commit from the repository (branch `gh-pages`) or the main GitHub site directly;
- if you are using 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 and 1.1.2. To switch to use them, you simply change your `\usepackage` invocation like

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

You have to take care of the options that may have changed between versions;

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

```
1 \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 is 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 meet the individual gusto you can set a bunch of package options.

There are arguably way too much 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`, the global style (`american` or `european`) and use styles (see 3.3) for the remaining options.

The standard options are what the authors like, for example you get this:



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

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

However, most of the global package options are not available in ConT_EXt; 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;
- `europeanports`: uses rectangular logic ports, as per european standards;
- `americangfsgearrester`: uses round gas filled surge arresters, as per american standards;
- `europeangfsgearrester`: uses rectangular gas filled surge arresters, as per european standards;
- `european`: equivalent to `europeancurrents`, `europeanvoltages`, `europeanresistors`, `europeaninductors`, `europeanports`, `europeangfsgearrester`;
- `american`: equivalent to `americancurrents`, `americanvoltages`, `americanresistors`, `americaninductors`, `americanports`, `americangfsgearrester`;
- `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}`;
- `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 gets not 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 4.2) for the batteries. This has been fixed, but to guarantee backward compatibility and to avoid nasty surprises, the fix is available with new options:
 - **oldvoltagedirection**: Use old way of voltage direction having a difference between european and american direction, with wrong default labelling for batteries;
 - **nooldvoltagedirection**: The standard from 0.5 onward, utilize the (German?) standard of voltage arrows in the direction of electric fields (without fixing batteries);
 - **RPvoltages** (meaning Rising Potential voltages): the arrow is in direction of rising potential, like in **oldvoltagedirection**, but batteries and current sources are fixed to follow the passive/active standard;
 - **EFvoltages** (meaning Electric Field voltages): the arrow is in direction of the electric field, like in **nooldvoltagedirection**, but batteries are fixed;

If none of these option are given, the package will default to **nooldvoltagedirection**, but will give a warning. The behavior is also selectable circuit by circuit with the **voltage dir** style.

- **betterproportions**²: nicer proportions of transistors in comparision to resistors;

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

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

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

²May change in the future!

2 Tutorials

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 one that are bipoles and are placed along a path (also known as `to`-style component, for their usage), and components that are nodes and can have any number of poles or connections.

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



```
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 can surprise somebody; this is actually the european-style symbol, and the type of symbol chosen reflects the default options of the package (see section 1.11). 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) --
3     (2,3)
4     to[R=$R_1$] (2,0) -- (0,0);
5   \draw (2,3) -- (4,3) to[R=$R_2$]
6     (4,0) -- (2,0);
7 \end{circuitikz}
```

You can use a single path or multiple path when drawing your circuit, it's just a question of style (but be aware that closing path could be non-trivial, see section 5.4), 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 nodes at it, like for example in the following circuit, where we add a current (with the key `i=...`, see section 4.3) and a connection dot (with the special shortcut `-*` which adds a `circ` node at the end of the connection, see sections 3.15 and 5.1).



```

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



```

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

```

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



```

1 \begin{circuitikz}[european, voltage shift
2   =0.5]
3   \draw (0,0) to[isourceC, l=$I_0$, v=$V_0$]
4     (0,3)
5     to[short, -*, f=$I_0$] (2,3)
6     to[R=$R_1$, f>_=$i_1$] (2,0) -- (0,0);
7   \draw (2,3) -- (4,3)
8     to[R=$R_2$, f>_=$i_2$]
9       (4,0) to[short, -*] (2,0);
10 \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) to[isource, l=$I_0$, v=$V_0$]
    (0,3)
3   to[short, -*, f=$I_0$] (2,3)
4   to[R=$R_1$, f>_=$i_1$] (2,0) -- (0,0);
5   \draw (2,3) -- (4,3)
6   to[R=$R_2$, f>_=$i_2$]
7   (4,0) to[short, -*] (2,0);
8   \draw[red, thick] (1.5,2.5) rectangle
    (4.5,3.5)
9   node[pos=0.5, above]{KCL};
10 \end{circuitikz}

```

2.2 A more complex tutorial: circuits, Romano style.

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

Please Notice that this section uses the “new” position for transistors 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 put with a different `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 using relative coordinates and named nodes as much as possible, so that changes in the circuit are easily done by changing keys numbers of position, and crucially, each block is reusable in other diagrams.

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

```

1 \def\normalcoord(#1){coordinate(#1)}
2 \def\showcoord(#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}](#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 sort of *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 the symbol. Using the following setting (see section 3.17.5) 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 finicky details. Without the `\killdepth` macro, the labels of different transistor 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).

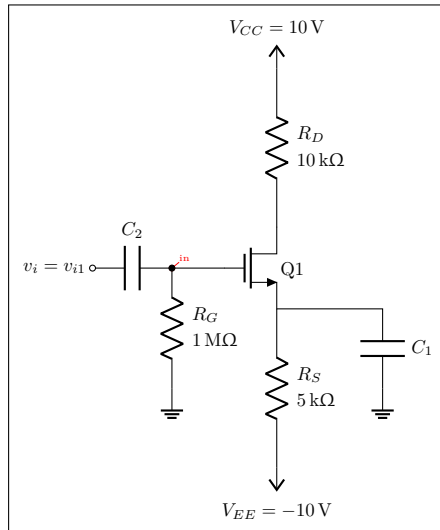


```

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, to refer to it to add the input source. Notice how the ground node is positioned: the coordinate (**in** | - **GND**) is the point with the horizontal coordinate of (**in**) and the vertical one of (**GND**), lining it up with the ground of the capacitor C_1 (you can think it as “the point on the vertical of **in** and the horizontal of **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){node[circle, red, draw, inner sep=1pt, pin={red, overlay, inner sep=0.5
10   pt, font=\tiny, pin distance=0.1cm, pin edge={red, overlay,}45:#1}]{#1}{}}
11 \begin{circuitikz}[american, ]
12 \draw (0,0) node[nmos,](Q1){\killdepth{Q1}};
13 \draw (Q1.S) to[R, l2^=$R_S$ and \SI{5}{k\ohm}] ++(0,-3) node[vee](VEE){$V_{EE}=\SI
14   {-10}{V}$}; %define VEE level
15 \draw (Q1.S) to[short] ++(2,0) to[C=$C_1$] ++(0,-1.5) node[ground](GND){};
16 \draw (Q1.G) to[short] ++(-1,0) \coord (in) to[R, l2^=$R_G$ and \SI{1}{M\ohm}] (in |-
17   GND) node[ground]{};
18 \draw (in) to[C, l_=$C_2$,*-o] ++(-1.5,0) node[left](vi1){$v_i=v_{i1}$};
19 \draw (Q1.D) to[R, l2_=$R_D$ and \SI{10}{k\ohm}] ++(0,3) node[vcc](VCC){$V_{CC}=\SI
20   {10}{V}$};
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
33   ]{};
34 \draw [densely dashed] (vo2) -- (load);

```

```

29 %
30 \draw [densely dashed] (vo1) -- (vi2);
31 %
32 \draw [blockdef] (vi1|-VEE) ++(0,-2) \coord(tmp)
33     -- node[midway, fill=white]{bloque 1} (vo1|- tmp);
34     \draw [blockdef] (vi2|-VEE) ++(0,-2) \coord(tmp)
35     -- node[midway, fill=white]{bloque 2} (vo2|- tmp);
36
37 \end{circuitikz}

```



You can see that after having found the place where we want to put the BJT transistor (line 18), 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   \tikzset{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

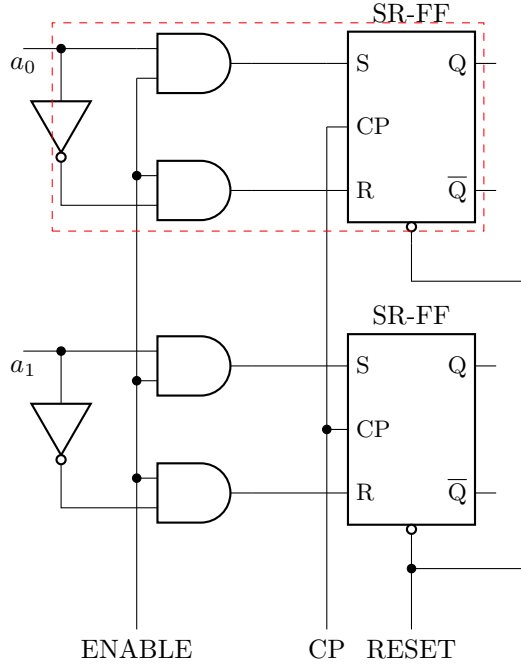
2.3 Tutorial: a logic circuit

Let's suppose we want to reproduce the circuit on the right³, 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 3.24.2); 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.

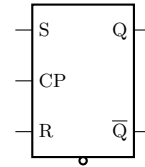
```
1 \ctikzset{
2     logic ports=ieee,
3     logic ports/scale=0.7,
4 }
```



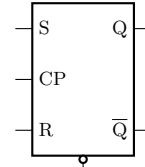
I want a reusable block, so I will start from a coordinate and then use only relative, defining coordinates along the way.

The first thing is to define a suitable flip-flop. The standard SR (see 3.25) is *almost* what we need, but not exactly the same. So let's define a new one:

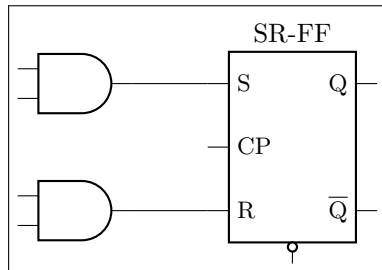
```
1 \tikzset{sr-ff/.style={flipflop, flipflop def={
2     t1=S, t2=CP, t3=R, t4={\ctikztextnot{Q}},
3     t6=Q, nd=1}},
4 }
```



If you look closer, you can notice that the new flip-flop has no lead in the bottom pin; this is due to the fact that there is no label here, and leads are drawn in flip-flops only if there is a label. This can be fixed by adding a blank label (like `td=`); otherwise you have to utilize the anchors on the internal “not” circle.



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}
```

³It seems a quite popular one on tex.stackexchange.com...

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:



```
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 version, 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 way 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]{$a_1$};

```

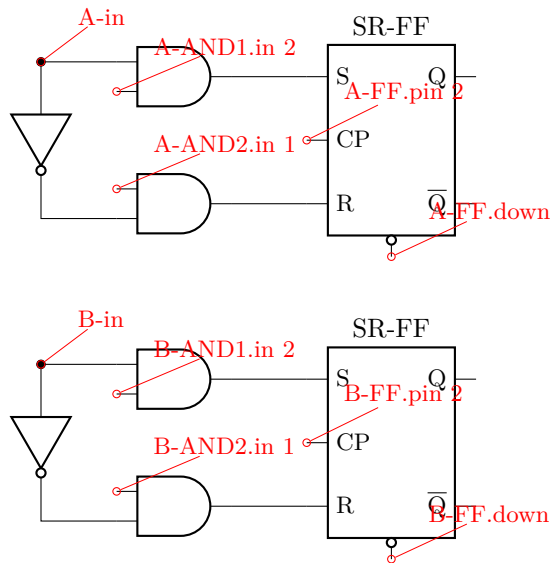
And finally:

```

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

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

3.1 Path-style components

The path-style components are used as shown below:

```

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

```

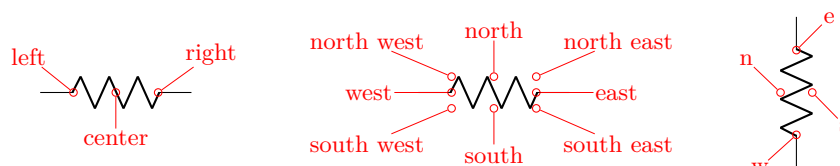
where `#1` is the name of the component, `#2` is an (optional) label, and `options` are optional labels, annotations, style specifier that will be explained in the rest of the manual.

Transistors and some other node-style components can also be placed using the syntax for bipoles. See section 3.17.8.

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

3.1.1 Anchors

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



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

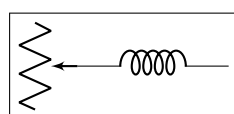


```

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}

```

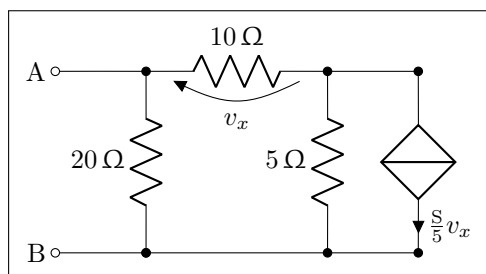
3.1.2 Customization

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.

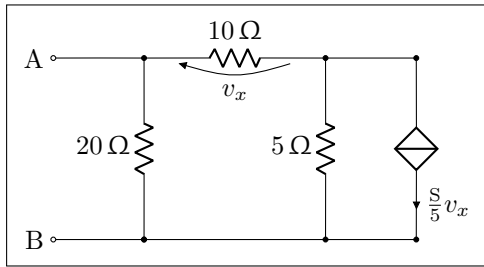
3.1.2.1 Components size Perhaps the most important parameter is `bipoles/length` (default 1.4 cm), which can be interpreted as the length of a resistor (including reasonable connections): all other lengths are relative to this value. For instance:



```

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}

```

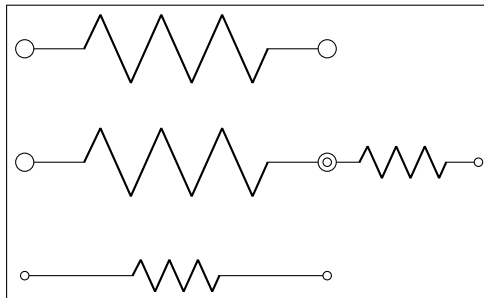


```

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=$\frac{\siemens}{5} v_x$, *-] (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 these:

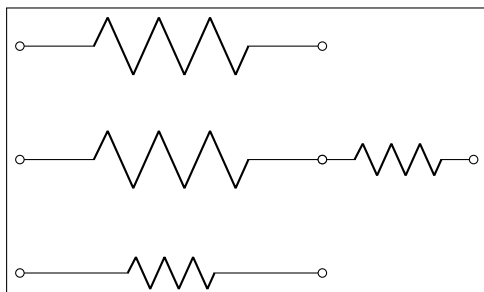


```

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:



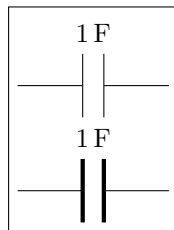
```

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}

```

3.1.2.2 Thickness of the lines (globally)

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



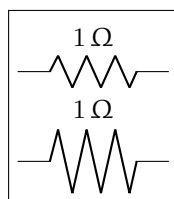
```

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);

```

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

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



```


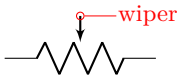
1 \tikz \draw (0,0) to[R=1<\ohm>] (2,0); \par
2 \ctikzset{bipoles/resistor/height=.6}
3 \tikz \draw (0,0) to[R=1<\ohm>] (2,0);

```

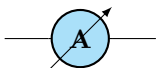
It is recommended to use the styling parameters to change the shapes; they are not so fine grained (for example, you can change the width of resistor, not the height at the moment), but they are more stable and coherent across your circuit.

3.1.3 Descriptions

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

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

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

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

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

3.2 Node-style components

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

```

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

```

Explanation of the parameters:

#1: component name⁴ (mandatory)

#2: list of comma separated options (optional)

#3: name of an anchor (optional)

#4: text written to the text anchor of the component (optional)

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

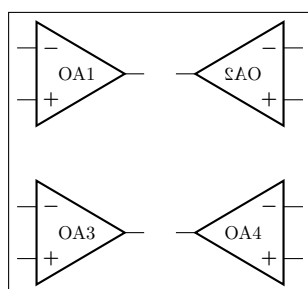
Notice: Nodes must have curly brackets at the end, even when empty. An optional anchor (#3) can be defined within round brackets to be addressed again later on. And please don’t forget the semicolon to terminate the `\draw` command.

Also notice: If using the `\tikzexternalize` feature, as of 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 this parameters affect also text labels, so they need to be un-scaled by hand.



```

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 when used in L^AT_EX 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`). In other formats they are undefined; contributions to fill the gap are welcome.

⁴For using bipoles as nodes, the name of the node is `#1shape`.



```

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

3.2.3 Descriptions

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

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

All the shapes defined by CircuiTikZ. These are all **pgf** nodes, so they are usable in both **pgf** and **TikZ**. 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`:

	<p>Plain amplifier, type: node, fillable (<code>node[plain amp]{}</code>). Class: amplifiers.</p>
--	--

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”:

	<p>Rotary switch, type: node (<code>node[rotaryswitch] (N){}</code>). Class: switches.</p>
--	---

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.

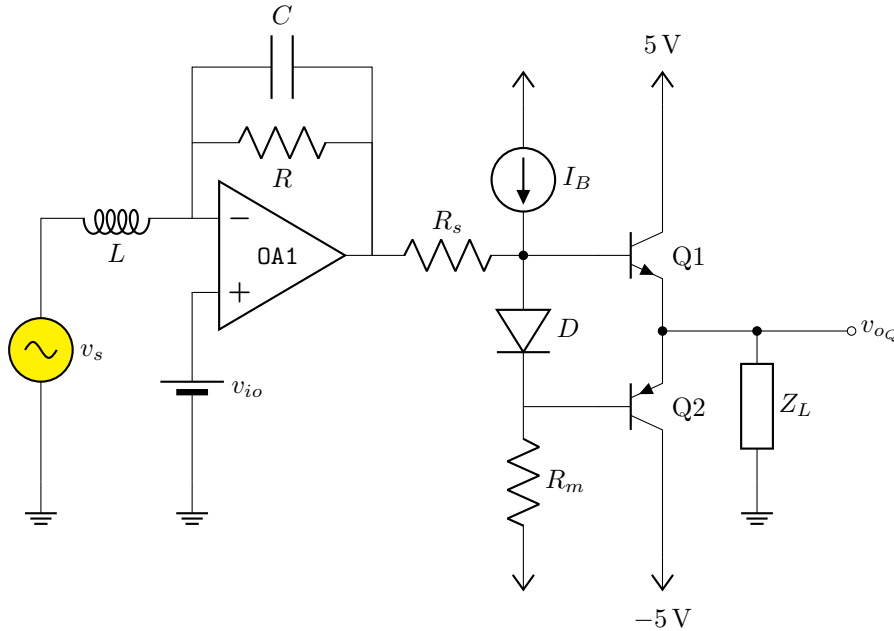
Let's see the effect over a simple circuit⁵.

```

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

```

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

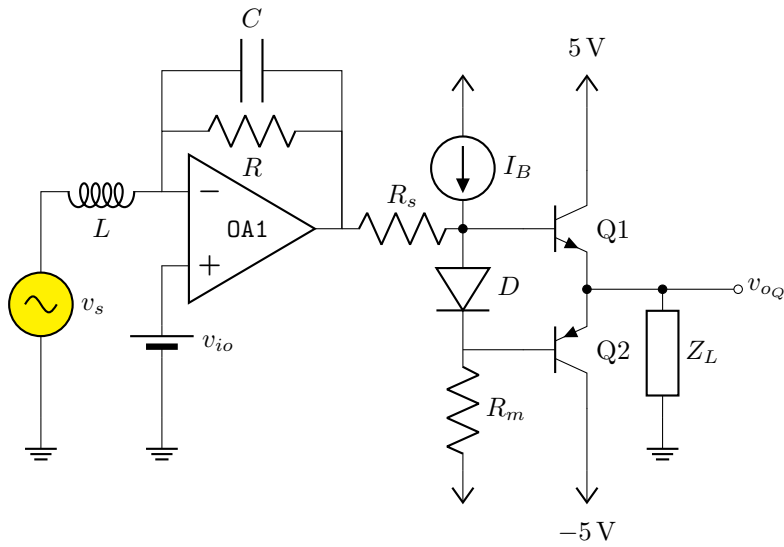


3.3.1 Relative size

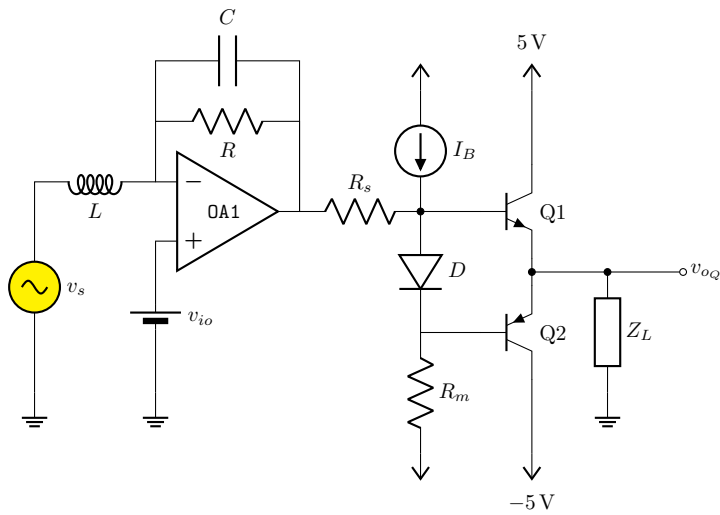
Component size can be changed globally (see section 3.1.2.1), or you can change their relative size by scaling a family of components by setting the key *class/scale*; for example, you can change the size of all the diodes in your circuit by setting *diodes/scale* to something different from the default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

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

⁵This is a just an example, the circuit is not intended to be functional.



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



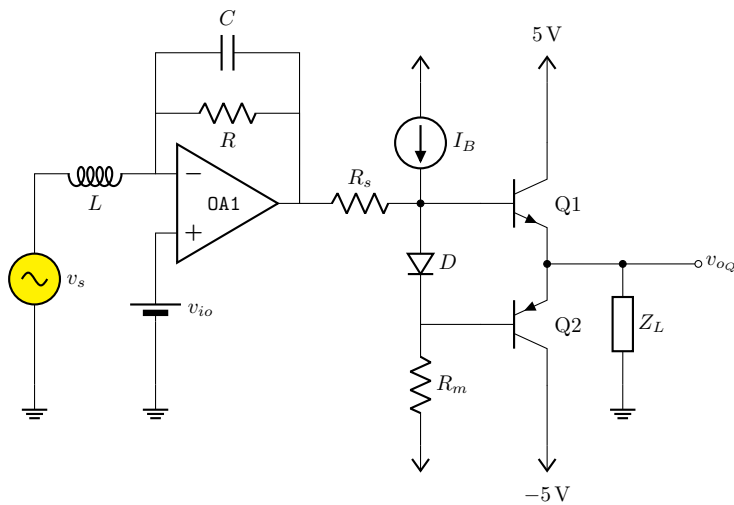
Using relative sizes as described in section 3.1.2.1 enables your style for the circuit. For example, setting:

```

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.

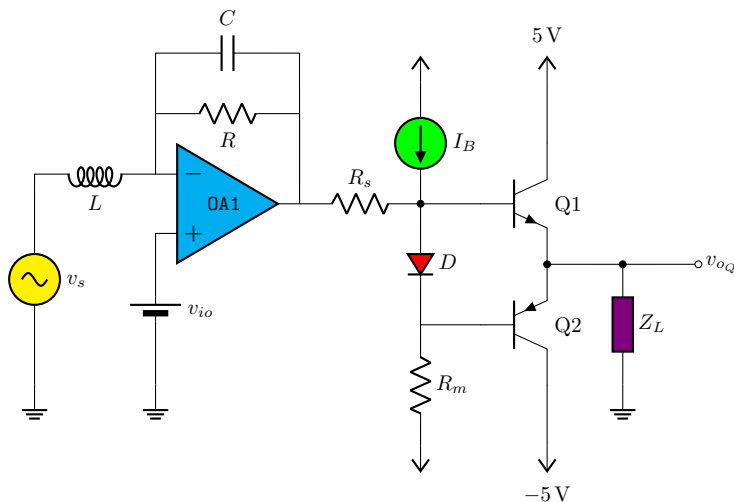
If you add to the previous styles the following commands:

```

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-color circuits can be nice, using more than that can become rapidly unbearable. Old textbooks used the two-color style quite extensively, filling

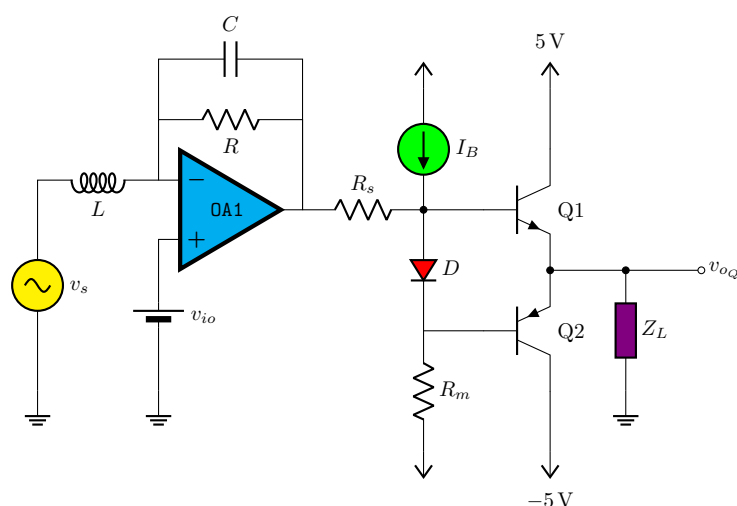
with a kind of light blue like `blue!30!white` “closed” components, but that was largely to hinder black-and-white photocopying...

3.3.3 Line thickness

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

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

```
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}
```

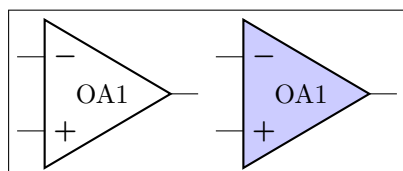
Please notice that the style is at TikZ level, not CircuiTikZ — that let’s you use it in the top option of the circuit, like:

```
\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

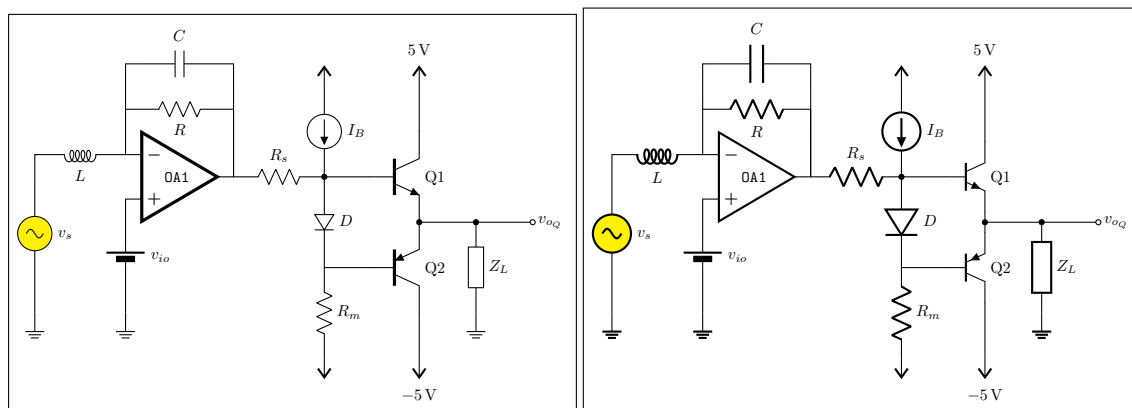
```
\ctikzsetstyle{romano}
```

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

```

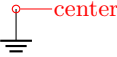
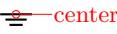








1 \ctikzloadstyle{legacy}% start from a know state
2 \tikzset{romano circuit style/.style={%
3 legacy circuit style, % load the legacy style
4 \circuitikzbasekey/.cd,%
5 % Resistors
6 resistors/scale=0.8,
7 [...]}

```

3.4 Grounds and supply voltages

3.4.1 Grounds

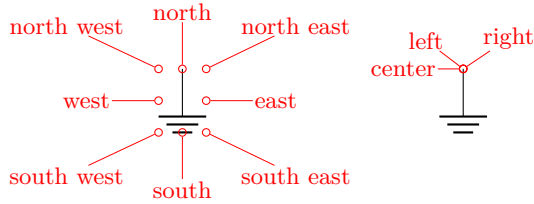
For the grounds, the `center` anchor is put on the connecting point of the symbol, so that you can use them directly in a `path` specification.

	Ground, type: node (<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 ⁶ , type: node (<code>node[cground]{}).</code> Class: grounds .
	European style ground, type: node (<code>node[eground]{}).</code> Class: grounds .
	European style ground, version 2 ⁷ , type: node (<code>node[eground2]{}).</code> Class: grounds .

⁶These last three were contributed by Luigi «Liverpool»

⁷These last two were contributed by @fotesan

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



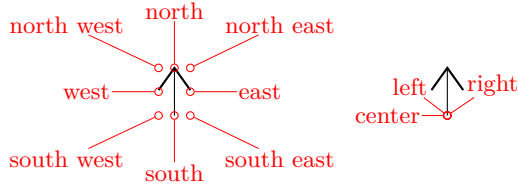
3.4.1.2 Grounds customization You can change the scale of these components (all the ground symbols together) by setting the key `grounds/scale` (default 1.0).

3.4.2 Power supplies

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

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

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



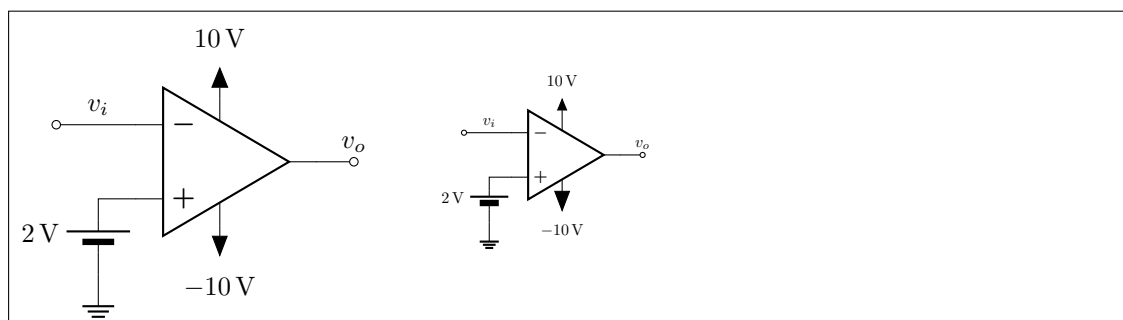
3.4.2.2 Power supplies customization You can change the scale of the power supplies by setting the key `power supplies/scale` (default 1.0).

Given that the power supply symbols are basically arrows, you can change them using all the options of the `arrows.meta` package (see the 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!

	<pre> 1 \begin{circuitikz} 2 \def\coord(#1){\showcoord(#1)<0:0.3>} 3 \draw (0,0) 4 nodevcc{VCC} \coord(vcc) ++(2,0) 5 nodevee{VEE} \coord(vee); 6 \ctikzset{monopoles/vcc/arrow={Stealth[red, width=6pt, 7 length=9pt]}} 8 \ctikzset{monopoles/vee/arrow={Latex[blue]}} 9 \draw (0,-2) 10 nodevcc{VCC} \coord(vcc) ++(2,0) 11 nodevee{VEE} \coord(vee); 12 \end{circuitikz} </pre>
--	--

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 that the arrows behave like the legacy symbols (which are shapes), *only in the arrow definitions*, you can use the special length parameter `\scaledwidth`⁸ in the arrow definition, which correspond to the width of the legacy `vcc` or `vee`. Compare the effects on the following circuit.



```

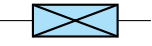
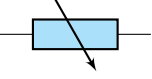


1 \ctikzset{%
2   monopoles/vcc/arrow={Triangle[width=0.8*\scaledwidth,length=\scaledwidth]},
3   monopoles/vee/arrow={Triangle[width=6pt,length=8pt]},
4 }
5 \begin{circuitikz}[baseline=(vo.center)]
6   \node [ocirc](TW) at (0,0) {};
7   \draw (TW.east) -- ++(1,0) node[midway, above]{$v_i$} node[op amp,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

```



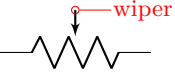
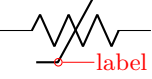
3.5 Resistive bipoles

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


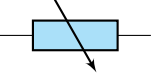
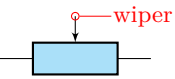
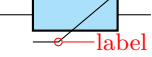
⁸Thanks to @Schrödinger's cat on [T_PX stackexchange site](#)

	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.

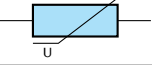

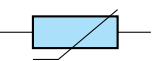
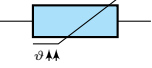

If **americanresistors** option is active (or the style [american resistors] is used; this is the default for the package), the resistors are displayed as follows:

	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.

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

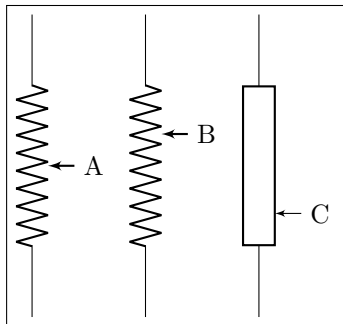
	R: Resistor, type: path-style, fillable , nodename: genericshape. Aliases: european resistor. Class: resistors.
	vR: Variable resistor, type: path-style, fillable , nodename: tgenericshape. Aliases: variable european resistor. Class: resistors.
	pR: Potentiometer, type: path-style, fillable , nodename: genericpotentiometershape. Aliases: european potentiometer. Class: resistors.
	sR: Resistive sensor, type: path-style, fillable , nodename: thermistorshape. Aliases: european resistive sensor. Class: resistors.

Other miscellaneous resistor-like devices:

	varistor : Varistor, type: path-style, fillable , nodename: varistorshape. Class: resistors.
	phR : Photoresistor, type: path-style, fillable , nodename: photoresistorshape. Aliases: photoresistor. Class: resistors.
	thR : Thermistor, type: path-style, fillable , nodename: thermistorshape. Aliases: thermistor. Class: resistors.
	thRp : PTC thermistor, type: path-style, fillable , nodename: thermistorptcshape. Aliases: thermistor ptc. Class: resistors.
	thRn : NTC thermistor, type: path-style, fillable , nodename: thermistorntcshape. Aliases: thermistor ntc. Class: resistors.

3.5.1 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`.



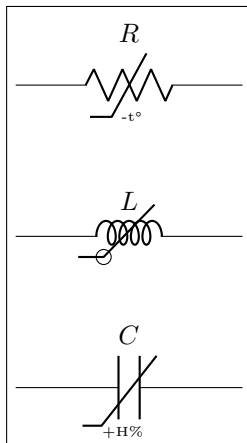
```

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}

```

3.5.2 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.
8     label) {+H\si{\%}};
9 \end{circuitikz}

```

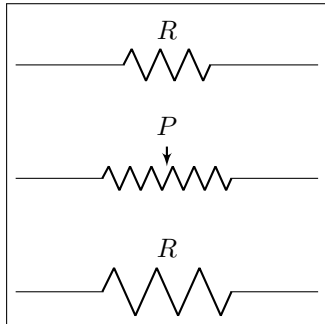
The anchor is positioned just on the corner of the segmented line crossing the component.

3.5.3 Resistive components customization

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

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

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



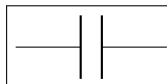
```

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}

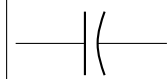
```

3.6 Capacitors and inductors: dynamical bipoles

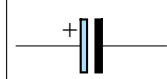
3.6.1 Capacitors



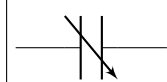
capacitor: Capacitor, type: path-style , nodename: capacitorshape. Aliases: C. Class: capacitors.



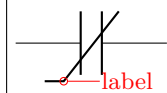
curved capacitor: Curved (polarized) capacitor, type: path-style , nodename: ccapacitorshape. Aliases: cC. Class: capacitors.



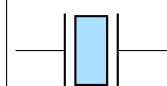
ecapacitor: Electrolytic capacitor, type: path-style, fillable , nodename: ecapacitorshape. Aliases: eC, elko. Class: capacitors.



variable capacitor: Variable capacitor, type: path-style , nodename: vcapacitorshape. Aliases: vC. Class: capacitors.



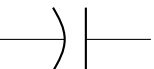
capacitive sensor: Capacitive sensor, type: path-style , nodename: capacitivesensshape. Aliases: sC. Class: capacitors.



piezoelectric: Piezoelectric Element, type: path-style, fillable , nodename: piezoelectricshape. Aliases: PZ. Class: capacitors.

There is also the (deprecated⁹ — its polarity is not coherent with the rest of the components) **polar capacitor**:

⁹Thanks to [Anshul Singhv](#) for noticing.

	polar capacitor: Polar capacitor, type: <code>path-style</code> , nodename: <code>pcapacitorshape</code> .Aliases: <code>pC</code> . Class: <code>capacitors</code> .
---	--

3.6.2 Capacitive sensors anchors


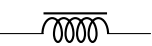


For capacitive sensors, see section [3.5.2](#).

3.6.3 Capacitors customizations




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

3.6.4 Inductors



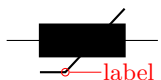
If the `cuteinductors` option is active (default behaviour), or the style `[cute inductors]` is used, the inductors are displayed as follows:

	L: Inductor, type: <code>path-style</code> , nodename: <code>cuteinductorshape</code> .Aliases: <code>cute inductor</code> . Class: <code>inductors</code> .
	cute choke: Choke, type: <code>path-style</code> , nodename: <code>cutechokeshape</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:

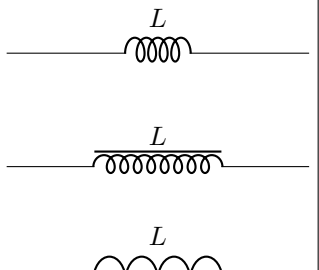
	<code>L</code> : Inductor, type: <code>path-style</code> , nodename: <code>fullgenericshape</code> . Aliases: <code>european inductor</code> . Class: <code>inductors</code> .
	<code>vL</code> : Variable inductor, type: <code>path-style</code> , nodename: <code>tfullgenericshape</code> . Aliases: <code>variable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

3.6.5 Inductors customizations

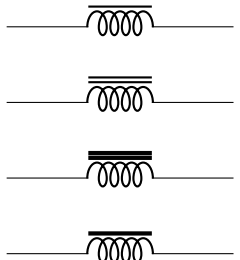
You can change the scale of the inductors by setting the key `inductors/scale` to something different from the default 1.0.

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.

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

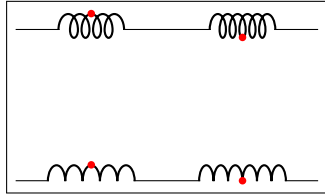
Chokes (which comes only in the `cute` style) can have single and double lines, and can have the line thickness adjust (the value is relative to the thickness of the inductor).

	<pre> 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/cthick=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} </pre>
---	---

3.6.6 Inductors anchors

For inductive sensors, see section 3.5.2.

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.midtap){};
11 \end{circuitikz}

```

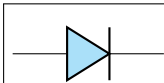
3.7 Diodes and such

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

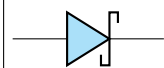
You can switch between the styles setting the key `diode` (for example `\ctikzset{diode=full}`) or `empty` or `stroke`, or with the styles `full diodes`, `empty diodes` and `stroke diodes`.

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

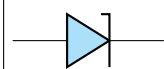
The package options `fulldiode`, `strokediode`, and `emptydiode` (and the styles `[full diodes]`, `[stroke diodes]`, and `[empty diodes]`) define which shape will be used by abbreviated commands such that `D`, `sD`, `zD`, `zzD`, `tD`, `pD`, `leD`, `VC`, `Ty`, `Tr` (no stroke symbol available!).



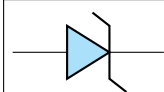
empty diode: Empty diode, type: `path-style`, fillable, nodename: `emptydiodeshape`. Aliases: `Do`. Class: `diodes`.



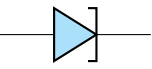
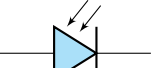
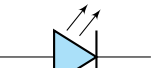
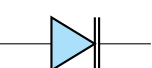
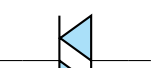









empty Schottky diode: Empty Schottky diode, type: `path-style`, fillable, nodename: `emptysdiodeshape`. Aliases: `sDo`. Class: `diodes`.



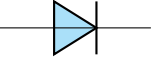
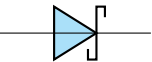
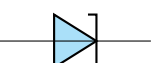
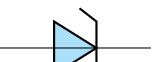
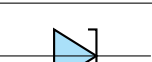



empty Zener diode: Empty Zener diode, type: `path-style`, fillable, nodename: `emptyzdiodeshape`. Aliases: `zDo`. Class: `diodes`.



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

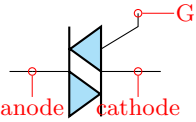
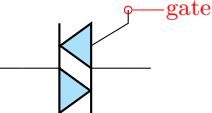
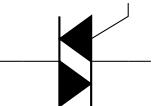
	empty tunnel diode: Empty tunnel diode, type: path-style, fillable , nodename: emptytdiodeshape. Aliases: tDo. Class: diodes.
	empty photodiode: Empty photodiode, type: path-style, fillable , nodename: emptypdiodeshape. Aliases: pDo. Class: diodes.
	empty led: Empty led, type: path-style, fillable , nodename: emptylediodeshape. Aliases: leDo. Class: diodes.
	empty varcap: Empty varcap, type: path-style, fillable , nodename: emptyvarcapshape. Aliases: VCo. Class: diodes.
	empty bidirectionaldiode: Empty bidirectionaldiode, type: path-style, fillable , nodename: emptybidirectionaldiodeshape. Aliases: biDo. Class: diodes.
	full diode: Full diode, type: path-style , nodename: fulldiodeshape. Aliases: D*. Class: diodes.
	full Schottky diode: Full Schottky diode, type: path-style , nodename: fullsdiodeshape. Aliases: sD*. Class: diodes.
	full Zener diode: Full Zener diode, type: path-style , nodename: fullzdiodeshape. Aliases: zD*. Class: diodes.
	full ZZener diode: Full ZZener diode, type: path-style , nodename: fullzzdiodeshape. Aliases: zzD*. Class: diodes.
	full tunnel diode: Full tunnel diode, type: path-style , nodename: fulltdiodeshape. Aliases: tD*. Class: diodes.
	full photodiode: Full photodiode, type: path-style , nodename: fullpdiodeshape. Aliases: pD*. Class: diodes.
	full led: Full led, type: path-style , nodename: fulllediodeshape. Aliases: leD*. Class: diodes.
	full varcap: Full varcap, type: path-style , nodename: fullvarcapshape. Aliases: VC*. Class: diodes.
	full bidirectionaldiode: Full bidirectionaldiode, type: path-style , nodename: fullbidirectionaldiodeshape. Aliases: biD*. Class: diodes.

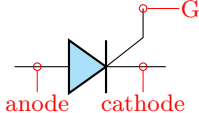
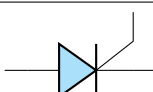

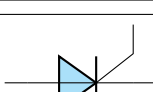
These shapes have no exact node-style counterpart, because the stroke line is built upon the empty variants:

	stroke diode: Stroke diode, type: path-style, fillable , nodename: emptydiodeshape. Aliases: D-. Class: diodes.
	stroke Schottky diode: Stroke Schottky diode, type: path-style, fillable , nodename: emptysdiodeshape. Aliases: sD-. Class: diodes.
	stroke Zener diode: Stroke Zener diode, type: path-style, fillable , nodename: emptyzdiodeshape. Aliases: zD-. Class: diodes.
	stroke ZZener diode: Stroke ZZener diode, type: path-style, fillable , nodename: emptyzzdiodeshape. Aliases: zzD-. Class: diodes.
	stroke tunnel diode: Stroke tunnel diode, type: path-style, fillable , nodename: emptytdiodeshape. Aliases: tD-. Class: diodes.
	stroke photodiode: Stroke photodiode, type: path-style, fillable , nodename: emptypdiodeshape. Aliases: pD-. Class: diodes.
	stroke led: Stroke led, type: path-style, fillable , nodename: emptylediodeshape. Aliases: leD-. Class: diodes.
	stroke varcap: Stroke varcap, type: path-style, fillable , nodename: emptyvarcapshape. Aliases: VC-. Class: diodes.

3.7.1 Tripole-like diodes

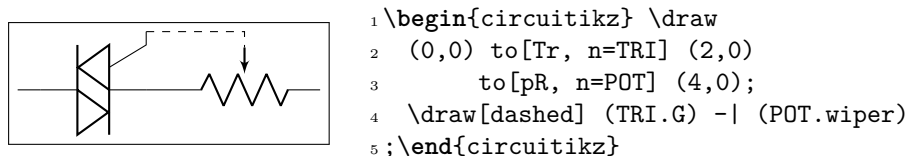
The following tripoles are entered with the usual command, of the form to[Tr, ...].

	triac: Standard triac (shape depends on package option), type: path-style, fillable , nodename: emptytriacshape. Aliases: Tr. Class: diodes.
	empty triac: Empty triac, type: path-style, fillable , nodename: emptytriacshape. Aliases: Tro. Class: diodes.
	full triac: Full triac, type: path-style , nodename: fulltriacshape. Aliases: Tr*. Class: diodes.

	thyristor : Standard thyristor (shape depends on package option), type: <code>path-style</code> , <code>fillable</code> , nodename: <code>emptythyristorshape</code> .Aliases: <code>Ty</code> . Class: <code>diodes</code> .
	empty thyristor : Empty thyristor, type: <code>path-style</code> , <code>fillable</code> , nodename: <code>emptythyristorshape</code> .Aliases: <code>Tyo</code> . Class: <code>diodes</code> .
	full thyristor : Full thyristor, type: <code>path-style</code> , nodename: <code>fullthyristorshape</code> .Aliases: <code>Ty*</code> . Class: <code>diodes</code> .
	stroke thyristor : Stroke thyristor, type: <code>path-style</code> , <code>fillable</code> , nodename: <code>emptythyristorshape</code> .Aliases: <code>Ty-</code> . Class: <code>diodes</code> .

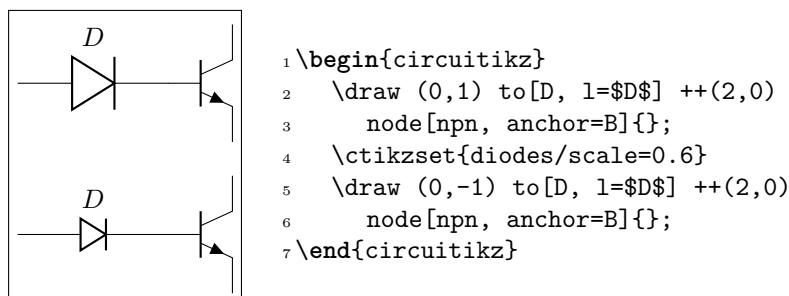
3.7.2 Triacs anchors

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:

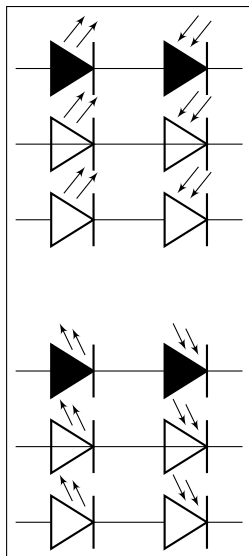


3.7.3 Diode customizations

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



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.



```

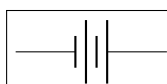
1 \begin{circuitikz}
2   \ctikzset{led arrows from anode} % default
3   \ctikzset{pd arrows to anode} % default
4   \ctikzset{full diodes}
5   \draw (0,0) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
6   \ctikzset{stroke diodes}
7   \draw (0,-1) to[leD] ++(1.5,0) to[pD] ++(1.5,0);
8   \ctikzset{empty diodes}
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}

```

3.8 Sources and generators

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

3.8.1 Batteries



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

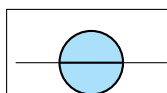


battery1: Single battery cell, type: path-style , nodename: battery1shape. Class: batteries.

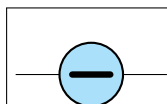


battery2: Single battery cell, type: path-style , nodename: battery2shape. Class: batteries.





3.8.2 Stationary sources



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



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



	american voltage source: Voltage source (american style), type: path-style, fillable , nodename: vsourceAMshape. Aliases: vsourceAM. Class: sources.
	european current source: Current source (european style), type: path-style, fillable , nodename: isourceshape. Aliases: isource. Class: sources.
	cute european current source: Current source (cute european style), type: path-style, fillable , nodename: isourceCshape. Aliases: isourceC, ceI. Class: sources.
	american current source: Current source (american style), type: path-style, fillable , nodename: isourceAMshape. Aliases: isourceAM. Class: sources.

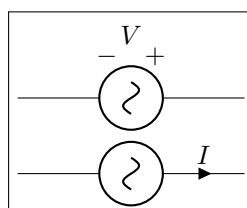
If (default behaviour) `européancurrents` option is active (or the style `[european currents]` is used), the shorthands `current source`, `isource`, and `I` are equivalent to `european current source`. Otherwise, if `americancurrents` option is active (or the style `[american currents]` is used) they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

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

3.8.3 Sinusoidal sources

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

	sinusoidal voltage source: Sinusoidal voltage source, type: path-style, fillable , nodename: vsourcesinshape. Aliases: vsourcesin, sV. Class: sources.
	sinusoidal current source: Sinusoidal current source, type: path-style, fillable , nodename: isourcesinshape. Aliases: isourcesin, sI. Class: sources.

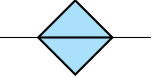

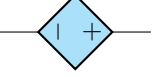
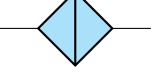

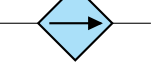



```

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}

```



3.8.4 Controlled sources

	European controlled voltage source: Controlled voltage source (European style), type: path-style, fillable, nodename: cvsourceshape. Aliases: cvsources. Class: csources.
	Cute European controlled voltage source: Voltage source (cute European style), type: path-style, fillable, nodename: cvsourcesCshape. Aliases: cvsourcesC, cceV. Class: csources.
	American controlled voltage source: Controlled voltage source (American style), type: path-style, fillable, nodename: cvsourcesAMshape. Aliases: cvsourcesAM. Class: csources.
	European controlled current source: Controlled current source (European style), type: path-style, fillable, nodename: cisourceshape. Aliases: cisources. Class: csources.
	Cute European controlled current source: Current source (cute European style), type: path-style, fillable, nodename: cisourcesCshape. Aliases: cisourcesC, cceI. Class: csources.
	American controlled current source: Controlled current source (American style), type: path-style, fillable, nodename: cisourcesAMshape. Aliases: cisourcesAM. Class: csources.
	Empty controlled source: Empty controlled source, type: path-style, fillable, nodename: ecsourceshape. Aliases: ecsources. Class: csources.

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



Similarly, if (default behaviour) `europanvoltage` 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 `americanvoltage` option is active (or the style `[american voltages]` is used) they are equivalent to `American controlled voltage source`.

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

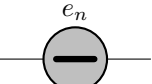
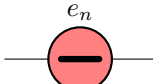
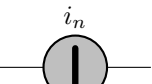
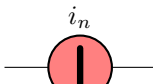
	controlled sinusoidal voltage source: Controlled sinusoidal voltage source, type: path-style, fillable, nodename: cvsourcesinshape. Aliases: controlled vsourcesin, cvsourcesin, csV. Class: csources.
	controlled sinusoidal current source: Controlled sinusoidal current source, type: path-style, fillable,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

3.8.5 Noise sources

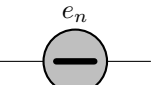
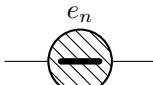
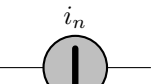
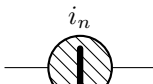
In this case, the “direction” of the source is undefined. Noise sources are filled in gray by default, but if you choose the dashed style, they become fillable.

	noise voltage source: Sinusoidal voltage source, type: path-style, nodename: vsourceNshape. Aliases: vsourceN, nV. Class: sources.
	noise current source: Sinusoidal current source, type: path-style, nodename: isourceNshape. Aliases: isourceN, nI. Class: sources.

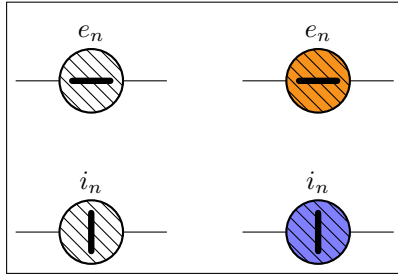
You can change the fill color with the key `circuitikz/bipoles/noise sources/fillcolor`:

		<pre>1 \begin{circuitikz} 2 \draw(0,0) to [nV, l=\$e_n\$] ++(2,0); 3 \draw(0,-2) to [nI, l=\$i_n\$] ++(2,0); 4 \begin{scope}[circuitikz/bipoles/noise 5 sources/fillcolor=red!50] 6 \draw(3,0) to [nV, l=\$e_n\$] ++(2,0); 7 \draw(3,-2) to [nI, l=\$i_n\$] ++(2,0); 8 \end{scope} 9 \end{circuitikz}</pre>
		

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

		<pre>1 \begin{circuitikz} 2 \draw(0,0) to [nV, l=\$e_n\$] ++(2,0); 3 \draw(0,-2) to [nI, l=\$i_n\$] ++(2,0); 4 \begin{scope}[circuitikz/bipoles/noise 5 sources/fillcolor=dashed] 6 \draw(3,0) to [nV, l=\$e_n\$] ++(2,0); 7 \draw(3,-2) to [nI, l=\$i_n\$] ++(2,0); 8 \end{scope} 9 \end{circuitikz}</pre>
		

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

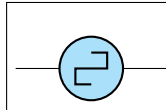


```

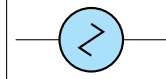
1 \begin{circuitikz}
2   \ctikzset{bipoles/noise sources/fillcolor=
   dashed}
3   \draw(0,0) to [nV, l=$e_n$] ++(2,0);
4   \draw(0,-2) to [nI, l=$i_n$] ++(2,0);
5   \begin{scope}
6     \draw(3,0) to [nV, l=$e_n$, fill=yellow
7       !50!red] ++(2,0);
8     \draw(3,-2) to [nI, l=$i_n$, fill=blue
9       !50!white] ++(2,0);
10  \end{scope}
11 \end{circuitikz}

```

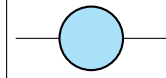
3.8.6 Special sources



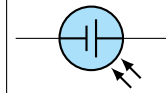
square voltage source: Square voltage source, type: path-style, fillable , nodename: vsourcesquashape. Aliases: vsourcesquare, sqV. Class: sources.



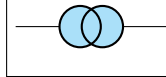
vsourcetri: Triangle voltage source, type: path-style, fillable , nodename: vsourcetrishape. Aliases: tV. Class: sources.



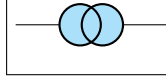
esource: Empty voltage source, type: path-style, fillable , nodename: esourceshape. Class: sources.



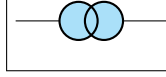
pvsources: Photovoltaic-voltage source, type: path-style, fillable , nodename: pvsourceshape. Class: sources.



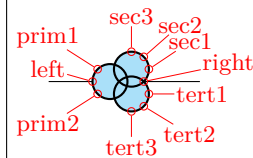
ioosources: Double Zero style current source, type: path-style, fillable , nodename: oosourceshape. Class: sources.



voosources: Double Zero style voltage source, type: path-style, fillable , nodename: oosourceshape. Class: sources.

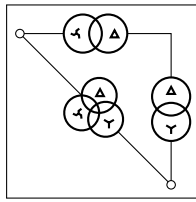


oosourcetrans: transformer source, type: path-style, fillable , nodename: oosourcetransshape. Class: sources.



ooosources: transformer with three windings, type: path-style, fillable , nodename: ooosourceshape. Class: sources.

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

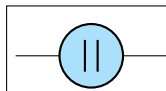


```

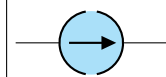
1 \begin{circuitikz}
2   \draw (0,0) to[oosourcetrans,prim=zig,sec=delta,o-] ++(2,0)
3     to[oosourcetrans, prim=delta, sec=wye,-o] ++(0,-2)
4     to[ooosource, prim=wye,sec=zig,tert=delta] (0,0);
5 \end{circuitikz}

```

3.8.7 DC sources

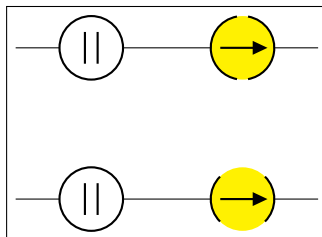


dcvsource: DC voltage source, type: `path-style`, fillable, nodename: `dcvsource`shape. Class: `sources`.



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}

```

3.8.8 Sources customizations

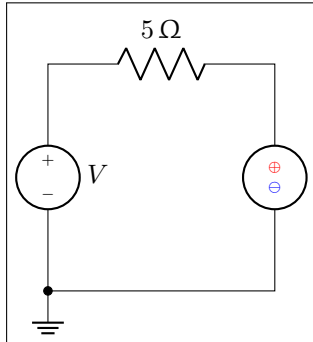
You can change the scale of the batteries by setting the key `batteries/scale`, for the controlled (dependent) sources with `csources/scale`, and for all the other independent sources and generators with `sources/scale`, to something different from the default 1.0.

The symbols drawn into the `american voltage source`¹⁰ can be changed by using the `\ctikzset` keys `bipoles/vsourceam/inner plus` and `bipoles/vsourceam/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).

Moreover, you can move the two symbols 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 `cvsourceam` to `vsourceam` (notice the initial “c”).

¹⁰Since version 1.1.0, thanks to the suggestions and discussion [in this TeX.SX question](#).



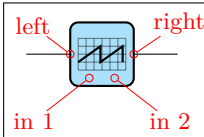
```

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}

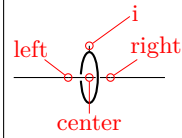
```

3.9 Instruments

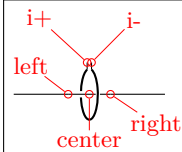
	ammeter: Ammeter, type: path-style, fillable , nodename: ammetershape. Class: instruments.
	voltmeter: Voltmeter, type: path-style, fillable , nodename: voltmetershape. Class: instruments.
	ohmmeter: Ohmmeter, type: path-style, fillable , nodename: ohmmetershape. Class: instruments.
	rmeter: Round meter (use t=... for the symbol), type: path-style, fillable , nodename: rmetershape. Class: instruments.
	rmeterwa: Round meter with arrow (use t=... for the symbol), type: path-style, fillable , nodename: rmeterwashape. Class: instruments.
	smeter: Square meter (use t=... for the symbol), type: path-style, fillable , nodename: smetershape. Class: instruments.
	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: Oscilloscope¹¹, type: path-style, fillable ,
nodename: oscopeshape. Class: instruments.



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



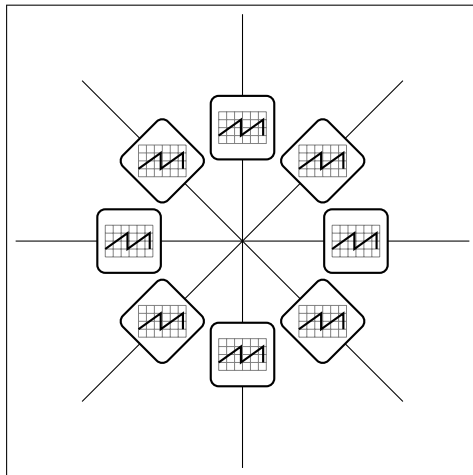
iloop2: Current loop (real), type: path-style ,
nodename: iloop2shape. Class: instruments.

3.9.1 Instruments customizations

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

3.9.2 Rotation-invariant elements

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

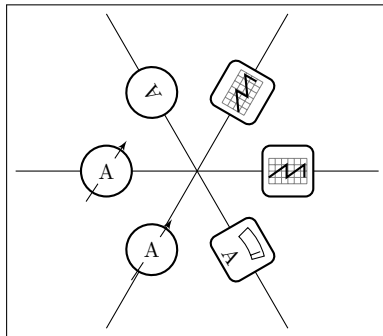


```
1 \begin{circuitikz}
2   \foreach \a in {0,45,...,350} {
3     \draw (0,0) to[oscope] (\a:3);
4   }
5 \end{circuitikz}
```

The `rmeter`, `rmaterwa`, 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`).

¹¹Suggested by @nobr1 on GitHub



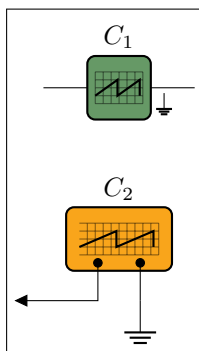
```

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}

```

3.9.3 Instruments as node elements

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



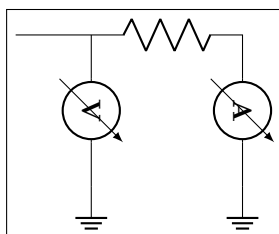
```

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[oscoshape, 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}

```

3.9.4 Measuring voltage and currents, multiple ways

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

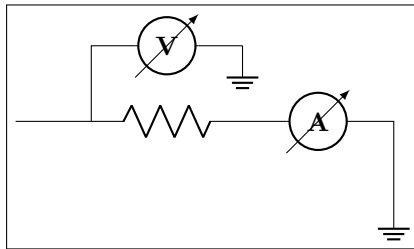


```

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.

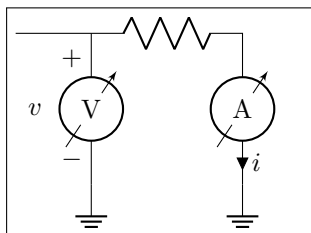


```

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).

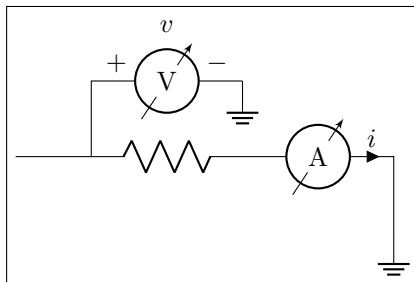


```

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:

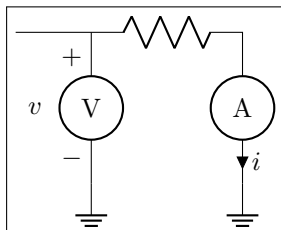


```

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:

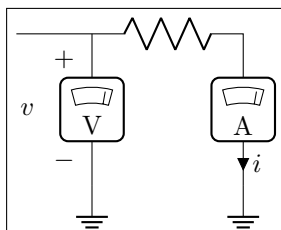


```

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:

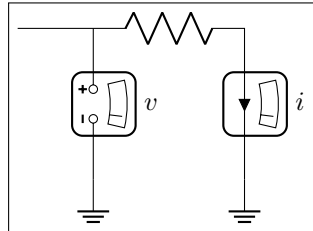


```

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 QUCS¹²-style probes, which have the nice property of explicitly showing the type of connection (in series or parallel) of the meter:

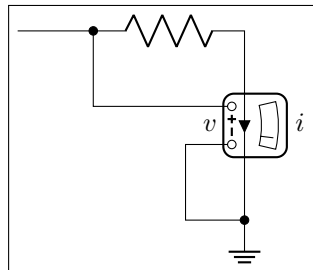


```

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-` :

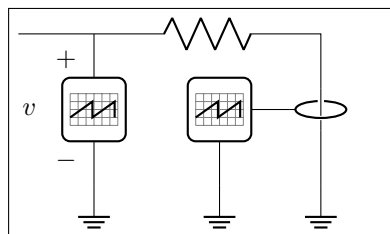


```

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,
8     anchor=right, name=0]{};
9   \draw (0.south) -- (0.south |- GND) node[
10    ground]{};
11 \end{circuitikz}

```

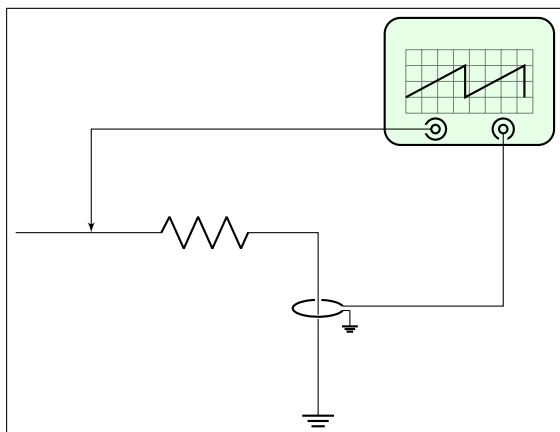
Or, if you want a more physical structure for the measurement setup:

¹²QUCS is an open source circuit simulator: <http://qucs.sourceforge.net/>

```

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}

```



3.10 Mechanical Analogy

	damper: Mechanical Damping, type: path-style, fillable , nodename: dampershape. Class: mechanicals.
	inerter: Mechanical Inerter, type: path-style, fillable , nodename: inetershape. Class: mechanicals.
	spring: Mechanical Stiffness, type: path-style , nodename: springshape. Class: mechanicals.
	viscoe: Mechanical viscoelastic element ¹³ , type: path-style, fillable , nodename: viscoeshape. Class: mechanicals.
	mass: Mechanical Mass, type: path-style, fillable , nodename: massshape. Class: mechanicals.




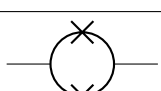
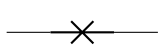
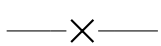
3.10.1 Mechanical elements customizations

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



¹³Suggested by @Alex in <https://tex.stackexchange.com/q/484268/38080>

3.11 Miscellaneous bipoles

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


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

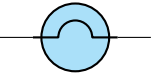
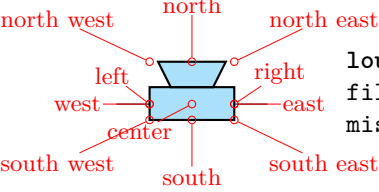
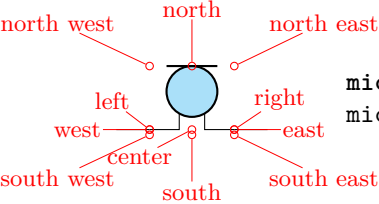
You can tune how big is the gap in the **openbarrier** component by setting the key **bipoles/openbarrier/gap** (default value 0.5; 0 means no gap and 1 full gap).

	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.

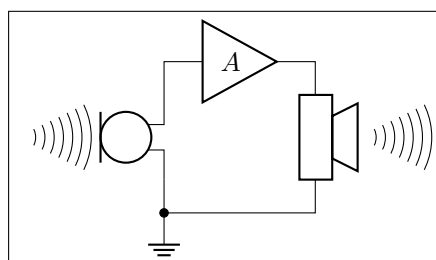
If (default behaviour) **europeangfsurgearrester** option is active (or the style **[european gas filled surge arrester]** is used), the shorthands **gas filled surge arrester** and **gf surge arrester** are equivalent to the european version of the component.

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.

You can use microphones and loudspeakers with **waves** (see section 3.19) too:



```

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

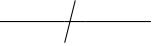
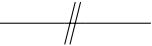
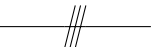
```

3.11.1 Miscellaneous element customization

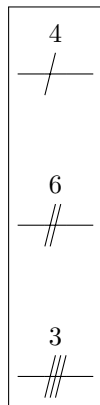
You can change the scale of all the miscellaneous elements by setting the key `misc/scale` to something different from the default 1.0.

3.12 Multiple wires (buses)

This 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 ¹⁴ , type: path-style , nodename: tmultiwireshape. Aliases: tmultiwire. Class: default.

¹⁴added by offline



```

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}

```

3.13 Crossings

Path style:

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

Node style:

	Jumper-style crossing node, type: node (<code>node[jump crossing]</code>). No class.
	Plain style crossing node, type: node (<code>node[plain crossing]</code>). 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.

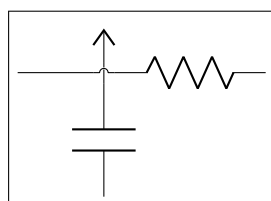
	<pre> 1 \begin{circuitikz}[] 2 \draw(1,-1) to[short] (1,1) 3 (0,0) to[short] (2,0); 4 \draw(4,-1) to[short] (4,1) 5 (3,0) to[short] (5,0) 6 (4,0) node[circ]{}; 7 \end{circuitikz} </pre>
--	---

However, sometime 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] 3 (2,0); 4 \draw(4,-1) to[short] (4,1) (3,0) to[short] 5 (5,0) 6 (4,0) node[circ]{}; 7 \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 sometime a bit of trial and error is needed to position it.

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



```

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}


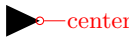
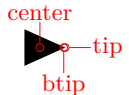
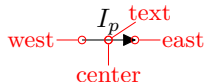
```

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

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

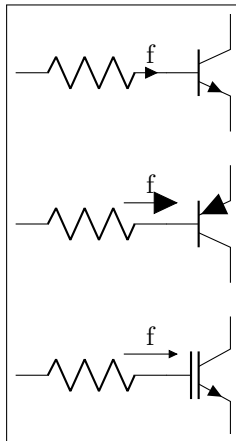
3.14 Arrows

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.

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

3.14.1 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.



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, i=f] ++(2,0) node[npn, anchor=B]{};
3   \draw (0,-2) to[R, f=f, current arrow scale=8] ++(2,0)
4     node[pnp, anchor=B, circuitikz/current arrow scale
5       =8]{};
6   \draw (0,-4) to[R, f=f, current arrow scale=24] ++(2,0)
7     node[nigt, anchor=B]{};
8 \end{circuitikz}

```

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

```

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

```

3.15 Terminal shapes

These are the so-called “bipole nodes” shapes, or poles (see section 5.1). These nodes are always filled; the “open” versions (starting with an `o`) are by default filled white, but you can override it with the `fill` parameter.

•	Connected terminal, type: node (<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}

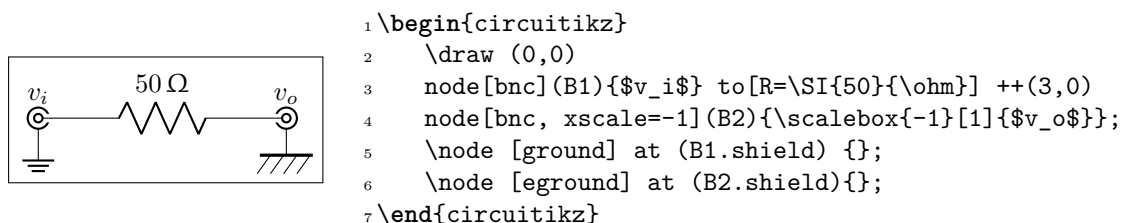
```

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 5.1 for more usage and configurability.

3.15.1 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):

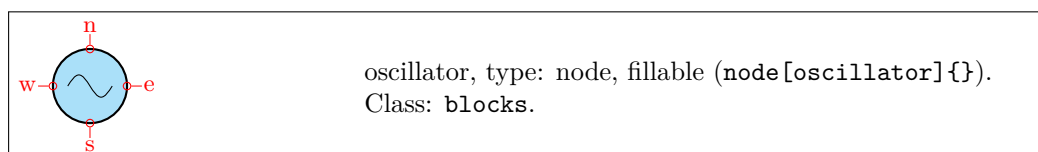
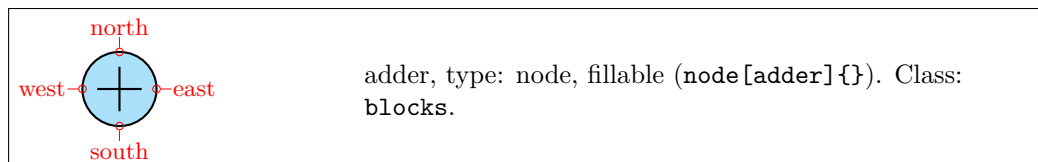
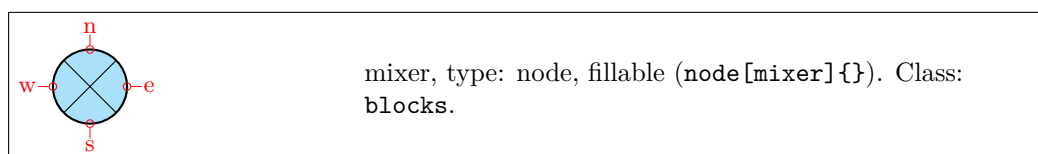


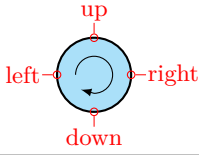
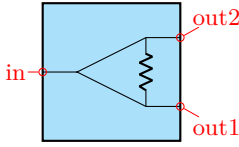
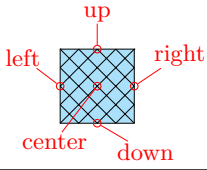
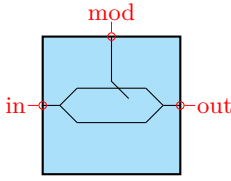
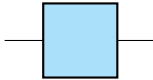
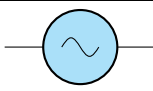
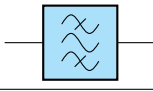
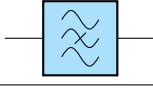



It also has a **zero** anchor if you need to rotate it about its real center.



3.16 Block diagram components

Contributed by Stefan Erhardt.


















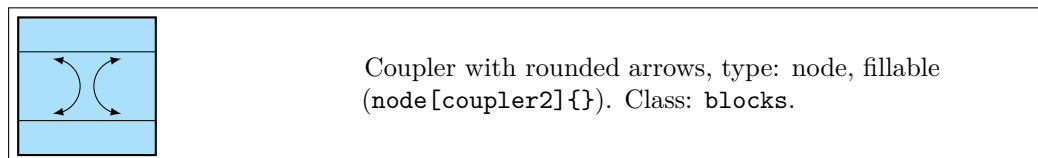
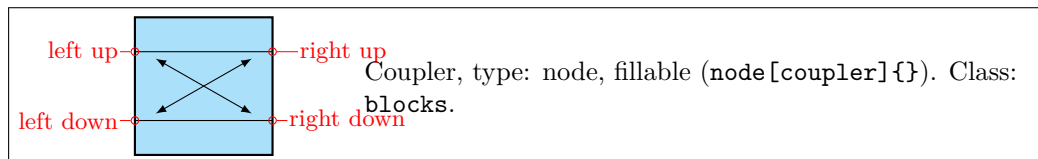
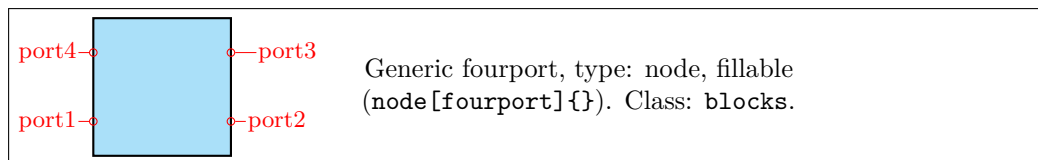
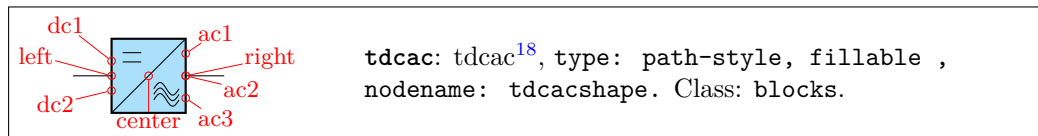
	circulator, type: node, fillable (node[circulator]{}). Class: blocks.
	wilkinson divider, type: node, fillable (node[wilkinson]{}). Class: blocks.
	gridnode ¹⁵ , type: node, fillable (node[gridnode]{}). Class: blocks.
	Mach Zehnder Modulator ¹⁶ , type: node, fillable (node[mzm]{}). Class: blocks.
	twoport: generic two port ¹⁷ , type: path-style, fillable , nodename: twoportshape. Class: blocks.
	vco: vco, type: path-style, fillable , nodename: vcoshape. Class: blocks.
	bandpass: bandpass, type: path-style, fillable , nodename: bandpassshape. Class: blocks.
	bandstop: bandstop, type: path-style, fillable , nodename: bandstopshape. Class: blocks.
	highpass: highpass, type: path-style, fillable , nodename: highpassshape. Class: blocks.
	lowpass: lowpass, type: path-style, fillable , nodename: lowpassshape. Class: blocks.
	adc: A/D converter, type: path-style, fillable , nodename: adcshape. Class: blocks.

¹⁵added by olflne

¹⁶added by dl1chb

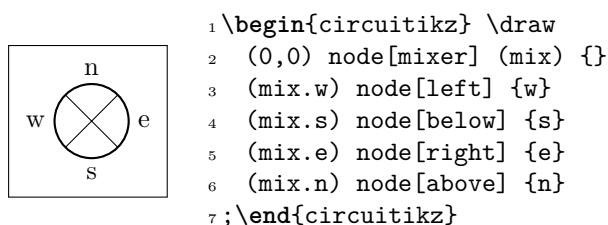
¹⁷To specify text to be put in the component: twoport[t=text]): 

	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.
	sacdc: sacdc, type: path-style, fillable , nodename: sacdcshape. Class: blocks.
	sdcac: sdcac, type: path-style, fillable , nodename: sdcacshape. Class: blocks.
	tacdc: tacdc, type: path-style, fillable , nodename: tacdcshape. Class: blocks.

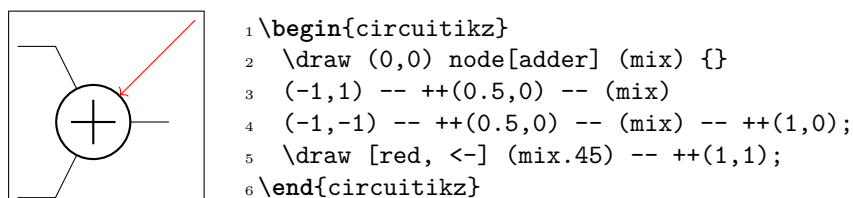


3.16.1 Blocks anchors

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

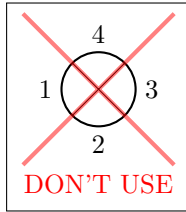


Moreover, they have proper border anchors since version 1.2.3, so you can do things like this:



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.

¹⁸the 4 converter blocks added by `olflite`

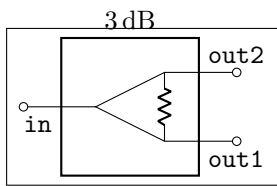


```

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:

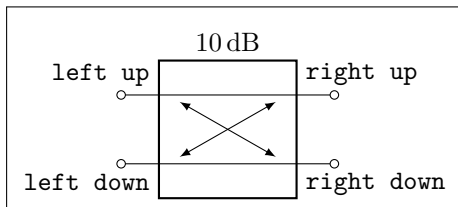


```

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:

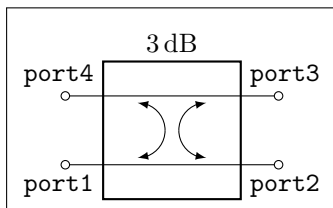


```

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
8     down}}
9   (c.right down) node[below right] {\texttt{right
10    down}}
11  (c.right up) node[above right] {\texttt{right
12    up}}
13  (c.left up) node[above left] {\texttt{left up}}
14  ;
15 \end{circuitikz}

```

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

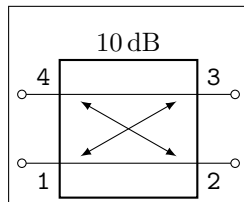


```

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.



```

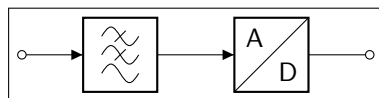
1 \begin{circuitikz} \draw(0,1.5) %bounding box
2 (0,0) node[coupler] (c) {\SI{10}{dB}}
3 (c.1) to[short,-o] ++(-0.5,0)
4 (c.2) to[short,-o] ++(0.5,0)
5 (c.3) to[short,-o] ++(0.5,0)
6 (c.4) to[short,-o] ++(-0.5,0)
7 (c.1) node[below left] {\texttt{1}}
8 (c.2) node[below right] {\texttt{2}}
9 (c.3) node[above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

```

3.16.2 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.

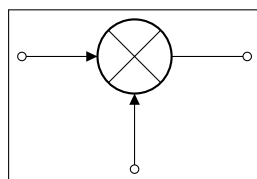


```

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}

```

3.16.2.1 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:

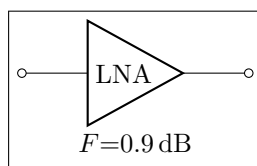


```

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

```

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

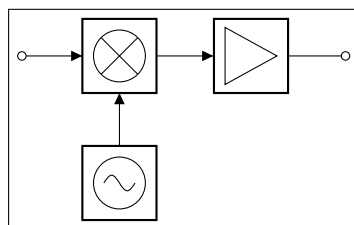


```

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

```

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

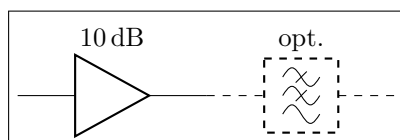


```

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

```

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



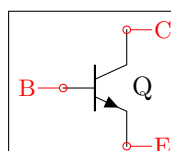
```

1 \begin{circuitikz}
2 \draw (0,0) to[amp,l=\SI{10}{dB}] ++(2.5,0);
3 \draw[dashed] (2.5,0) to[lowpass,l=opt.]
4 ++(2.5,0);
\end{circuitikz}

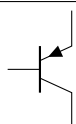
```

3.17 Transistors

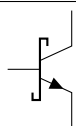
3.17.1 Standard bipolar transistors



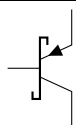
nnp, type: node (node[npn]{Q}). Class: transistors.



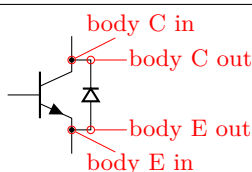
pnp, type: node (node[pnp]{}). Class: transistors.



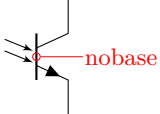
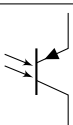
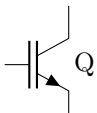
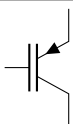
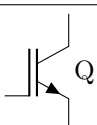

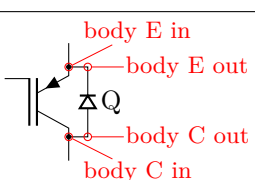
schottky npn, type: node (node[npn, schottky base]{}). Class: transistors.



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



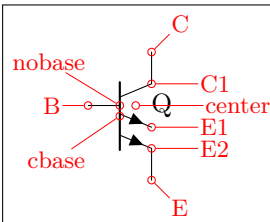
nnp, type: node (node[npn, bodydiode]{}). Class: transistors.

	photo npn, type: node (node[npn,photo]{}). Class: transistors.
	photo pnp, type: node (node[pnp,photo]{}). Class: transistors.
	night, type: node (node[night]{Q}). Class: transistors.
	pigbt, type: node (node[pigbt]{}). Class: transistors.
	Lnight, type: node (node[Lnight]{Q}). Class: transistors.
	Lpigbt, type: node (node[Lpigbt]{}). Class: transistors.
	Lpigbt, type: node (node[Lpigbt, bodydiode]{Q}). Class: transistors.

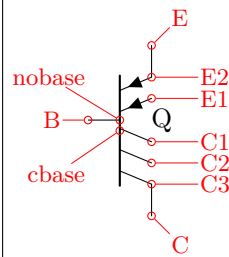
3.17.2 Multi-terminal bipolar transistors

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 `nnp` 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.

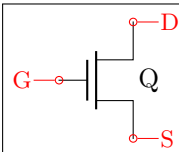


bjt npn, type: node (node[bjtnpn, collectors=1, emitters=2]{Q}). Class: transistors.

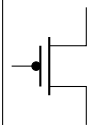


bjt pnp, type: node (node[bjtpnp, collectors=3, emitters=2]{Q}). Class: transistors.

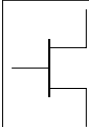
3.17.3 Field-effect transistors



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

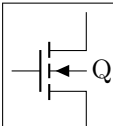


pmos, type: node (node[pmos]{}). Class: transistors.

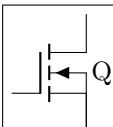


hemt, type: node (node[hemt]{}). Class: transistors.

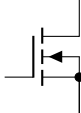
NFETs and PFETs have been incorporated based on code provided by Clemens Helfmeier and Theodor Borsche. Use the package options `fetsolderdot`/`nofetsolderdot` to enable/disable solderdot at some fet-transistors. Additionally, the solderdot option can be enabled/disabled for single transistors with the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

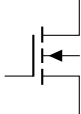


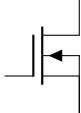
nfet, type: node (node[nfet]{Q}). Class: transistors.

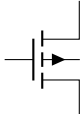


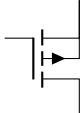
nigfete, type: node (node[nigfete]{Q}). Class: transistors.

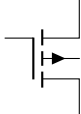
	nigfete, type: node (node[nigfete,solderdot]{}). Class: transistors .
---	---

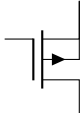
	nigfetebulk, type: node (node[nigfetebulk]{}). Class: transistors .
---	--

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

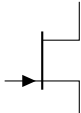
	pfet, type: node (node[pfet]{Q}). Class: transistors .
---	---

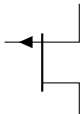
	pigfete, type: node (node[pigfete]{}). Class: transistors .
--	--

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

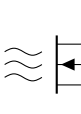
	pigfetd, type: node (node[pigfetd]{}). Class: transistors .
---	--

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

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

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

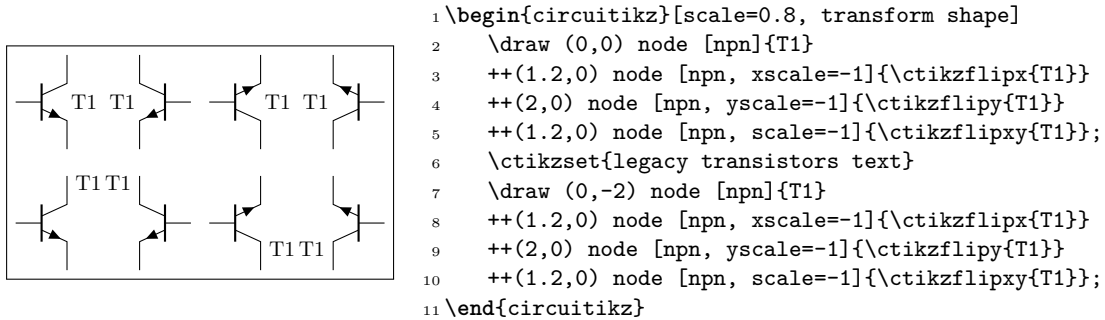
ISFET

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

3.17.4 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`.

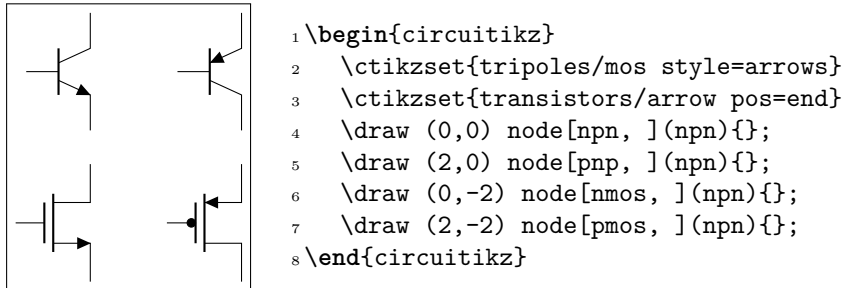
Notice the use of the utility functions `\ctikzflip{x,y,xy}` as explained in section 3.2.1.



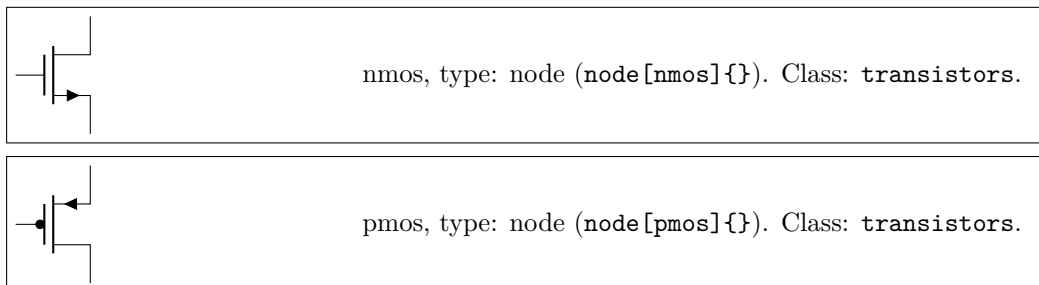
3.17.5 Transistors customization

3.17.5.1 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 `currarrow` (see section 3.14.1) and the dots on P-type transistors (if any) are the same as the nodes/poles (see section 5.1).

3.17.5.2 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`).

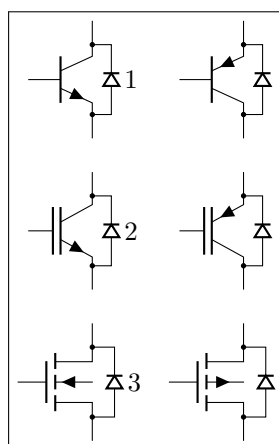


If the option `arrowmos` is used (or after the command `\ctikzset{tripoles/mos style/arrows}` is given), this is the output:



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

3.17.5.3 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):

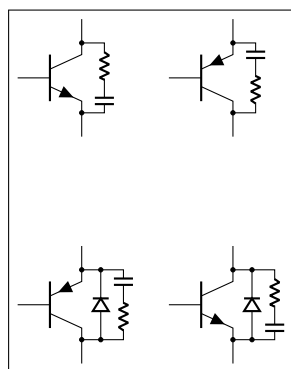


```

1 \begin{circuitikz}
2   \draw (0,0) node[npn,bodydiode] (npn){1}
3     ++(2,0) node[pnp,bodydiode] (pnp){};
4   \draw (0,-2) node[nigt,bodydiode] (nigt){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}

```

You can use the `body ...` anchors to add more or different things to the transistors in addition (or instead) of the flyback diode.

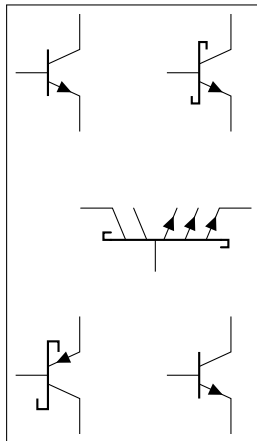


```

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}

```

3.17.5.4 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 0.07; the unit is the standard resistor length, scaled if needed.)

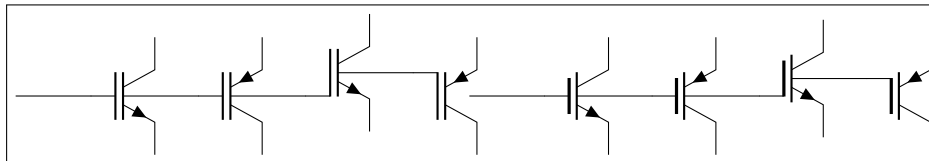


```

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}

```

3.17.5.5 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.

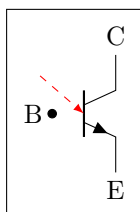


```

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}

```

3.17.5.6 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:

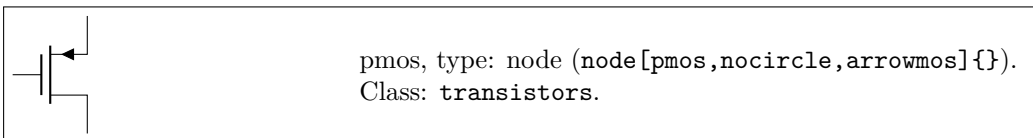
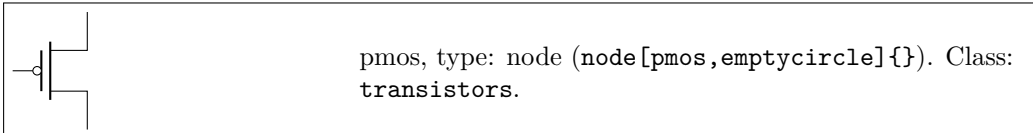


```

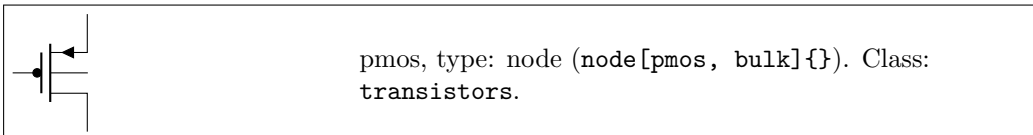
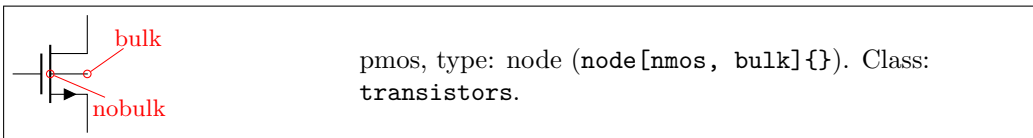
1 \begin{circuitikz}
2   \draw (2,0) node[npn,nobase] (npn){};
3   \draw (npn.E) node[below]{E};
4   \draw (npn.C) node[above]{C};
5   \draw (npn.B) node[circ]{} node[left]{B};
6   \draw[dashed,red,-latex] (1,0.5)--(npn.nobase);
7 \end{circuitikz}

```

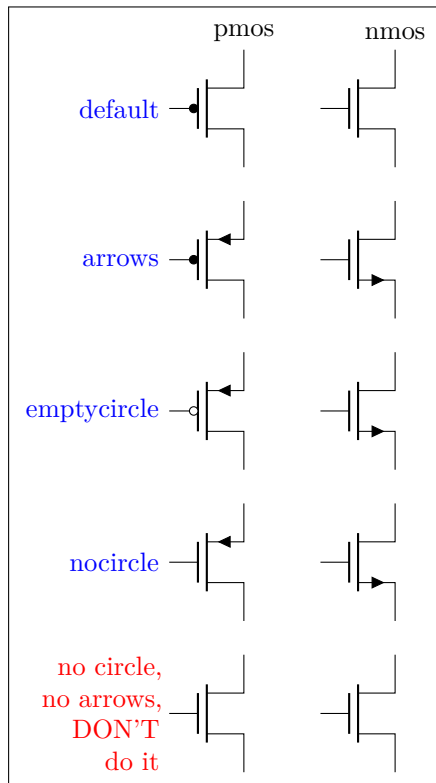
To draw the PMOS circle non-solid, use the option `emptycircle` or the command `\ctikzset{tripoles/pmos style/emptycircle}`. To remove the dot completely (only useful if you have `arrowmos` enabled, otherwise there will be no difference between P-MOS and N-MOS), you can use the option `nocircle` or `\ctikzset{tripoles/pmos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



3.17.5.7 Bulk terminals. You can add a bulk terminal¹⁹ 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):



¹⁹Thanks to Burak Kelleci <kellecib@hotmail.com>.



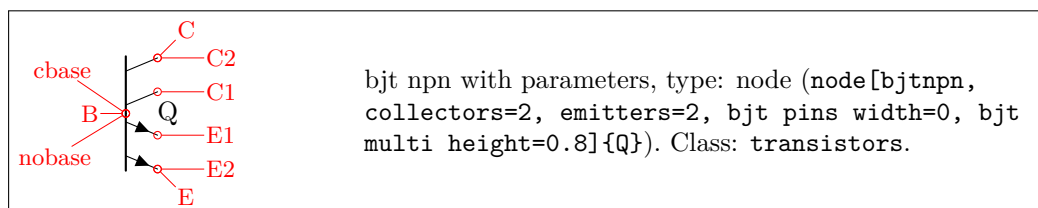
```

1 \begin{circuitikz}[
2   info/.style={left=1cm, blue, text width=5
3     em, 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[
12    pmos]{} (2,-4) node[nmos]{};
13  \ctikzset{tripoles/pmos style/nocircle}
14  \draw (0,-6) node[info]{nocircle} node[
15    pmos]{} (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}

```

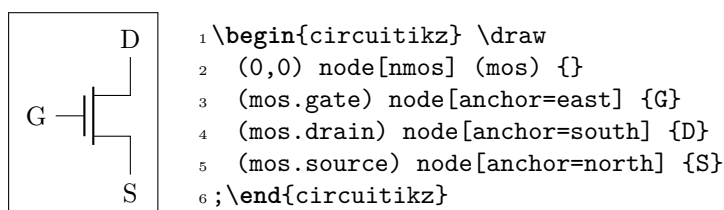
3.17.6 Multiple terminal transistors customization

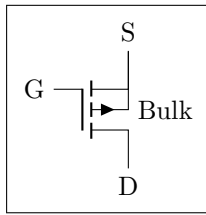
You can create completely “bare” transistors (without the connection leads to the B, C y E terminals), by changing the parameter `tripoles/bjt/pins width` (default 0.3; it is expressed as a fraction of the basic (scaled) length) or using the style `bjt pins width`; and you can change the distance between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



3.17.7 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):



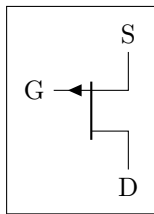


```

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

```

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

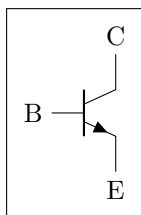


```

1 \begin{circuitikz} \draw
2   (0,0) node[pjfet] (pjfet) {}
3   (pjfet.G) node[anchor=east] {G}
4   (pjfet.D) node[anchor=north] {D}
5   (pjfet.S) node[anchor=south] {S}
6 ;\end{circuitikz}

```

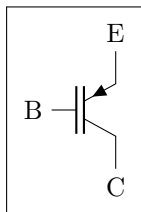
For NPN, PNP, NIGBT and PIGBT transistors, the anchors are **base**, **emitter** and **collector** anchors (which can be abbreviated with **B**, **E** and **C**):



```

1 \begin{circuitikz} \draw
2   (0,0) node[npn] (npn) {}
3   (npn.base) node[anchor=east] {B}
4   (npn.collector) node[anchor=south] {C}
5   (npn.emitter) node[anchor=north] {E}
6 ;\end{circuitikz}

```

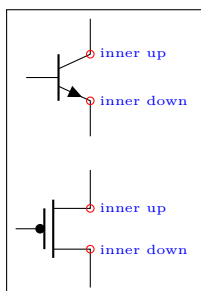


```

1 \begin{circuitikz} \draw
2   (0,0) node[pigbt] (pigbt) {}
3   (pigbt.B) node[anchor=east] {B}
4   (pigbt.C) node[anchor=north] {C}
5   (pigbt.E) node[anchor=south] {E}
6 ;\end{circuitikz}

```

Finally, 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:

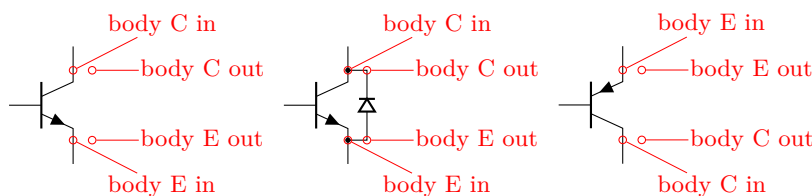


```

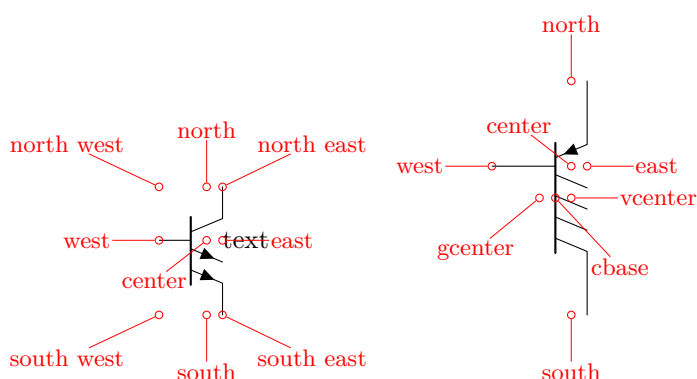
1 \begin{circuitikz}
2   \node [npn] (A) at(0,2) {};
3   \node [pmos] (B) at(0,0) {};
4   \foreach \e in {A, B}
5     \foreach \a in {inner up, inner down} {
6       \node[red, circle, inner sep=1pt, draw]
7         at (\e.\a) {};
8       \node [right, font=\tiny, blue]
9         at (\e.\a) {\a};
10    }
11 \end{circuitikz}

```

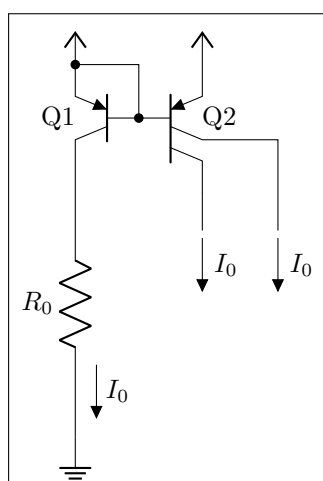
Additionally, you can access the position for the flyback diodes and possibly snubbers as shown in 3.17.5.3.



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 than 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.

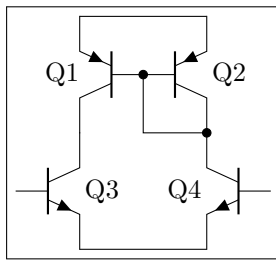


```

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`):



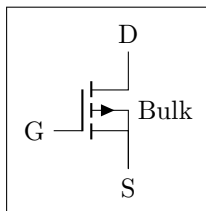
```

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

```

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

Similarly, transistors like other components can be reflected vertically:



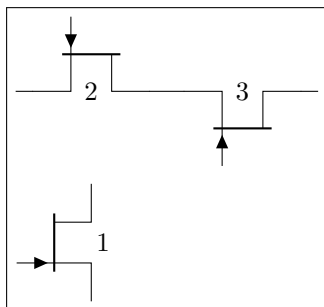
```

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}

```

3.17.8 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:

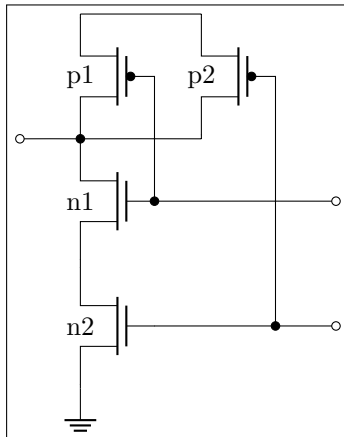


```

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, -*]
6     (0,3)
7   (p1.G) -- (n1.G) to[short, *-o] ($(n1.G)+(3,0)$)
8   (n2.G) ++(2,0) node[circ] {} -| (p2.G)
9   (n2.G) to[short, -o] ($(n2.G)+(3,0)$)
10  (0,3) to[short, -o] (-1,3)
11 \end{circuitikz}

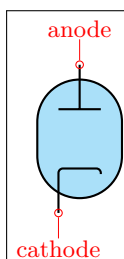
```

Transistor paths have the possibility to use the poles syntax (see section 5.1) 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.

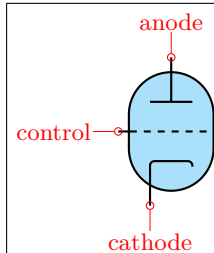
The **name** property is available also for bipoles; this is useful mostly for triac, potentiometer and thyristor (see 3.7.1).

3.18 Electronic Tubes

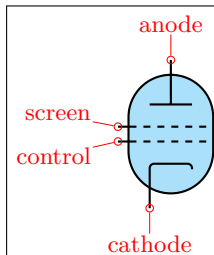
Electronic tubes, also known as vacuum tubes, control current flow between electrodes. They come in many different flavours. Contributed by J. op den Brouw (J.E.J.opdenBrouw@hhs.nl).



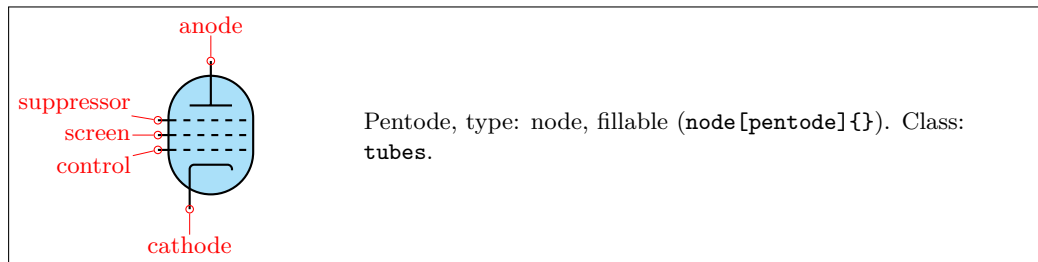
Tube Diode, type: node, fillable (`node[diodetube]{}).` Class: **tubes.**



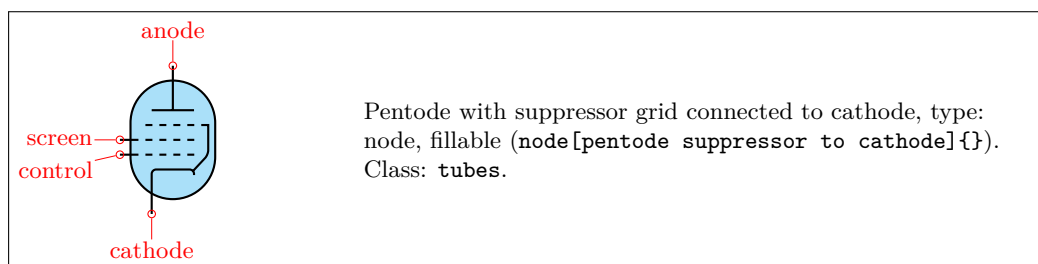
Triode, type: node, fillable (`node[triode]{}).` Class: **tubes.**



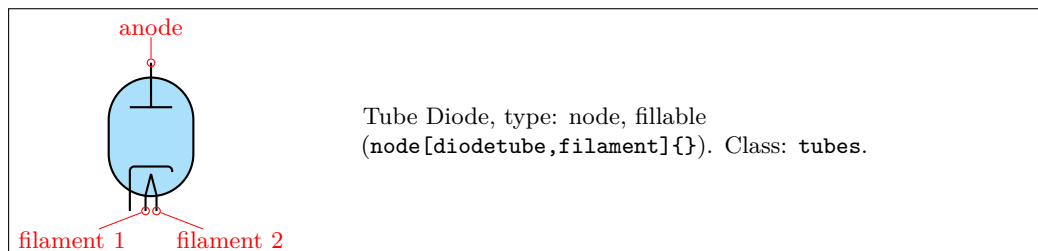
Tetrode, type: node, fillable (`node[tetrode]{}).` Class: **tubes.**



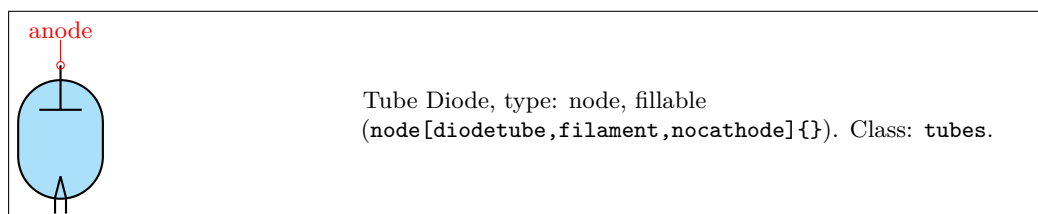
Some pentodes have the suppressor grid internally connected to the cathode, which saves a pin on the tube's housing.



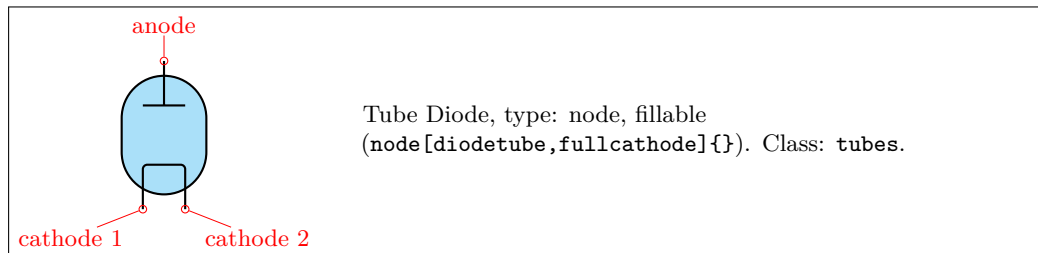
Note that the `diode tube` is used as component name to avoid clashes with the semiconductor diode. Normally, the filament is not drawn. If you want a filament, put the `filament` option in the node description:



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`.

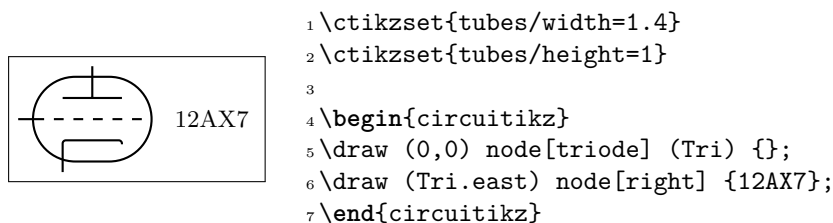


3.18.1 Tubes customization

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:

```

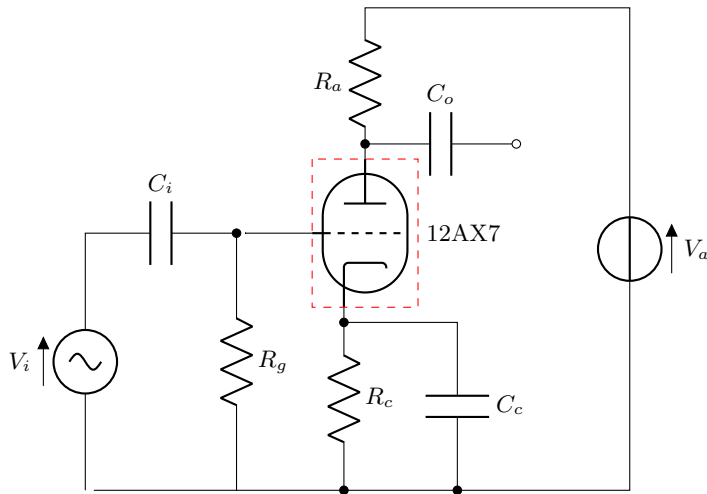
1 \begin{circuitikz}
2 \draw (0,0) node (start) {}
3     to[sV=$V_i$] ++(0,2+\ctikzvalof{tubes/height})
4     to[C=$C_i$] ++(2,0) node (Rg) {}
5     to[R=$R_g$] (Rg |- start)
6 (Rg)     to[short,*-] ++(1,0)
7     node[triode,anchor=control] (Tri) {} ++(2,0)
8 (Tri.cathode) to[R=$R_c$,-*] (Tri.cathode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

```

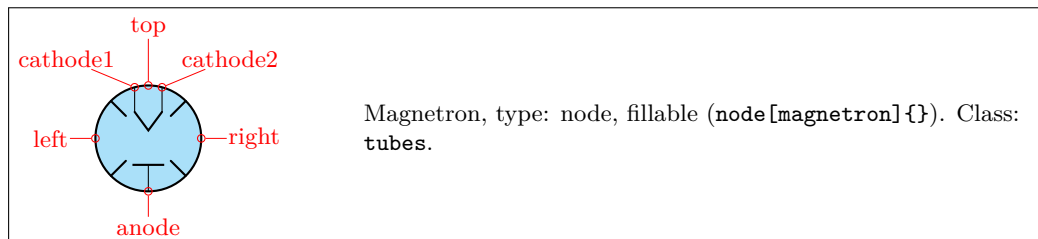
```

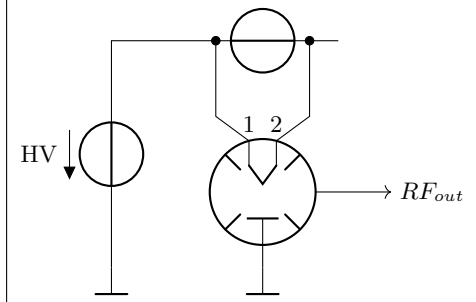
10         to [short] ++(3.5,0) node(Vatop) {}
11         to [V<=$V_a$] (Vatop |- start)
12         to [short] (start)
13 (Tri.anode) ++(0,0.2) to[C=$C_o$,*-o] ++(2,0)
14 (Tri.cathode) ++(0,-0.2) to[short,*-] ++(1.5,0) node(Cctop) {}
15         to[C=$C_c$,-*] (start -| Cctop)
16 ;
17 \draw[red,thin,dashed] (Tri.north west) rectangle (Tri.south east);
18 \draw (Tri.east) node[right] {12AX7};
19 \end{circuitikz}

```



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





```

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}

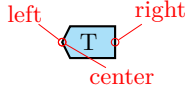
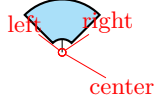
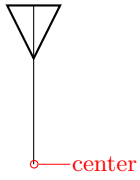
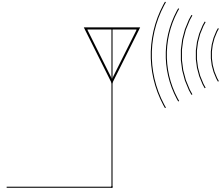
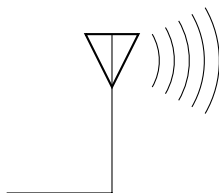
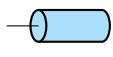


```

3.19 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.

	<p>Bare Antenna, type: node, fillable (<code>node[bareantenna]{A}</code>). Class: RF.</p>
	<p>Bare TX Antenna, type: node, fillable (<code>node[bareTXantenna]{Tx}</code>). Class: RF.</p>
	<p>Bare RX Antenna, type: node, fillable (<code>node[bareRXantenna]{Rx}</code>). Class: RF.</p>
	<p>Waves, type: node (<code>node[waves]{}</code>). Class: RF.</p>
	<p><code>mstline</code>: Microstrip transmission line²⁰, type: path-style, fillable, <code>nodename: mstlineshape</code>. Class: RF.</p>
	<p>Microstrip linear stub, type: node, fillable (<code>node[mslstub]{text}</code>). Class: RF.</p>

	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.
	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: <code>path-style</code> , fillable , nodename: <code>tlineshape</code> . Aliases: <code>transmission line</code> , <code>tline</code> . Class: RF.
	match, type: node (<code>node[match]{}</code>). Class: RF.

3.19.1 RF elements customization

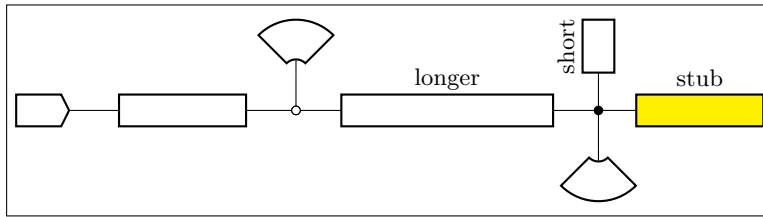
The RF elements can be scaled using the key `RF/scale`, default 1.0.

3.19.2 Microstrip customization

The microstrip linear components' (`mstline`, `msslstub`, `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 `mstlineelen`.

²⁰This four components were suggested by @tcpluess on GitHub



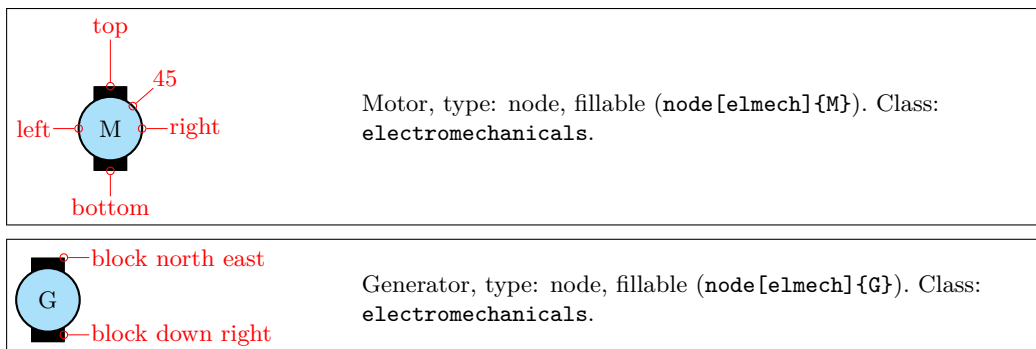
```

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}

```

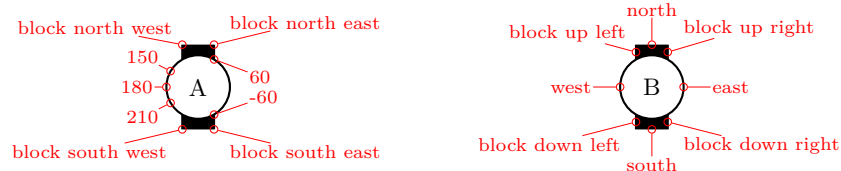
3.20 Electro-Mechanical Devices

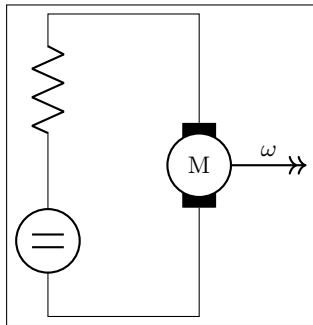
The internal part of the motor and generator are, by default, filled white (to avoid compatibility problems with older versions of the package).



3.20.1 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”:

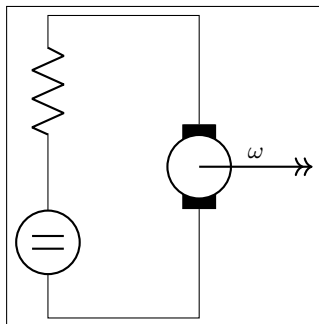




```

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}

```

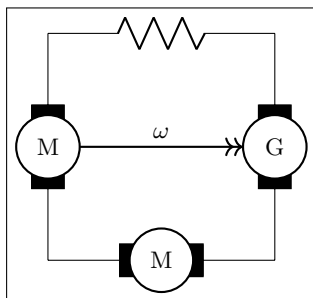


```

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 3.17.8). Don't forget to use parameter n to name the node and get access to the anchors:



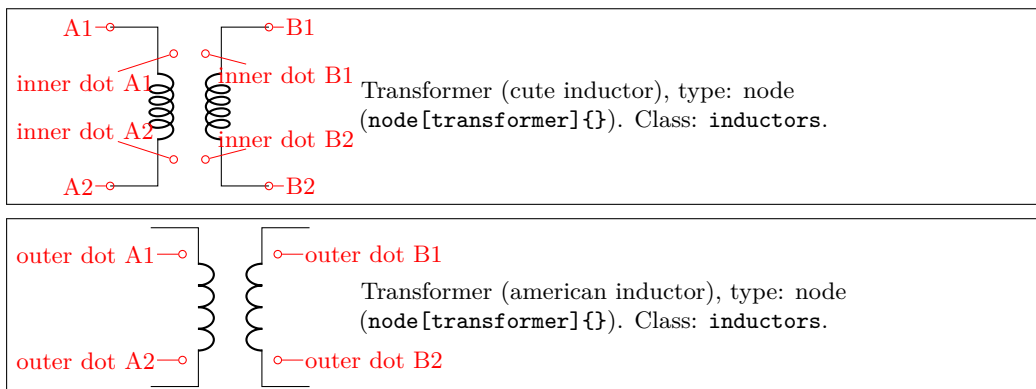
```

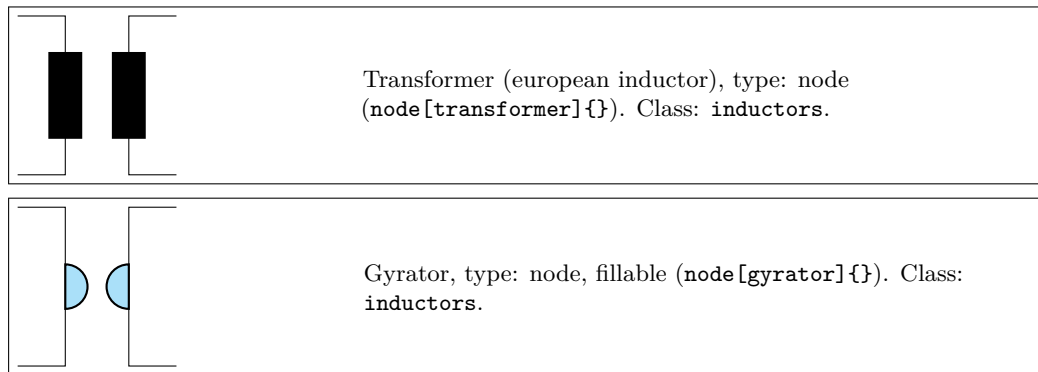
1 \begin{circuitikz}
2 \draw (0,0) to [Telmech=M,n=motor] ++(0,-3) to [
  Telmech=M] ++(3,0) to [Telmech=G,n=generator]
  ++(0,3) to [R] (0,0);
3 \draw[thick,->>] (motor.left)--(generator.left)node[
  midway,above]{\omega};
4 \end{circuitikz}

```

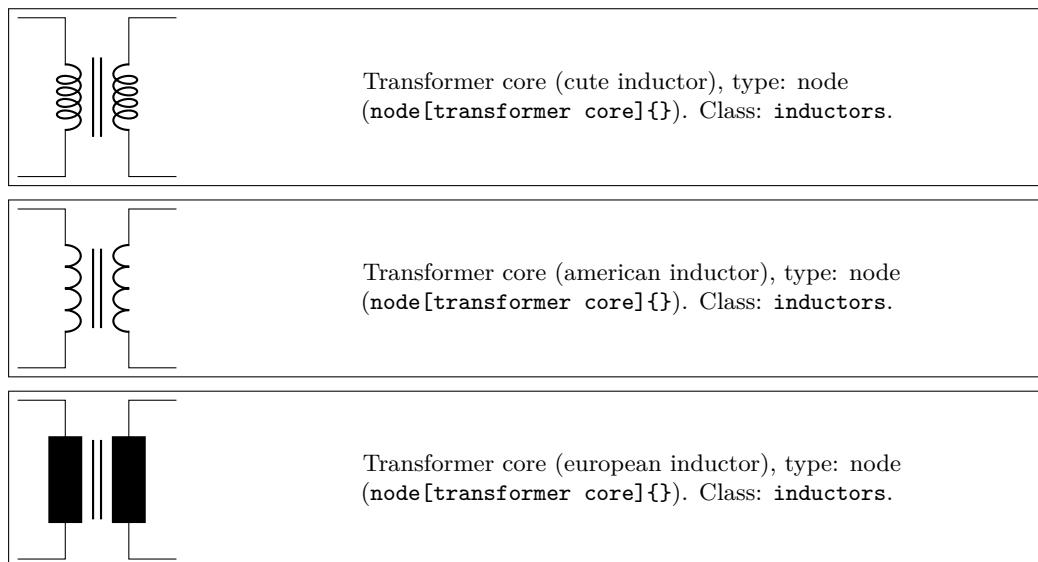
3.21 Double bipoles (transformers)

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





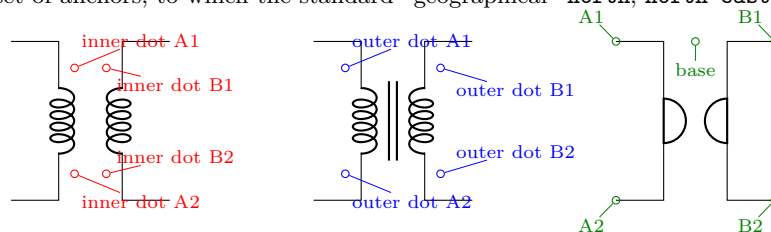
Transformers with core are also available:



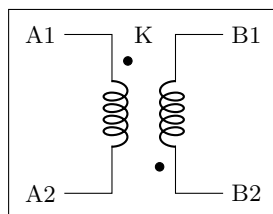
3.21.1 Double dipoles anchors

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

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



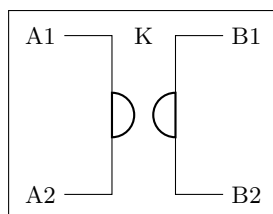
Also, the standard “geographical” `north`, `north east`, etc. are defined. A couple of examples follow:



```

1 \begin{circuitikz} \draw
2   (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   (T.inner dot A1) node[circ]{}
9   (T.inner dot B2) node[circ]{}
10;\end{circuitikz}

```

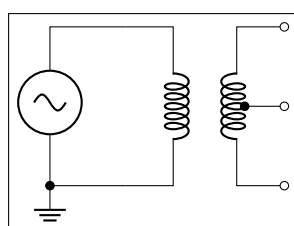
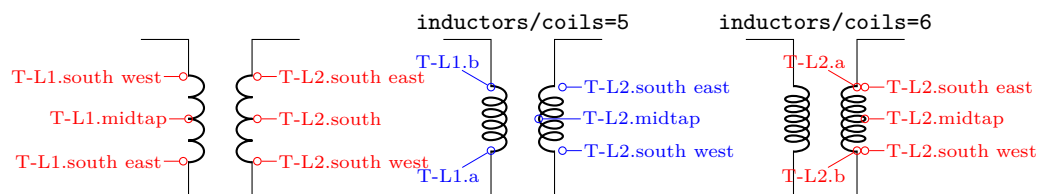


```

1 \begin{circuitikz} \draw
2   (0,0) node[gyrator] (G) {}
3   (G.A1) node[anchor=east] {A1}
4   (G.A2) node[anchor=east] {A2}
5   (G.B1) node[anchor=west] {B1}
6   (G.B2) node[anchor=west] {B2}
7   (G.base) node{K}
8;\end{circuitikz}

```

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.



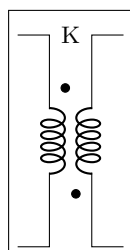
```

1 \begin{circuitikz}
2 \draw (0,0) node[ground](GND){} to [sV] ++(0,2) -- ++(1,0)
3   node[transformer, circuitikz/inductors/coils=6,
4     anchor=A1] (T){};
5 \draw (T.A2) to[short, -*] (T.A2-|GND);
6 \draw (T-L2.midtap) to[short, *-o] (T.B1 |- T-L2.midtap);
7 \node [ocirc] at (T.B1){}; \node [ocirc] at (T.B2){};
8 \end{circuitikz}

```

3.21.2 Double dipoles customization

Transformers are in the `inductors` class (also the gyrator...), so they scale with the key `inductors/scale`. You can change the aspect of a quadpole using the corresponding parameters `quadpoles/*/width` and `quadpoles/*/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!



```

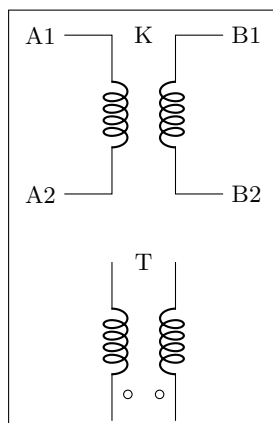
1 \begin{circuitikz}
2 \ctikzset{quadpoles/transformer/width=1,
3   quadpoles/transformer/height=2}
4 \draw (0,0) node[transformer] (T) {}
5   (T.base) node{K}
6   (T.inner dot A1) node[circ]{}
7   (T.inner dot B2) node[circ]{};
8 \end{circuitikz}

```

Transformers also inherits the `inductors/scale` (see 3.6.5) 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`.

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):

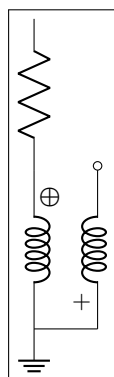


```

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/width=0.6}
9 \draw (0,-3) node[transformer] (P) {}
10 (P.base) node{T}
11 (P.inner dot A2) node[ocirc]{}
12 (P.inner dot B2) node[ocirc]{};
13 \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`):



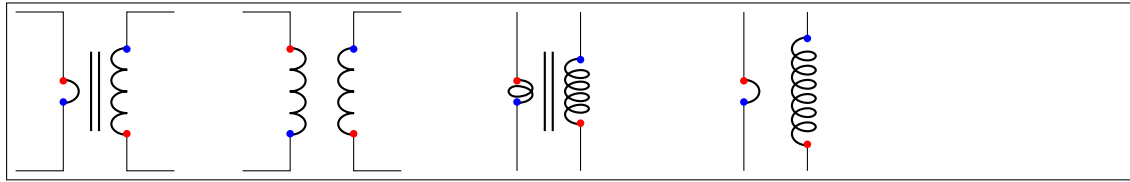
```

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}

```

3.21.3 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.

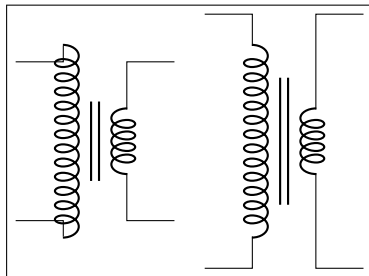


```

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.



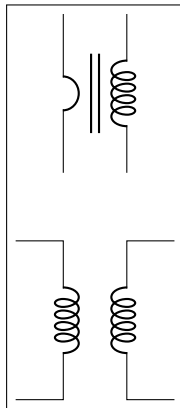
```

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 transformer²¹ defined like this:

²¹Suggested by Alex Pacini on [GitHub](#)



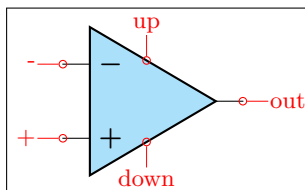
```

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}

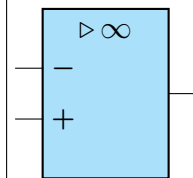
```

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.

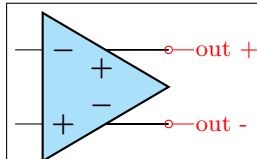
3.22 Amplifiers



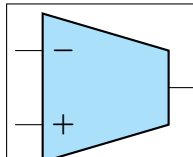
Operational amplifier, type: node, fillable (`node[op amp]{}`). Class: **amplifiers**.



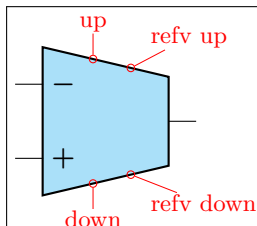
Operational amplifier compliant to DIN/EN 60617 standard, type: node, fillable (`node[en amp]{}`). Class: **amplifiers**.



Fully differential operational amplifier²², type: node, fillable (`node[fd op amp]{}`). Class: **amplifiers**.

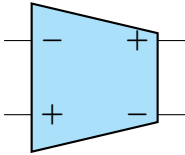
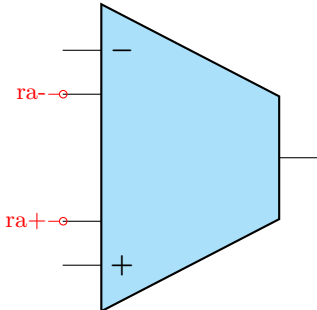
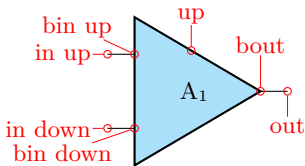
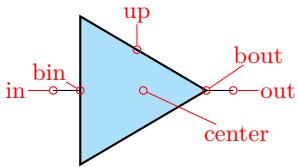
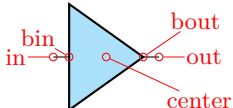


transconductance amplifier, type: node, fillable (`node[gm amp]{}`). Class: **amplifiers**.



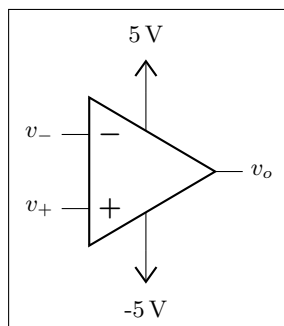
plain instrumentation amplifier, type: node, fillable (`node[inst amp]{}`). Class: **amplifiers**.

²²Contributed by Kristofer M. Monisit.

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

3.22.1 Amplifiers anchors

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

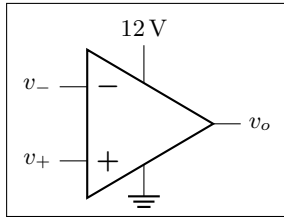


```

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:

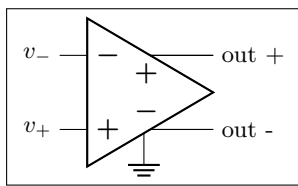


```

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:

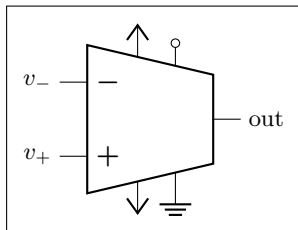


```

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):

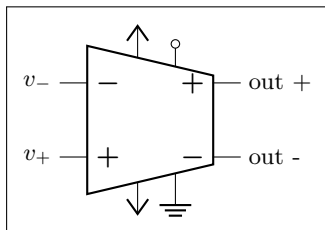


```

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:

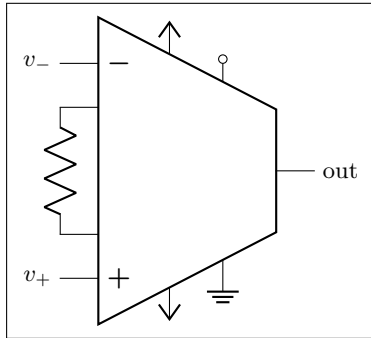


```

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) defines also terminals to add an amplification resistor:

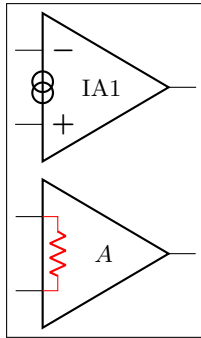


```

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 have also “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** anchors (on the border midway between input) is available.



```

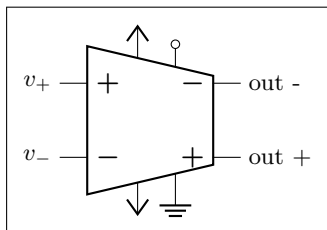
1 \begin{circuitikz}[]
2   \draw (0,2.2) node[op amp] (OA){IA1};
3   \node[oosourceshape, rotate=90, scale=0.5]
4     at (OA.leftedge) {};
5   \draw (0,0) node[plain amp] (A){$A$};
6   \draw [color=red] (A.bin up) -- ++(0.2,0)
7     coordinate (tmp)
8     to[R, resistors/scale=0.5]
9     (tmp|-A.bin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

```

3.22.2 Amplifiers customization

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

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



```

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 have also the anchors **in up**, **in down**, **out up**, **out down** that will not change with the positive or negative sign.

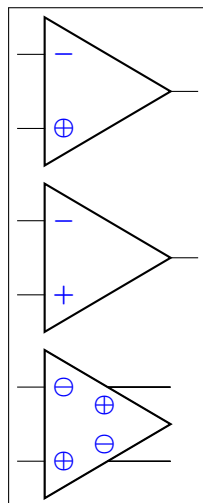
You can change the symbols “+” or “-” appearing in the amplifiers if you want, both globally and on component-by-component basis. The plus and minus symbols can be changed with **\ctikzset** of the keys

`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

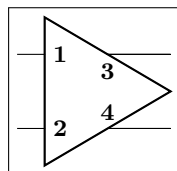
```
1 \tikzset{amp symbol font={%
2 \color{blue}\fontsize{12}{12}\selectfont\boldmath}}
```

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



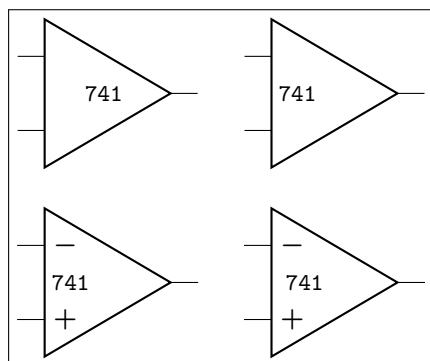
```
1 \begin{circuitikz}[]
2   % change in this circuit only
3   \tikzset{amp symbol font={\color{blue}\small\boldmath}}
4   % local change
5   \draw (0,2.2) node[op amp, amp plus=${\oplus}]{};
6   \draw (0,0) node[op amp]{};
7   % from now on...
8   \ctikzset{amplifiers/plus=${\oplus}}
9   \ctikzset{amplifiers/minus=${\ominus}}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

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.

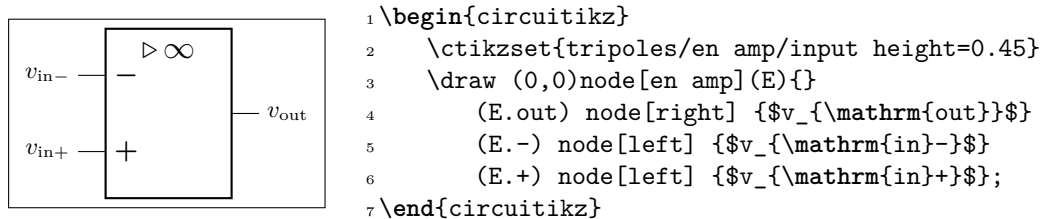


```
1 \begin{circuitikz}[]
2   \draw (0,2.5) node[plain amp]{\texttt{741}};
3   \draw (3,2.5)
4     node[plain amp, component text=left]
5     {\texttt{741}};
6   \ctikzset{component text=left}
7   \draw (0,0) node[op amp]{\texttt{741}};
8   \ctikzset{left text distance=0.6em}
9   \draw (3,0) node[op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

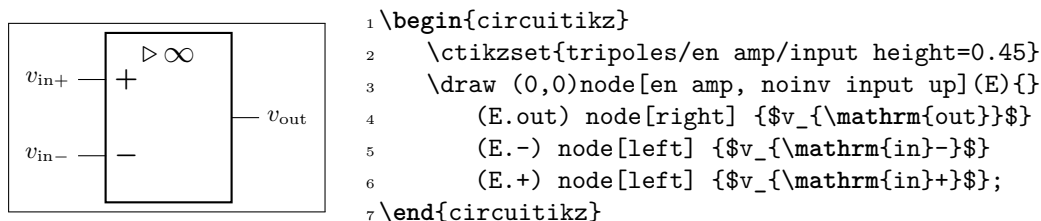
These keys are also used for the positioning of the labels in the label positioning of IEEE logic gates (see 3.24.2).

3.22.2.1 European-style amplifier customization Thanks to the suggestions from David Rouvel (david.rouvel@iphc.cnrs.fr) there are several possible customization for the European-style amplifiers. Since 0.9.0, the default appearance of the symbol has changed to be more in line with the standard; notice that to have a bigger triangle by default we should require more packages, and I fear ConTeXt compatibility; but see later on how to change it. Notice that the font used for the symbol is defined in `tripoles/en amp/font2` and that the font used for the + and - symbols is `tripoles/en amp/font`.

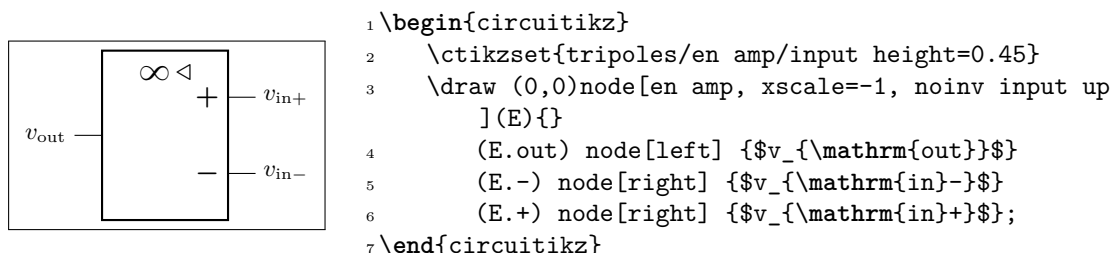
You can change the distances of the inputs, using `tripoles/en amp/input height` (default 0.3):



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

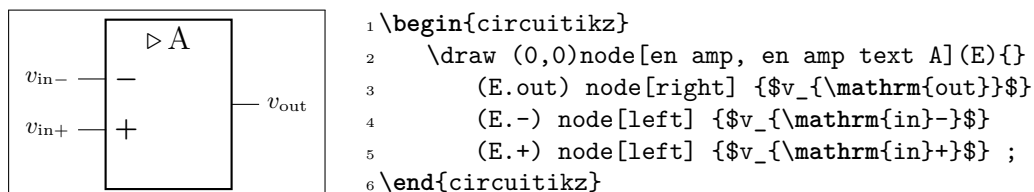


To flip the amplifier in the horizontal direction, you can use `xscale=-1` as usual:

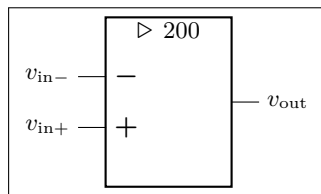


Notice that the label is fully mirrored, so check below for the generic way to change this.

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



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



```

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}

```

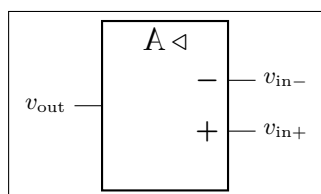
Notice two things here: the first, that `\triangleright` is enclosed in braces to remove the default spacing it has as a binary operator, and that `en amp text A` is simply a shortcut for

```

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

```

To combine flipping with a generic label you just do:

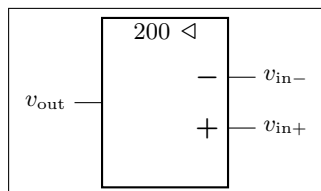


```

1 \begin{circuitikz}
2   \draw (0,0)node[en amp, xscale=-1, en amp text A](
3     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}

```

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



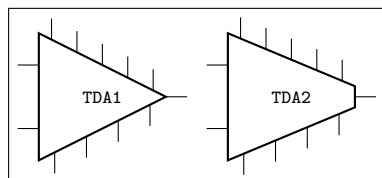
```

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

```

3.22.3 Designing your own amplifier

If you need a different kind of amplifier, you can use the `muxdemux` (see section 3.26) shape for defining one that suits your needs (you need version 1.0.0 for this to work).



```

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,
7     muxdemux def={Rh=0.5}]{TDA2};
8 \end{circuitikz}

```

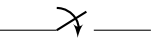
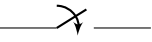
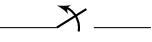

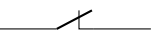
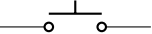

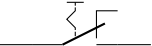

3.23 Switches and buttons

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

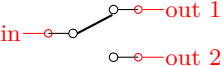
The switches can be scaled with the key `switches/scale` (default 1.0). Notice that scaling the switches will not scale the poles, which are controlled with their own parameters (see section 3.15).

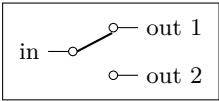
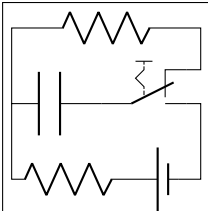
3.23.1 Traditional switches

These are all of the to-style type:

	<code>switch</code> : Switch, type: <code>path-style</code> , nodename: <code>cspstshape</code> . Aliases: <code>spst</code> . Class: <code>switches</code> .
	<code>closing switch</code> : Closing switch, type: <code>path-style</code> , nodename: <code>cspstshape</code> . Aliases: <code>cspst</code> . Class: <code>switches</code> .
	<code>opening switch</code> : Opening switch, type: <code>path-style</code> , nodename: <code>ospstshape</code> . Aliases: <code>ospst</code> . Class: <code>switches</code> .
	<code>normal open switch</code> : Normally open switch, type: <code>path-style</code> , nodename: <code>nossshape</code> . Aliases: <code>nos</code> . Class: <code>switches</code> .
	<code>normal closed switch</code> : Normally closed switch, type: <code>path-style</code> , nodename: <code>ncssshape</code> . Aliases: <code>ncs</code> . Class: <code>switches</code> .
	<code>push button</code> : Normally open push button, type: <code>path-style</code> , nodename: <code>pushbuttonshape</code> . Aliases: <code>normally open push button</code> , <code>nopb</code> . Class: <code>switches</code> .
	<code>normally closed push button</code> : Normally closed push button, type: <code>path-style</code> , nodename: <code>ncpushbuttonshape</code> . Aliases: <code>ncpb</code> . Class: <code>switches</code> .
	<code>toggle switch</code> : Toggle switch, type: <code>path-style</code> , nodename: <code>toggleswitchshape</code> . Class: <code>default</code> .
	<code>reed</code> : Reed switch, type: <code>path-style</code> , fillable , nodename: <code>reedshape</code> . Class: <code>switches</code> .

while this is a node-style component:

	<code>spdt</code> , type: <code>node</code> (<code>node[spdt]{}</code>). Class: <code>switches</code> .
---	---

	<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>

3.23.2 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:

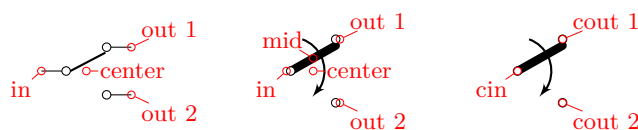
	cute closed switch: Cute closed switch, type: <code>path-style</code> , nodename: <code>cuteclosedswitchshape</code> . Aliases: <code>ccsw</code> . Class: <code>switches</code> .
	cute open switch: Cute open switch, type: <code>path-style</code> , name=B, nodename: <code>cuteopenswitchshape</code> .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

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

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

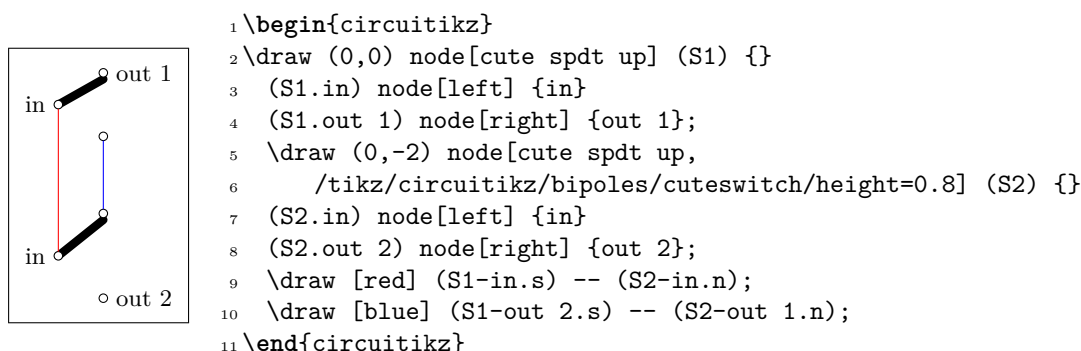
3.23.2.1 Cute switches anchors

The nodes-style switches have the following anchors:

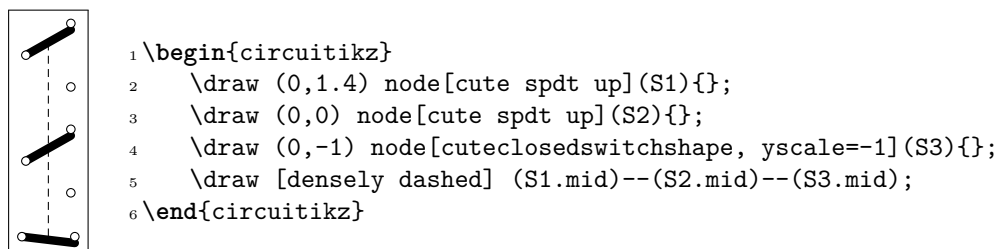


Please notice the position of the normal anchors at the border of the `ocirc` shape for the cute switches; they are thought to be compatible with an horizontal wire going out. Additionally, you have the `cin`, `cout 1` y `cout 2` which are anchors on the center of the contacts.

For more complex situations, the contact nodes are available²³ using the syntax *name of the node*-in, ...-out 1 and ...-out 2, with all their anchors.

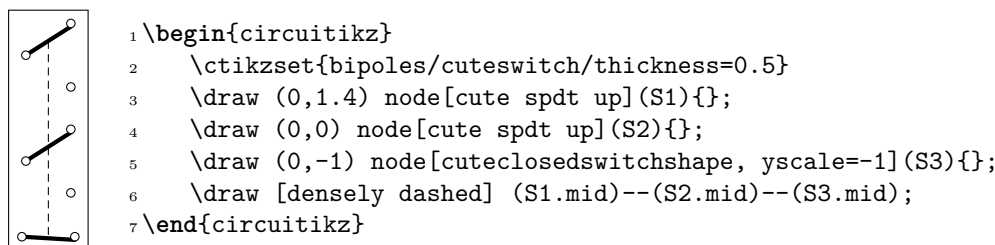


The `mid` anchor in the cute switches (both path- and node-style) can be used to combine switches to get more complex configurations:



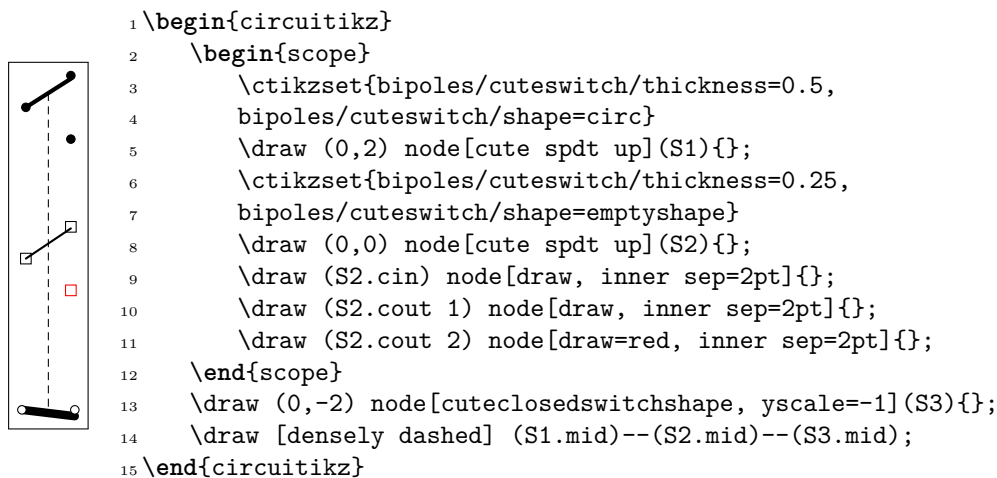
3.23.2.2 Cute switches customization

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.



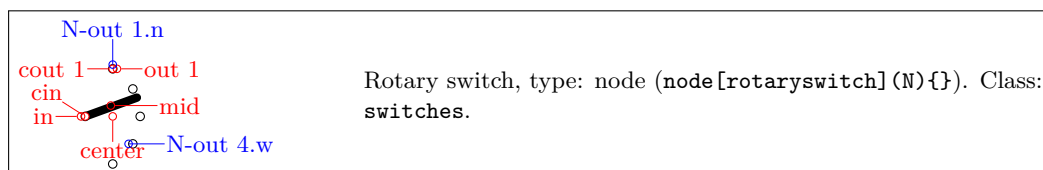
Finally, the switches are normally drawn using the `ocirc` shape, but you can change it, as in the following example, with the key `bipoles/cuteswitch/shape`. Be careful that the shape is used with its defaults (which can lead to strange results), and that the standard anchors will be correct only for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

²³Thanks to @marmot on tex.stackexchange.com.



3.23.3 Rotary switches

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



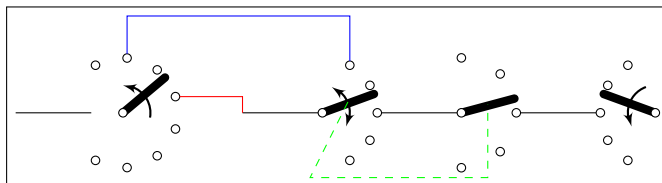
Notice that the name of the shape is `rotaryswitch`, no spaces. The default rotary switch component has 5 channels (this is set in the parameter `multipoles/rotary/channels`), spanning from -60° to 60° (parameter `multipoles/rotary/angle`) and with the wiper at 20° (parameter `multipoles/rotary/wiper`).

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.

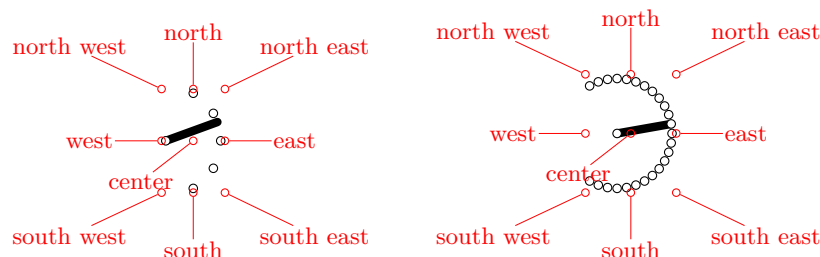


```

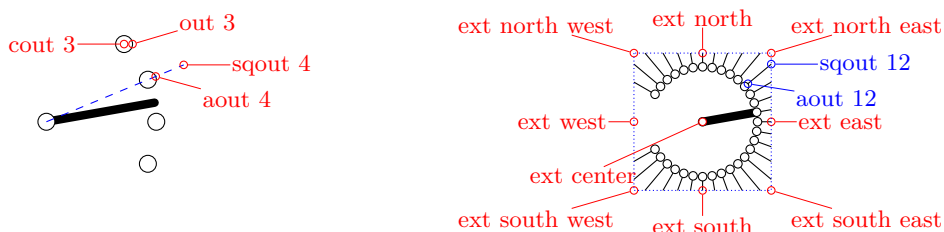
1 \begin{circuitikz}
2 \ctikzset{multipoles/rotary/arrow=both}
3 \draw (0,0) -- ++(1,0) node[rotary switch <-=8 in 120 wiper 40, anchor=in](A){};
4 \draw (3,0) -- ++(1,0) node[rotary switch, anchor=in](B){}; % default values
5 \draw[red] (A.out 4) -| (3,0);
6 \draw[blue] (A.out 2.n) -- ++(0,0.5) -| (B.out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

```

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



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



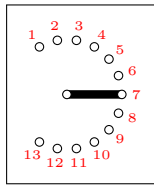
The code for the diagram at the left, above, without the markings for the anchors, is:

```

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:

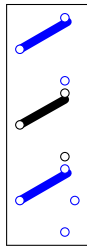


```

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

```

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

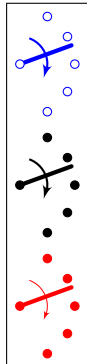


```

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

```

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

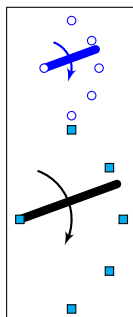


```

1 \begin{circuitikz}
2   \ctikzset{multipoles/rotary/thickness=0.5}
3   \draw (0,1.6) node[rotary switch ->, color=blue](S1){};
4   \ctikzset{multipoles/rotary/shape=circ}
5   \draw (0,0) node[rotary switch ->](S2){};
6   \ctikzset{bipoles/thickness=0.5}
7   \draw (0,-1.6) node[rotary switch ->, color=red](S3){};
8 \end{circuitikz}

```

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



```

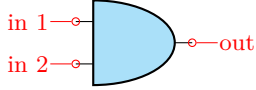
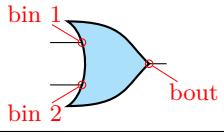
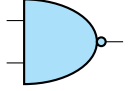
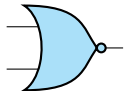
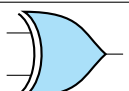
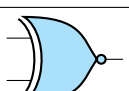
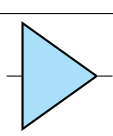
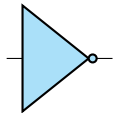
1 \begin{circuitikz}
2   \draw (0,2) node[rotary switch ->, color=blue](S1){};
3   \ctikzset{tripoles/spdt/width=1.6, fill=cyan,
4     multipoles/rotary/shape=rectangle}
5   \draw (0,0) node[rotary switch ->](S2){};
6 \end{circuitikz}

```

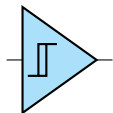
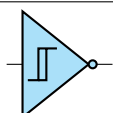
3.24 Logic gates

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).

3.24.1 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.

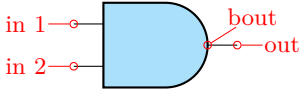
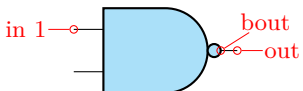

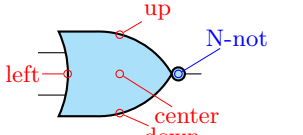
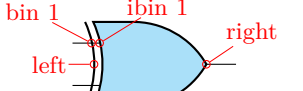
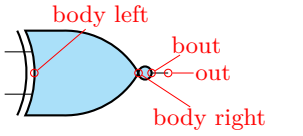
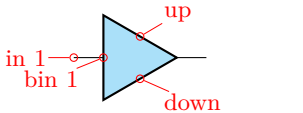
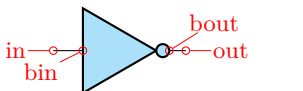
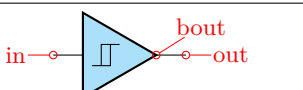
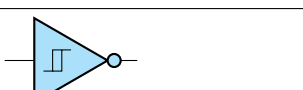
There is no “european” version of the following symbols; for now they are used both in **american** and **european** styles, but it may change in the future.

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


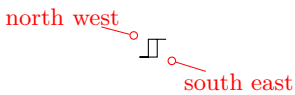
3.24.2 IEEE logic gates

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 available²⁴.


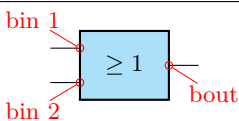

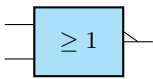
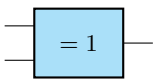
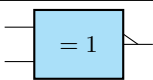


These ports are completely independent from the legacy set (either **american** or **european**); they are not enabled by default because the relative size of the ports is very different from the legacy ones, and that will disrupt every schematic (especially if drawn with absolute coordinate). If you 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.

	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 .
	Inverting Schmitt port matched to IEEE standard ports, type: node, fillable (<code>node[ieeestd invschmitt port]{}).</code> Class: logic ports .

²⁴Thanks to Jason for proposing it and digging out the info, see this [GitHub issue](#).

	Inverting dot for IEEE ports, type: node, fillable (node[notcirc]{}). Class: logic ports.
	Schmitt symbol to add to input pins if needed, type: node, fillable (node[schmitt symbol]{}). Class: logic ports.

3.24.3 European Logic gates

	European AND port, type: node, fillable (node[european and port]{}). Class: logic ports.
	European OR port, type: node, fillable (node[european or port]{}). Class: logic ports.
	European NAND port, type: node, fillable (node[european nand port]{}). Class: logic ports.
	European NOR port, type: node, fillable (node[european nor port]{}). Class: logic ports.
	European XOR port, type: node, fillable (node[european xor port]{}). Class: logic ports.
	European XNOR port, type: node, fillable (node[european xnor port]{}). Class: logic ports.
	European BUFFER port, type: node, fillable (node[european buffer port]{}). Class: logic ports.
	European NOT port, type: node, fillable (node[european not port]{}). Class: logic ports.

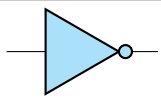
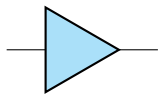
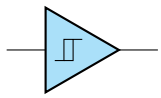
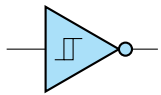
If (default behaviour) `americanports` option is active (or the style `[american ports]` is used), the shorthands `and port`, `or port`, `buffer port`, `nand port`, `nor port`, `not port`, `xor port`, `xnor port`, `schmitt port` and `invschmitt port` are equivalent to the american version of the respective logic port.

If otherwise `europenports` option is active (or the style `[european 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.

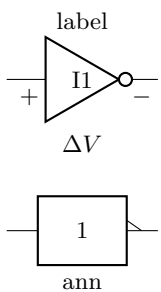
Finally, for version 1.1.0 and up, you can use the style `ieee ports` to set the shorthands to the set of `ieeestd` ports. (There is no global option for this).

3.24.4 Path-style logic ports

The one-input, one-output ports have a handy path-style equivalent; they are the following:

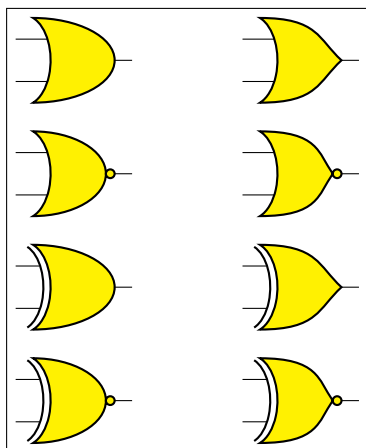
	<code>inline not:</code> “not” logic port, <code>type: path-style</code> , <code>fillable</code> , <code>nodename: not port</code> . Class: logic ports.
	<code>inline buffer:</code> “buffer” logic port, <code>type: path-style</code> , <code>fillable</code> , <code>nodename: buffer port</code> . Class: logic ports.
	<code>inline schmitt:</code> Schmitt logic port, <code>type: path-style</code> , <code>fillable</code> , <code>nodename: schmitt port</code> . Class: logic ports.
	<code>inline invschmitt:</code> Inverting Schmitt logic port, <code>type: path-style</code> , <code>fillable</code> , <code>nodename: invschmitt port</code> . Class: logic ports.

Those ports follows the current selected style, although you can change it on the fly (even if it has not 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.

	<pre> 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} </pre>
---	--

3.24.5 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 going 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.



```

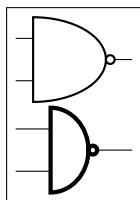
1 \begin{circuitikz}[
2   american]
3   % legacy shapes
4   \ctikzset{american or shape=roundy}
5   \ctikzset{logic ports/fill=yellow}
6   \node [or port](01) at (0,0) {};
7   \node [nor port](02) at (0,-1.5) {};
8   \node [xor port](03) at (0,-3) {};
9   \node [xnor port](04) at (0,-4.5) {};
10  \begin{scope}[xshift=3cm]
11    % new shapes
12    \ctikzset{american or shape=pointy}
13    \node [or port](01) at (0,0) {};
14    \node [nor port](02) at (0,-1.5) {};
15    \node [xor port](03) at (0,-3) {};
16    \node [xnor port](04) at (0,-4.5) {};
17  \end{scope}
18 \end{circuitikz}

```

3.24.5.1 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`:

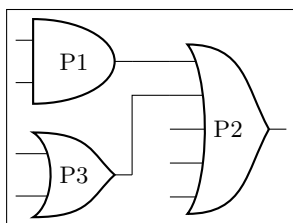


```

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` :

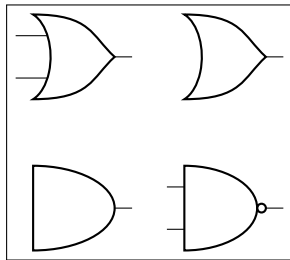


```

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 do do the connection to the “border”-anchors.

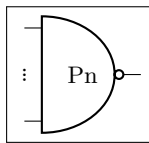


```

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] (A1) at (0,0) {};
6   \node [nand port, input leads] (A1) at (2,0) {};
7 \end{circuitikz}

```

This is useful if you need to draw a generic port, like the one following here:



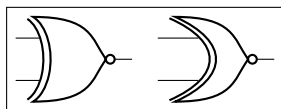
```

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 go “into” the shape, default 0.3) and `angle` (the angle at which the base starts, default 70).



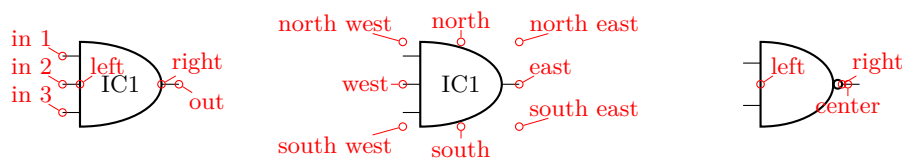
```

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] {};

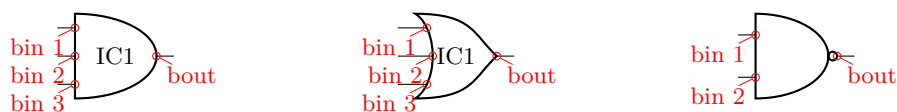
```

3.24.5.2 American logic port anchors

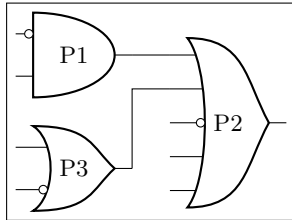
These are the anchors for logic ports:



You have also “border pin anchors”:



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

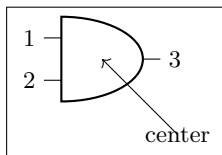


```

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,
7     number 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}

```

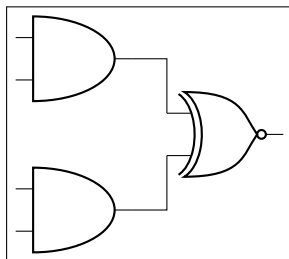
As you can see, the `center` anchor is (for historic reasons) not in the center at all. You can fix this with the command `\ctikzset{logic ports origin=center}`:



```

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)
8   node{center};
9 \end{circuitikz}

```

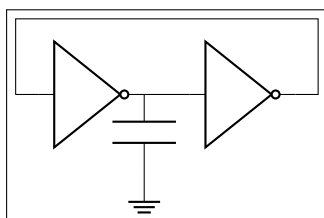


```

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`):

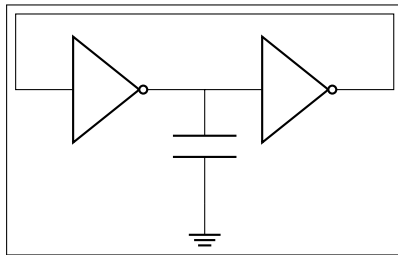


```

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:



```

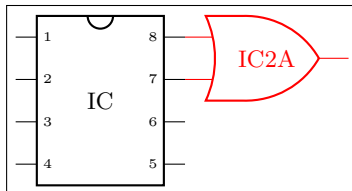
1 \begin{circuitikz}[american]
2   \draw (0,0) node[ground]{} to[C] ++(0,1.5)
3   coordinate(c)
4   to[inline not] ++(2.5,0) -- ++(0,1)
5   -| ++(-5,-1)
6   to[inline not] (c);
7 \end{circuitikz}

```

3.24.6 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 does not change when you change the size, so you can't have a "tall" port or a "squatty" ones. The two-inputs gates, by default, have their default size designed so that they match the chips component (see 3.27).

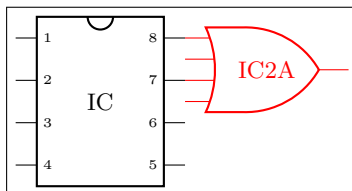


```

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:

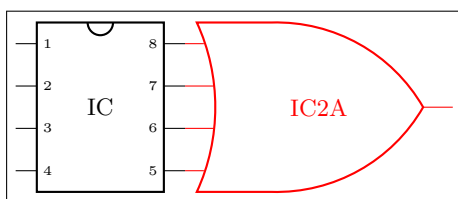


```

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.

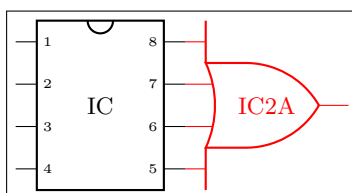


```

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 other are put on an extended input line.



```

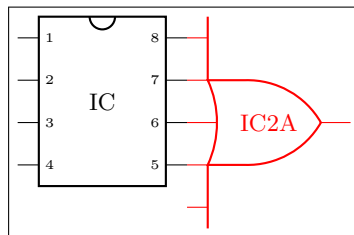
1 \begin{circuitikz}
2   \draw (0,0) node[dipchip](C){IC} (C.pin 8)
3   node[or port, anchor=in 1,
4   number inputs=4,
5   inner inputs=2,
6   color=red](A){IC2A};
7 \end{circuitikz}

```

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.

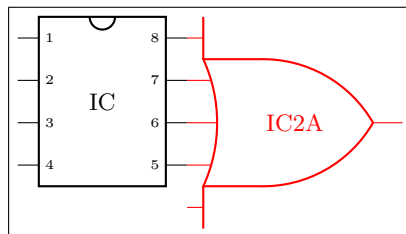


```

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.

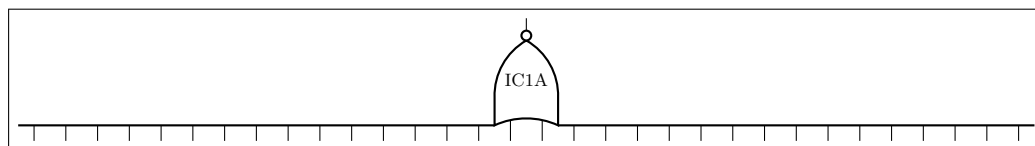


```

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.

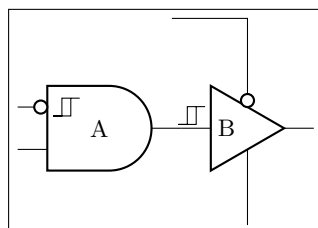


```

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...):



```

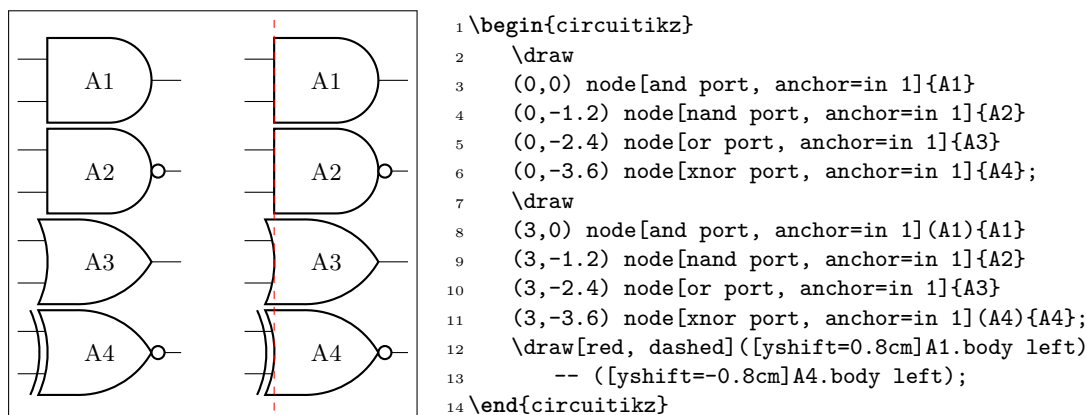
1 \begin{circuitikz}
2   \draw (0,0) node[and port] (A){A} (A.out)
3     node[buffer port, anchor=in,
4       component text=left] (B){B} (B.bin)
5     node[schmitt symbol, above left] {}
6     (A.bin 1) node[schmitt symbol, right] {};
7   \node [notcirc, left] at (A.bin 1) {};
8   \node [notcirc, above] (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.²⁵

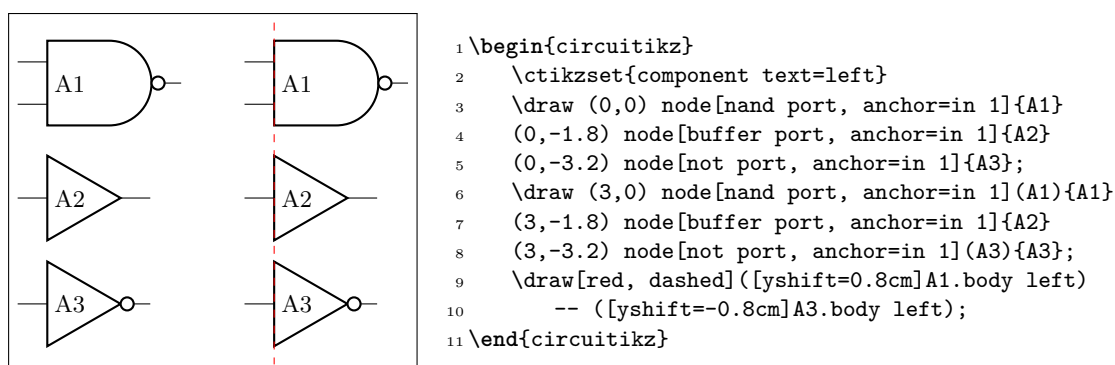
²⁵You can use the same key with amplifiers, too.

3.24.6.1 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.



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.

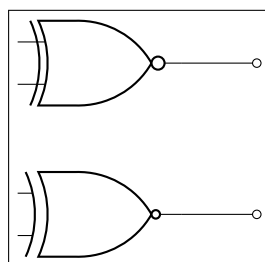
The single input ports (**not port**, **buffer port** and their Schmitt equivalent) are smaller than the six standard ports, so they are not kept aligned by default; they just have the same distance at the input side. For the not ports, the **left** position of the text results often in a better look (the centered text in the triangle seems to be much more at the right).



3.24.6.2 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
baselen	0.4	the basic length for every dimension, as a fraction of the (scaled) resistor length
height	2	the height of the port, in term of baselen . Pin distance is given by this parameter divided by the inner pins.
pin length	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.
not radius	0.154	radius of the “not circle” added to the negated-output ports. The default value is the IEEE recommended one.
xor bar distance	0.192	distance of the detached input shape in xor and xnor ports. The default value is the IEEE recommended one.
xor leads in	1	If set to 0, there will be no leads drawn between the detached input line and the body in the xor and xnor ports. IEEE recommends 1 here.
schmitt symbol size	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.

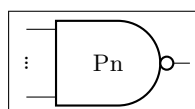


```

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):

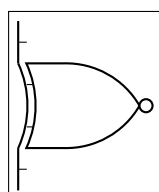


```

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):

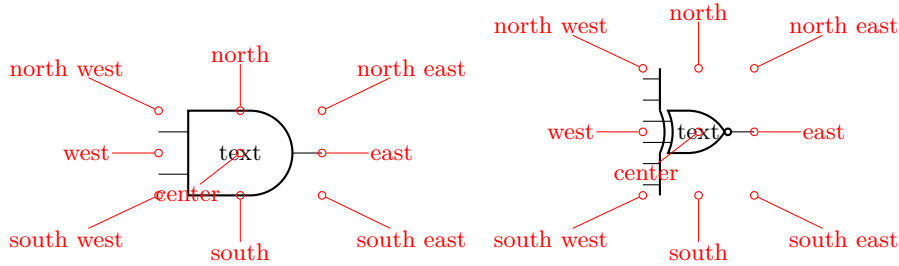


```

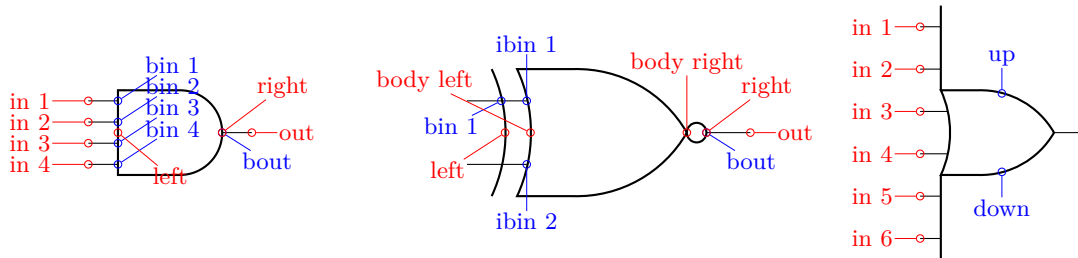
1 \begin{circuitikz}
2   \ctikzset{ieeestd ports/pin length=0.2}
3   \draw (0,0) node[xnor port,
4     number inputs=4, inner inputs=2](B){};
5 \end{circuitikz}

```

3.24.6.3 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**.

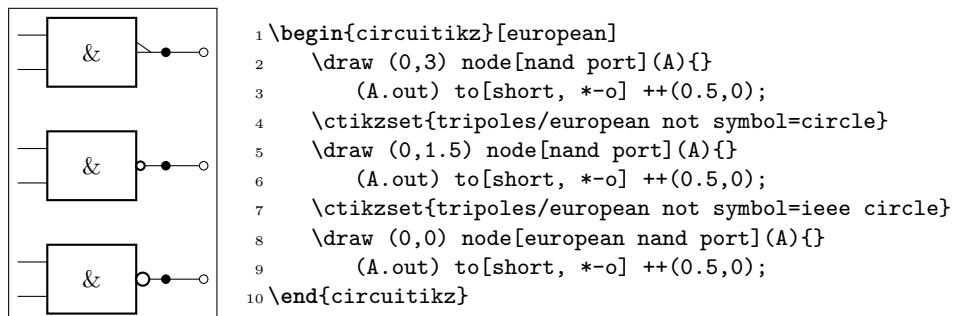
The main body of the port is marked with **body left** and **body right** anchors (as seen in the middle port in the diagram above); you have also an **up** and **down** anchors centered on the body (you can use them as enable signals or similar things).

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.

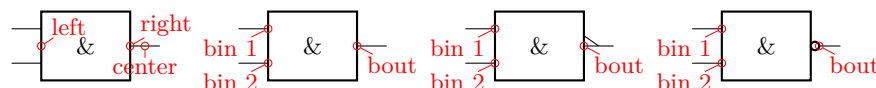
3.24.7 European logic port usage

European logic port 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.

3.24.7.1 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**; 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**.



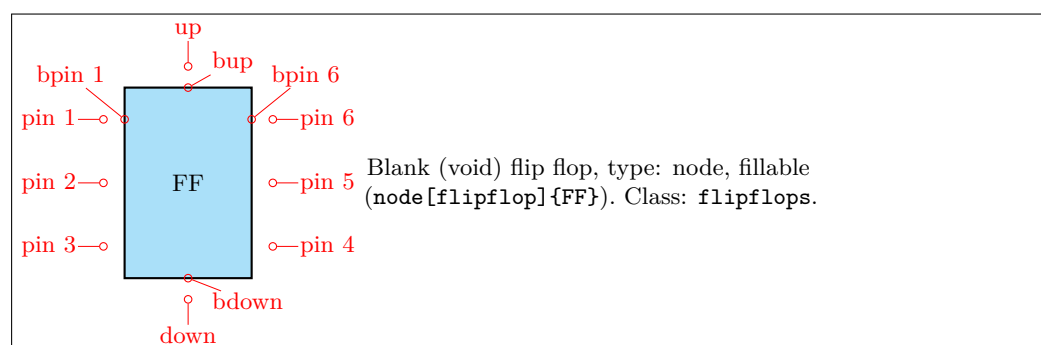
3.24.7.2 European logic port anchors The anchors are basically the same as in the american-style ports.



3.25 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 correspondig pin;
- a *negation* flag (`n0`, ...`n6`, `nu`, `nd`), with value 0 or 1, which will put and `ocirc` shape on the outer border of the correspondig pin.

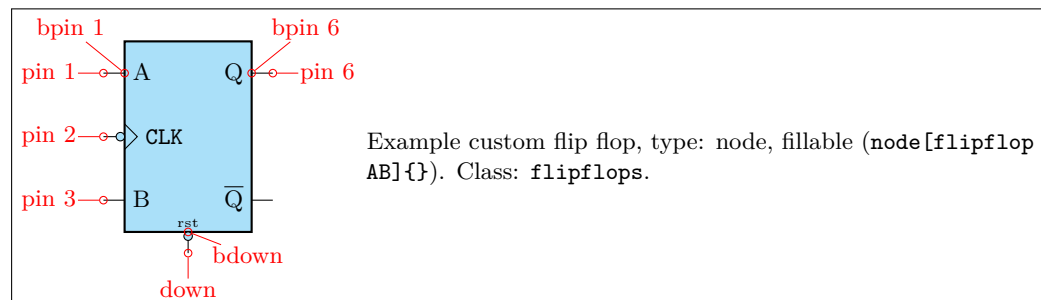
To set all this keys, an auxiliary style `flipflop def` is defined, so that you can do the following thing:

```

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 }}

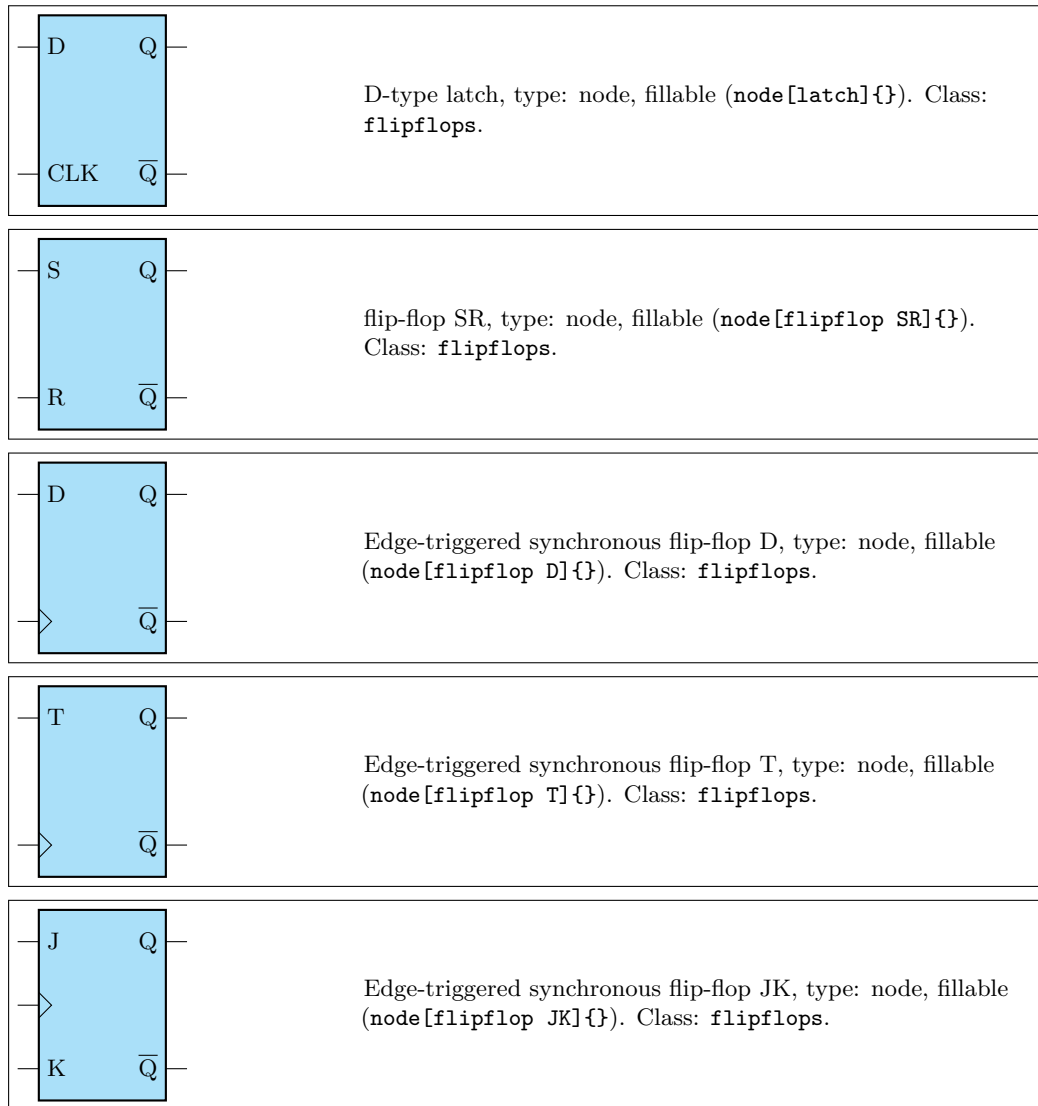
```

to obtain:

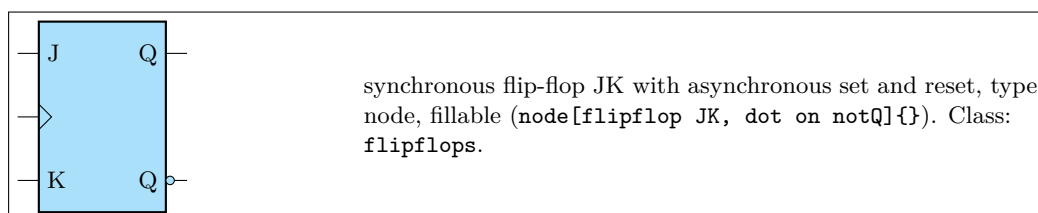


`\ctikztextnot{}` is a small utility macro to set a overbar to a text, like $\overline{\text{RST}}$ (created by `\ctikztextnot{RST}`).

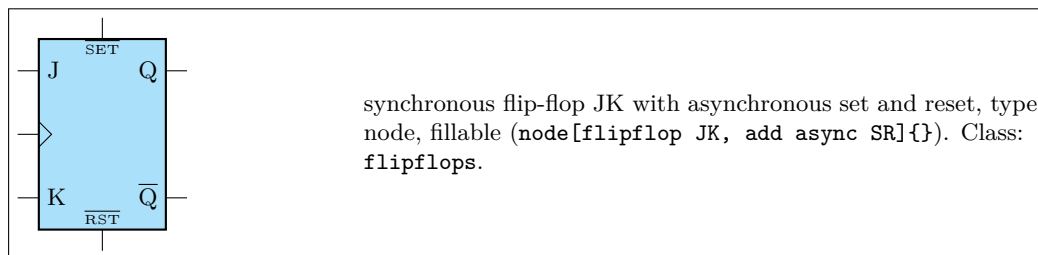
By default, the following flip-flops are defined:



If you prefer that the negated output is labelled \overline{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:

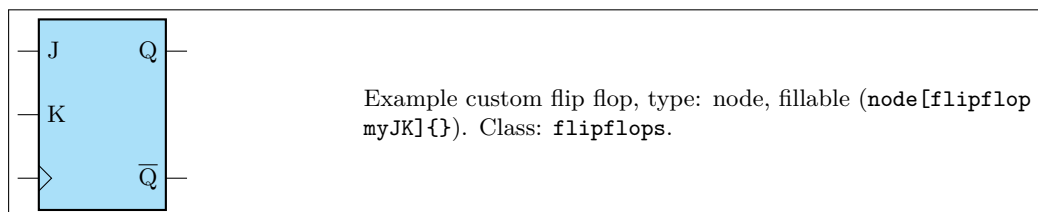


3.25.1 Custom flip-flops

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 }
```



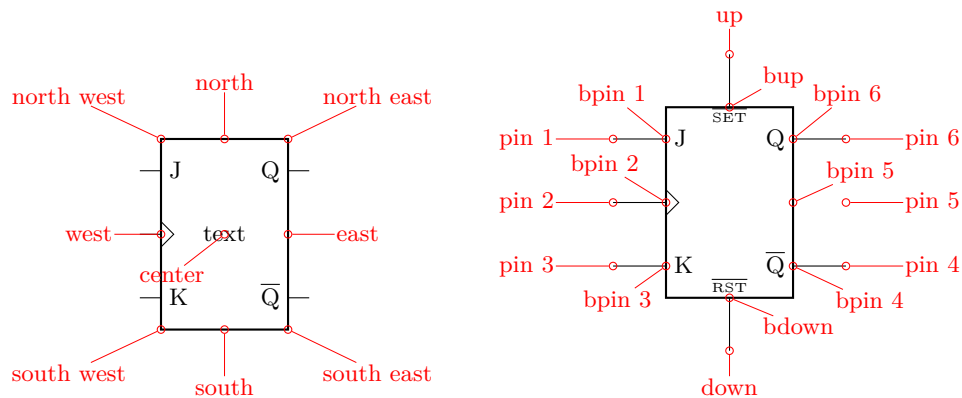
The standard definition of the default flip-flops are the following (in the file `pgfcircmultipoles.tex`):

```

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 }
```

3.25.2 Flip-flops anchors

Flip-flops have all the standard geometrical anchors, although it should be noticed that the external pin 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`).

3.25.3 Flip-flops customization

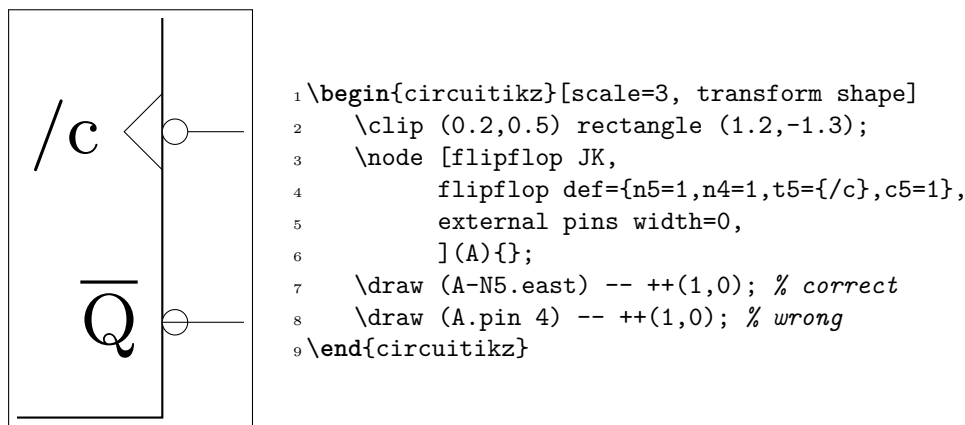
Flip-flop's size is controlled by the class parameters (like `flipflops/scale`) and the specific `\ctikzset` keys `multipoles/flipflop/width` and `multipoles/flipflop/pin spacing`. Class parameters are also used for line thickness and fill color. The default values are matched with the logic ports ones.

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.

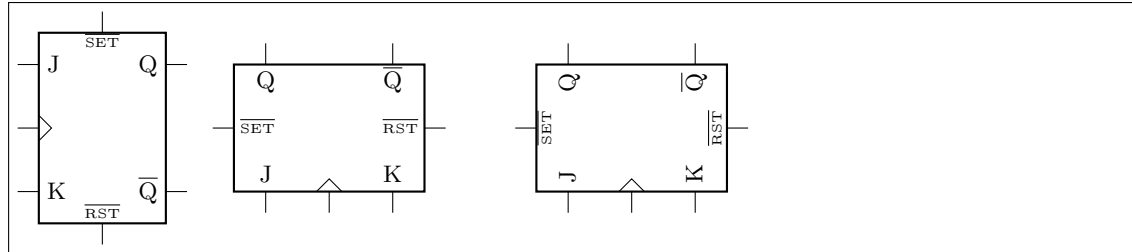
As in chips, you can change the length of the external pin with the key `external pins width`; you can for example have a pinless flip-flop like this:



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 do 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.

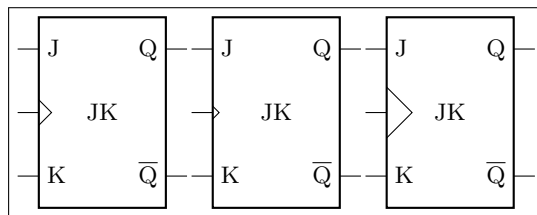


```

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).

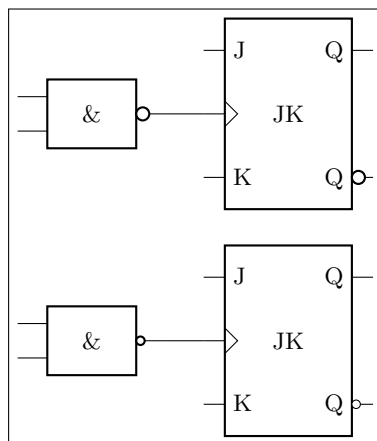


```

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`).



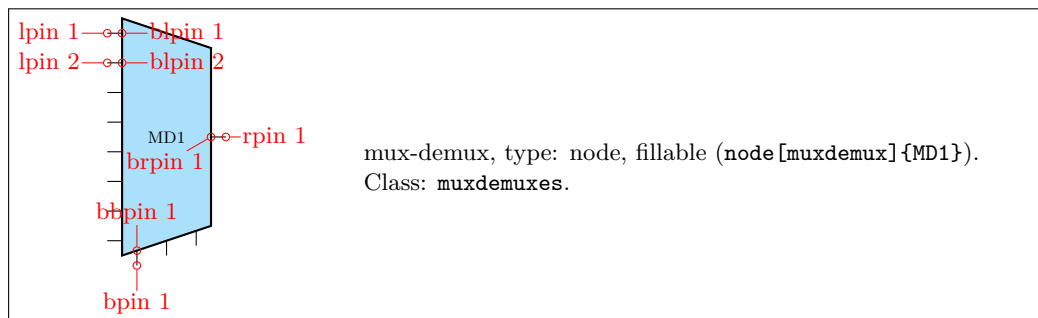
```

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}

```

3.26 Multiplexer and de-multiplexer

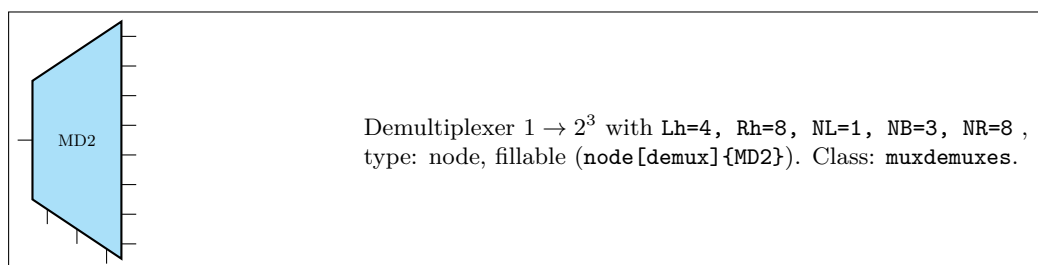
The shape used for muxes and de-muxes is probably the most configurable shape of the package; it has been added by Romano in v1.0.0. The basic shape is a multiplexer with 8 input pin, one output pin, and three control pins ($2^3 \rightarrow 1$ multiplexer). The pins are not named as input or output pins (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:

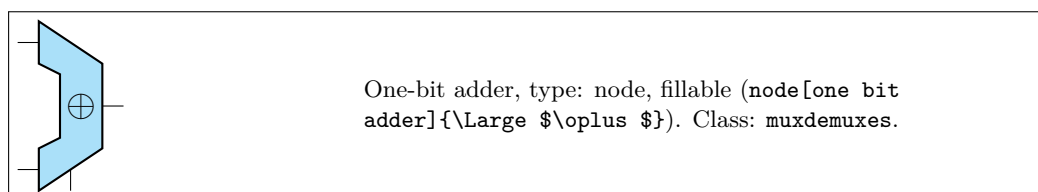
```
1 \tikzset{demux/.style={muxdemux, muxdemux def={Lh=4, Rh=8, NL=1, NB=3, NR=8}}}
```

will generate the following shape (the definition above is already defined in the package):



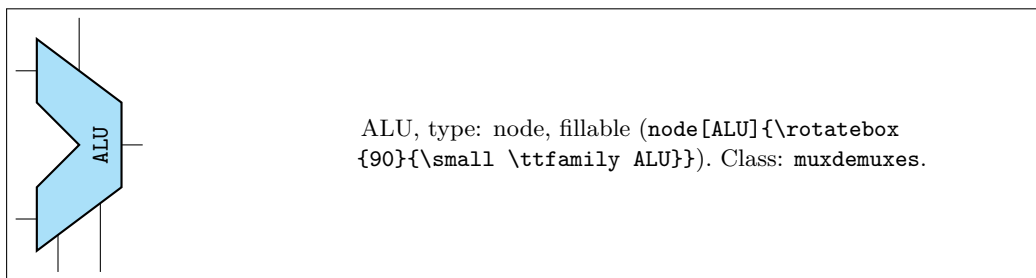
The shape can be also defined with an inset. For example it can be used like this to define a 1-bit adder (also already available):

```
1 \tikzset{one bit adder/.style={muxdemux,  
2     muxdemux def={Lh=4, NL=2, Rh=2, NR=1, NB=1, w=1.5,  
3     inset w=0.5, inset Lh=2, inset Rh=1.5}}}
```



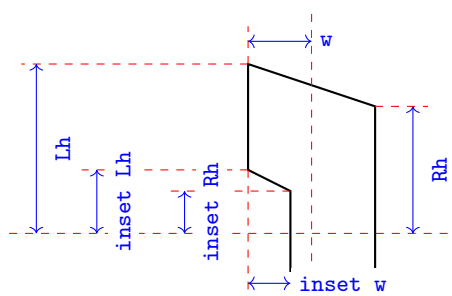
Or a Arithmetic Logic Unit (again, already defined by default):

```
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}}}
```



3.26.1 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.

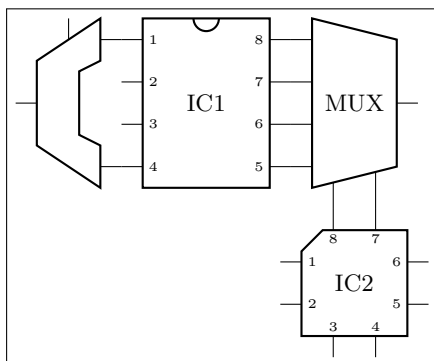


The default values are $Lh = 8$, $Rh = 6$, $w = 3$ and no inset: `inset Lh = inset Rh = inset w = 0`. In addition, you can set the following parameters:

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 numbers 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}

```

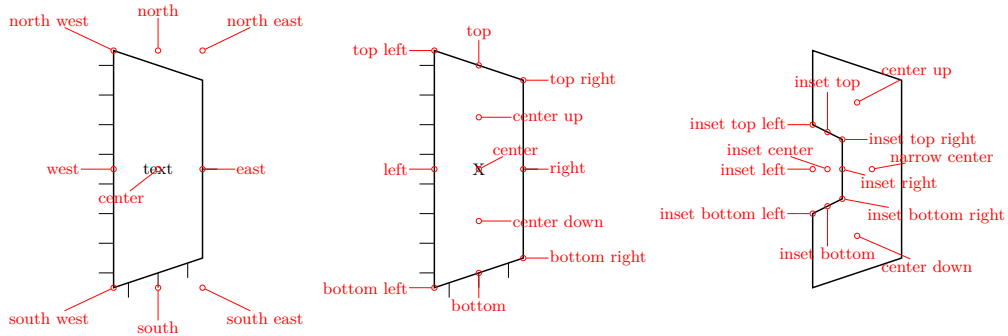
3.26.2 Mux-Demux customization

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`).

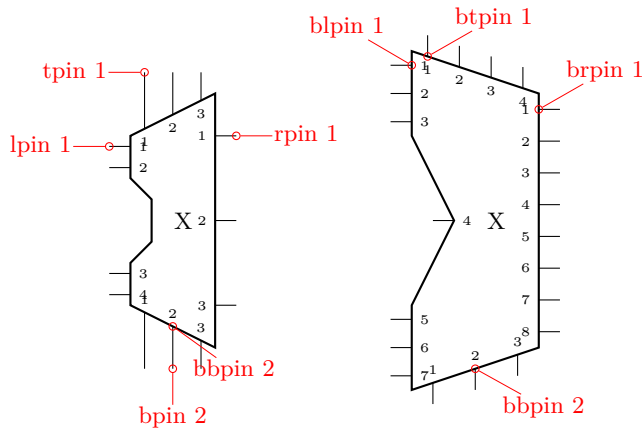
The main difference between setting `external pins width` to 0 or using `no inputs lead` is that in the first case the normal pin anchors and the border anchors will coincide, and in the second case they will not move and stay where they should have been if the leads were drawn.

3.26.3 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.



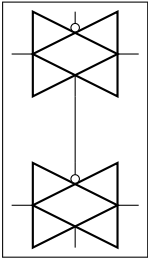
The code that implemented the printing of the numbers (which in `muxdemuxes`, differently from chips, are never printed automatically) in the last graph is the following one.


```

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 }

```

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...



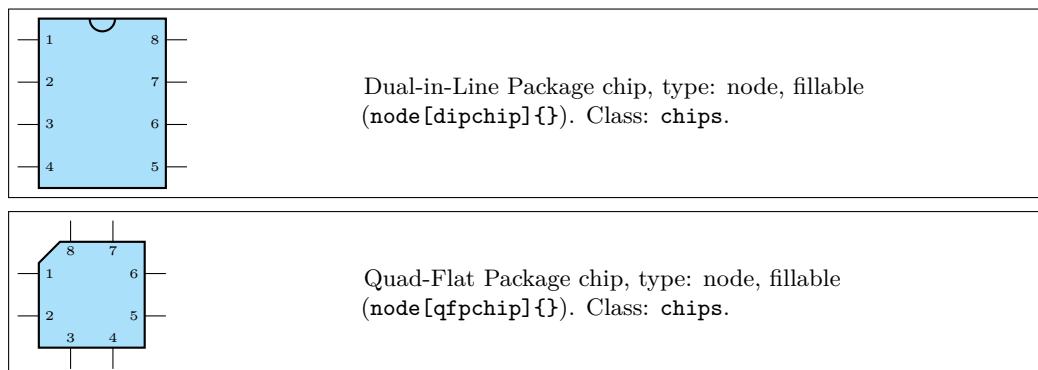
```

1 \def\tgate#1{
2   node[simple triangle, anchor=left, no input leads] (#1-LR){}
3   (#1-LR.right) node[simple triangle, xscale=-1,
4     anchor=left] (#1-RL){}
5   ([yshift=.5ex]#1-RL.btpin 1) node[ocirc]{}
6 \begin{circuitikz}[
7   simple triangle/.style={muxdemux, muxdemux def={
8     NL=1, NR=1, NB=1, NT=1, w=2, Lh=2, Rh=0,
9   }}]
10  \draw (0,0) \tgate{A} (0,-2) \tgate{B};
11  \draw (A-RL.bpin 1) -- (B-RL.tpin 1);
12 \end{circuitikz}

```

3.27 Chips (integrated circuits)

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



3.27.1 DIP and QFP chips customization

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

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

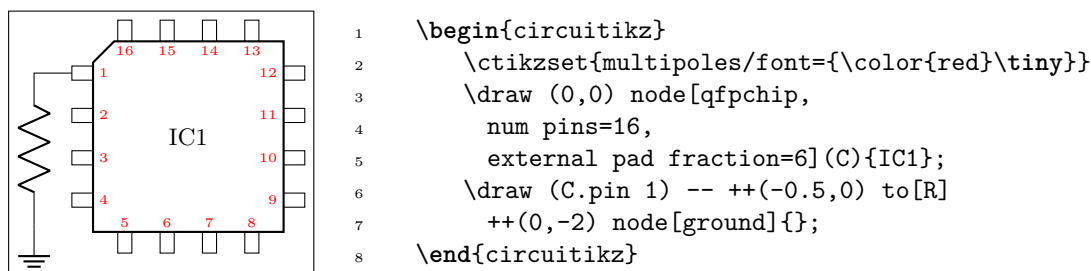
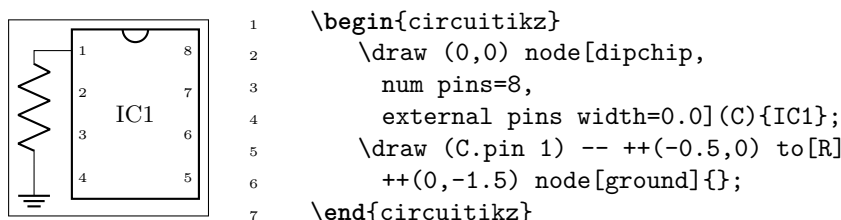
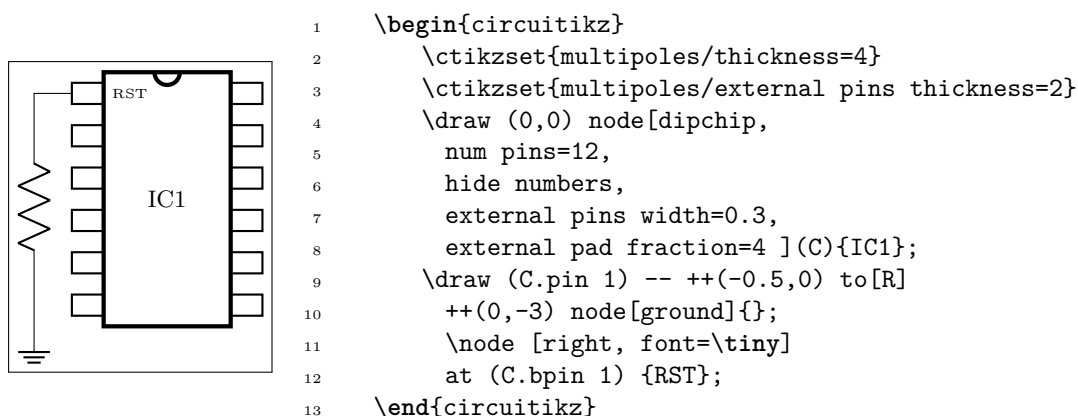
For the QFP chips, you can only chose the pin spacing with `multipoles/qfpchip/pin spacing` key.

The pins of the chip can be “hidden” (that is, just a spot in the border, optionally marked with a number) or “stick out” with a thin lead by setting `multipoles/external pins width` greater than 0 (default value is 0.2, so you’ll have leads as shown above). Moreover, you can transform the thin lead into a pad by setting the key `multipoles/external pad fraction` to something different 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.

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.

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

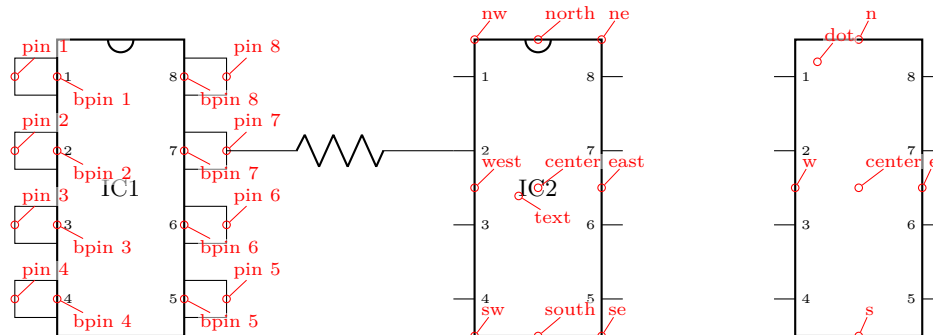
The line thickness of the main shape is controlled by `multipoles/thickness` (default 2) and the one of the external pins/pads with `multipoles/external pins thickness` (default 1).



3.27.2 Chips anchors

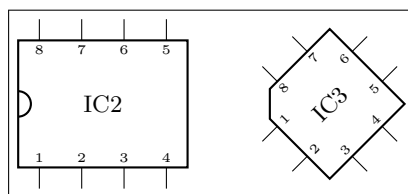
Chips have anchors on pins and global anchors for the main shape. The pin anchors to be used to connect wires to the chip are called `pin 1`, `pin 2`, ..., with just one space between `pin` and the number. Border pin anchors (`bpin 1...`) are always on the box border, and can be used to add numbers or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

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



3.27.3 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.



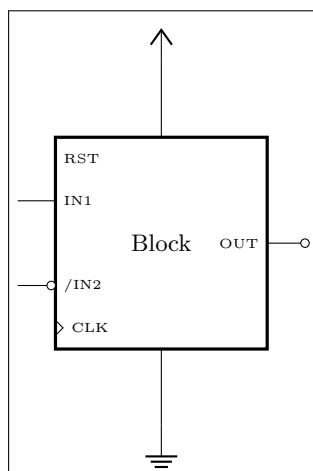
```

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}

```

3.27.4 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}

```

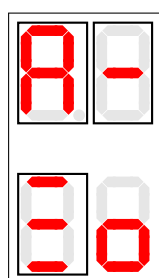
3.28 Seven segment displays



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.²⁶

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.

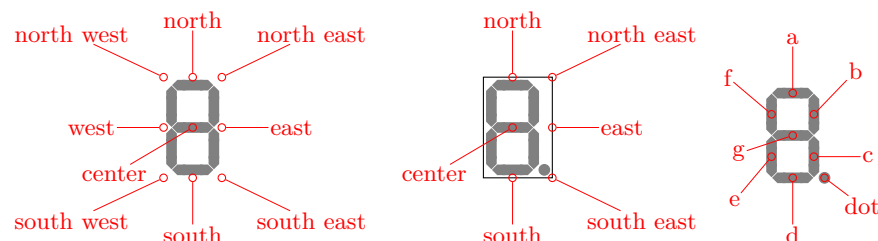
The option `dot` specifies if you want a decimal dot or not. The key `none` will remove the dot and the space it would take; `empty` will not show the dot at all but reserve the space, and `on` or `off` will show the dot in the corresponding state.

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

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

3.28.1 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.



²⁶This component has been loosely inspired by the package `SevenSeg` by Germain Gondor, 2009, see TExample.net.

3.28.2 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"

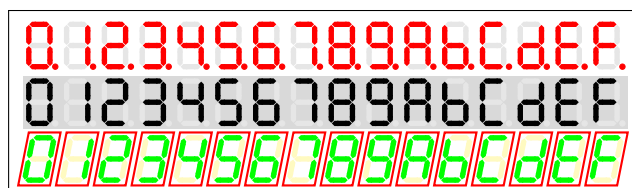
```

A couple of examples are shown below.

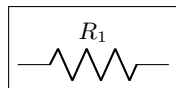
```

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}

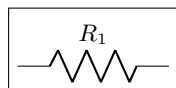
```



4 Labels and similar annotations



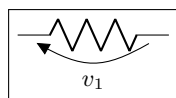
```
1 \begin{circuitikz}
2   \draw (0,0) to[R, l=$R_1$] (2,0);
3 \end{circuitikz}
```



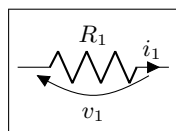
```
1 \begin{circuitikz}
2   \draw (0,0) to[R=$R_1$] (2,0);
3 \end{circuitikz}
```



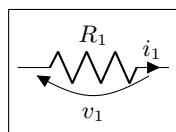
```
1 \begin{circuitikz}
2   \draw (0,0) to[R, i=$i_1$] (2,0);
3 \end{circuitikz}
```



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

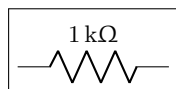


```
1 \begin{circuitikz}
2   \draw (0,0) to[R=$R_1$, i=$i_1$, v=$v_1$] (2,0);
3 \end{circuitikz}
```



```
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:

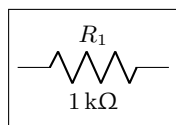


```
1 \begin{circuitikz}\draw
2   (0,0) to[resistor=1<\kilo\ohm>] (2,0)
3 ;\end{circuitikz}
```

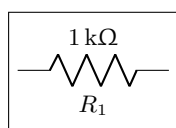
4.1 Labels and Annotations

Since Version 0.7, beside the original label (l) option, there is a new option to place a second label, called annotation (a) at each bipole.

The position of annotations and labels can be adjusted with `_` and `^`.

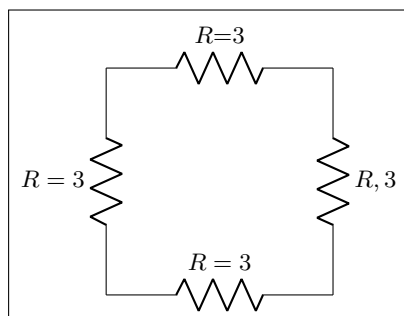


```
1 \begin{circuitikz}
2   \draw (0,0) to[R, l=$R_1$,a=1<\kilo\ohm>] (2,0);
3 \end{circuitikz}
```



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

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

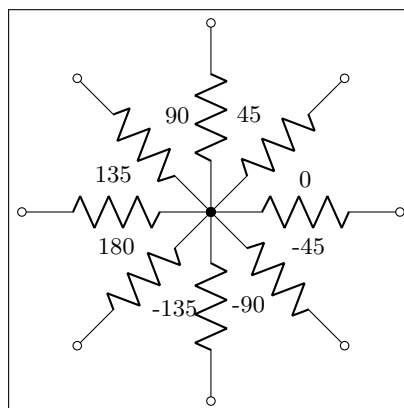


```

1  \def\eq{=}
2  \begin{circuitikz}
3      % the following will fail:
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}

```

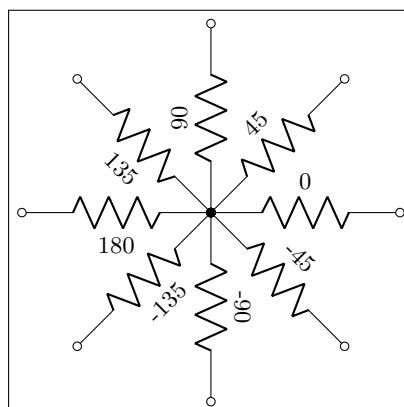
The default orientation of labels is controlled by the options `smartlabels`, `rotatelabels` and `straightlabels` (or the corresponding `label/align` keys). Here are examples to see the differences:



```

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}

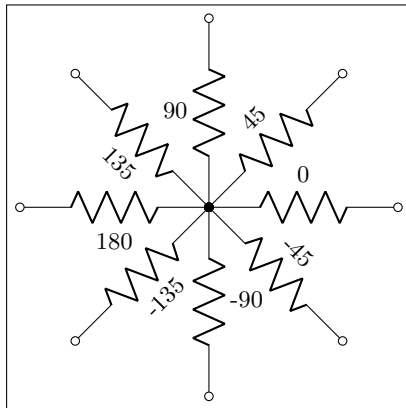
```



```

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}

```

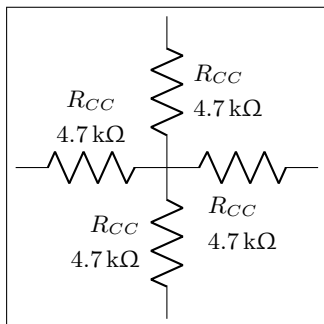


```

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}

```

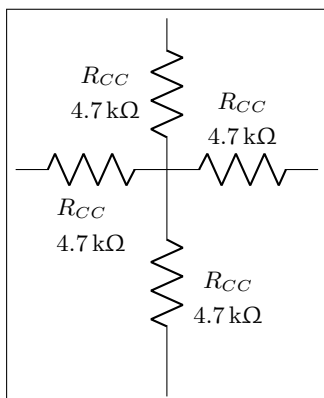
You also can use stacked (two lines) labels. The example should be self-explanatory: the two lines are specified as `l2=`*line1* and *line2*. You can use the keys `l2 halign` to control horizontal position (left, center, right) and `l2 valign` to control the vertical one (bottom, ccenter, top).



```

1 \begin{circuitikz}[ american, ]
2   %
3   % default is l2 halign=l, l2 valign=c
4   %
5   \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm},
6               , l2 valign=t] (2,0);
7   \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm},
8               , ] (0,2);
9   \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, l2 halign=c, l2 valign=b] (-2,0);
10  \draw (0,0) to[R, l2_=$R_{CC}$ and \SI{4.7}{k\ohm}, l2 halign=r, l2 valign=c] (0, -2);
11 \end{circuitikz}

```



```

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

```

4.2 Currents and voltages

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

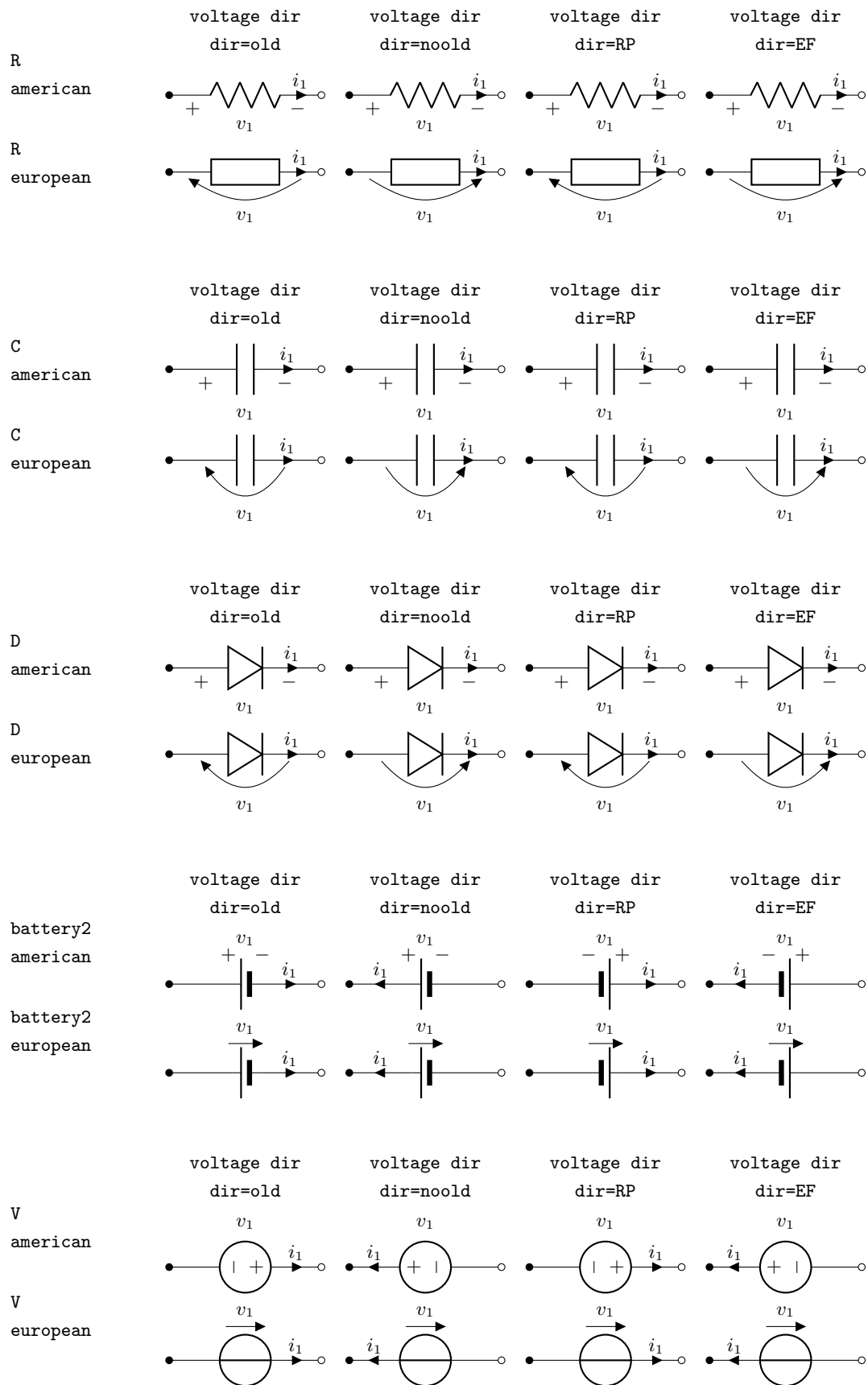
- **oldvoltagedirection**, or the key style **voltage dir=old**: Use old way of voltage direction having a difference between european and american direction, with wrong default labelling for batteries (it was the default before version 0.5);
- **nooldvoltagedirection**, or the key style **voltage dir=noold**: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

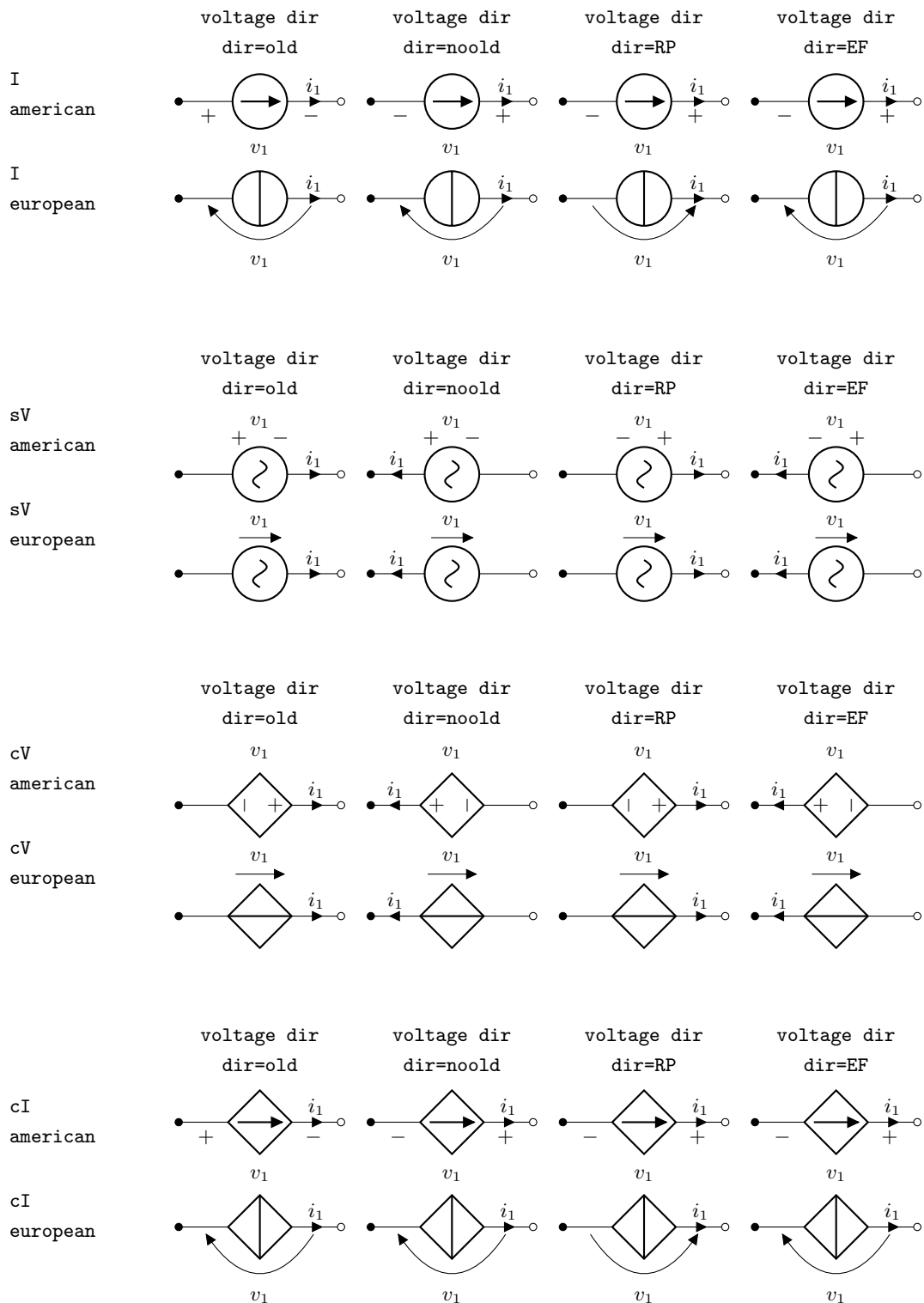
Notice that the four styles are designed to be used at the environment level: that is, you should use them at the start of your environment as in `\begin{circuitikz}[voltage dir=old]` ... and not as a key for single components, in which case the behaviour is not guaranteed.

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

```

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 }
```



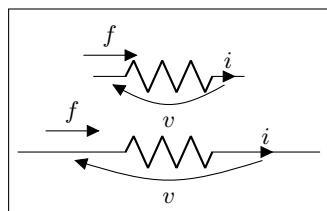


Obviously, you normally use just one between current and flows, but anyway you can change direction of the voltages, currents and flows using the complete keys `i_>`, `i^<`, `i>_`, `i>^`, as shown in the following examples.

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

4.2.1 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.

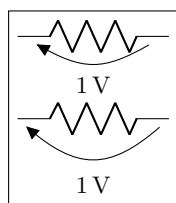


```

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:

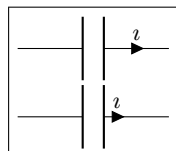


```

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);

```

The same concept as `distance from node` applies to the key `current/distance` for the position of the current's arrow:

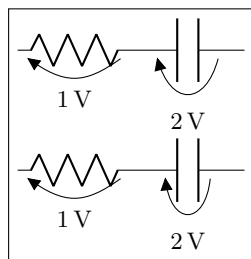


```

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);

```

You can change globally these 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/.
4   initial=.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 kind 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 4.8.

4.3 Currents

Inline (along the wire) currents are selected with $i_{>}$, $i^{<}$, $i_{>}$, $i^{>}$, and various combination; the default position and direction is obtained with the simple key $i=$...

Basically, \wedge 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}
```

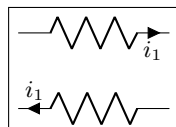


```
1 \begin{circuitikz}
2   \draw (0,0) to[R, i^{<}=$i_1$] (2,0);
3 \end{circuitikz}
```



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

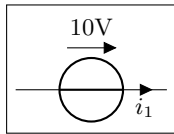
Also notice that the direction of the path is important:



```
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,²⁷ following the chosen global voltage direction strategy (see section 4.2).

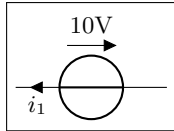
²⁷This, in hindsight, has been a bad feature — and I'm partly responsible for it. But removing it would create *too small* variations in circuits, so it stays.



```

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

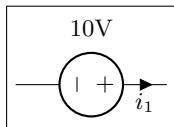
```



```

1 \begin{circuitikz}[voltage dir=EF]
2   \draw (0,0) to[V=10V, i_=$i_1$] (2,0);
3 \end{circuitikz}

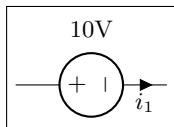
```



```

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

```

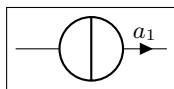


```

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:

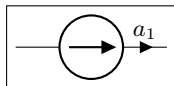


```

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.



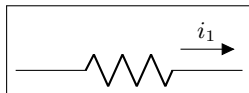
```

1 \begin{circuitikz}[american currents]
2   \draw (0,0) to[I=$a_1$] (2,0);
3 \end{circuitikz}

```

4.4 Flows

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



```

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

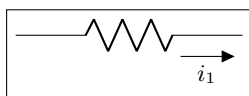
```



```

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

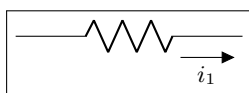
```



```

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

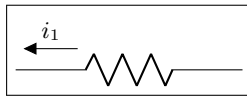
```



```

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

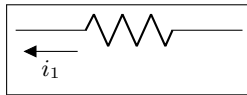
```



```

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

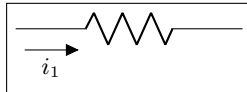
```



```

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

```



```

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

```

4.5 Voltages

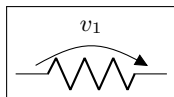
See the introduction at Currents and Voltages (section 4.2, page 135) for the default direction of the voltage and currents.

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 4.3) with the `_<>` symbols.

4.5.1 European style

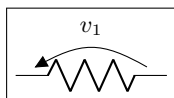
The default, with curved arrows. Use option `europeanvoltage` or style `[european voltages]`, or setting (even locally) `voltage=european`.



```

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

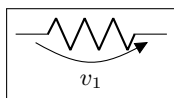
```



```

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

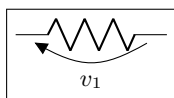
```



```

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

```

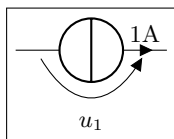


```

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

```

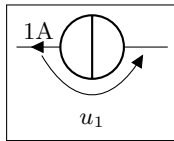
The default direction for active elements can change, depending on the global `voltage dir` setting, so be careful.



```

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

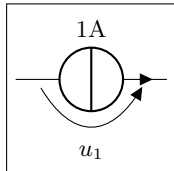
```



```

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

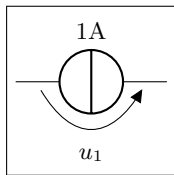
```



```

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

```

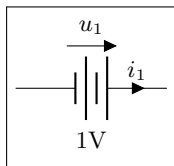


```

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

```

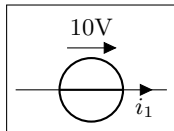
Moreover, for historical reasons, voltage generators have differently looking arrows (they are straight even in curved European style).



```

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

```

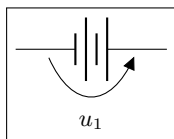


```

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.

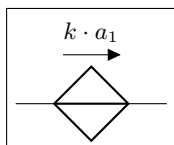


```

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:

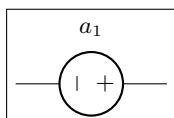


```

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]`.



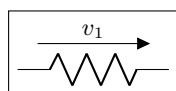
```

1 \begin{circuitikz}[american voltages]
2   \draw (0,0) to[V=$a_1$] (2,0);
3 \end{circuitikz}

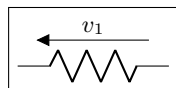
```


4.5.2 Straight European style

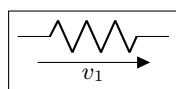
Using straight arrows. Use option `straightvoltages` or style `[straight voltages]`, or setting (even locally) `voltage=straight`.



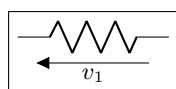
```
1 \begin{circuitikz}[straight voltages]
2   \draw (0,0) to[R, v^>=$v_1$] (2,0);
3 \end{circuitikz}
```



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

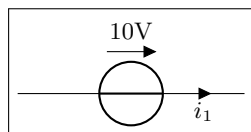


```
1 \begin{circuitikz}[straight voltages]
2   \draw (0,0) to[R, v_>=$v_1$] (2,0);
3 \end{circuitikz}
```

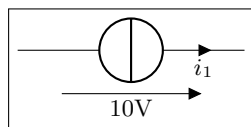


```
1 \begin{circuitikz}[straight voltages]
2   \draw (0,0) to[R, v_<=$v_1$] (2,0);
3 \end{circuitikz}
```

Again, voltage generators are treated differently:

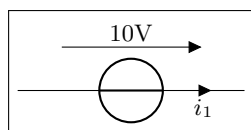


```
1 \begin{circuitikz}[straight voltages]
2   \draw (0,0) to[V=10V, i_=$i_1$] (3,0);
3 \end{circuitikz}
```



```
1 \begin{circuitikz}[straight voltages]
2   \draw (0,0) to[I, v=10V, i_=$i_1$] (3,0);
3 \end{circuitikz}
```

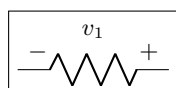
And you can override that with `bipole/is voltage` keeping into account that the default direction will be the one of passive components:



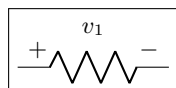
```
1 \begin{circuitikz}[straight voltages]
2   \draw (0,0) to[V=10V, bipole/is voltage=false,
3                 i_=$i_1$] (3,0);
4 \end{circuitikz}
```

4.5.3 American style

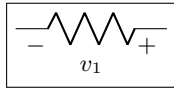
Use option `americanvoltage` or set `[american voltages]` or use the option `voltage=american`.



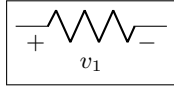
```
1 \begin{circuitikz}[american voltages]
2   \draw (0,0) to[R, v^>=$v_1$] (2,0);
3 \end{circuitikz}
```



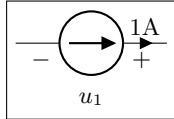
```
1 \begin{circuitikz}[american voltages]
2   \draw (0,0) to[R, v^<=$v_1$] (2,0);
3 \end{circuitikz}
```



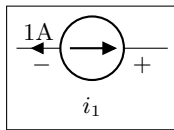
```
1 \begin{circuitikz}[american voltages]
2   \draw (0,0) to[R, v_>=$v_1$] (2,0);
3 \end{circuitikz}
```



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



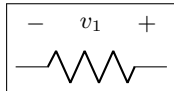
```
1 \begin{circuitikz}[american]
2   \draw (0,0) to[I=1A, v_=$u_1$] (2,0);
3 \end{circuitikz}
```



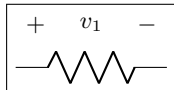
```
1 \begin{circuitikz}[american]
2   \draw (0,0) to[I<=1A, v_=$i_1$] (2,0);
3 \end{circuitikz}
```

4.5.4 Raised American style

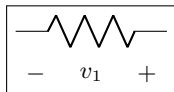
Since version 1.2.1, “raised” American voltages are available; to use them, set the style `[raised voltages]` or use the option `voltage=raised`. This is a version of the American-style voltage where the signs are raised to the level of the label. The label is centered between the two signs, and the position of the signs is calculated 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 4.2.1).



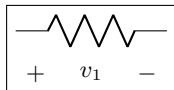
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,0) to[R, v^>=$v_1$] (2,0);
3 \end{circuitikz}
```



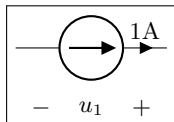
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,0) to[R, v^<=$v_1$] (2,0);
3 \end{circuitikz}
```



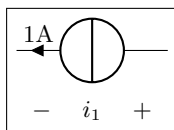
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,0) to[R, v_>=$v_1$] (2,0);
3 \end{circuitikz}
```



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



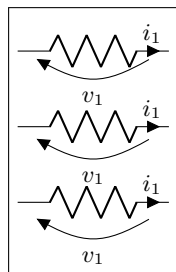
```
1 \begin{circuitikz}[american]
2   \ctikzset{voltage=raised}
3   \draw (0,0) to[I=1A, v_=$u_1$] (2,0);
4 \end{circuitikz}
```



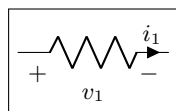
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,0) to[I<=1A, v_=$i_1$] (2,0);
3 \end{circuitikz}
```

4.5.5 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):

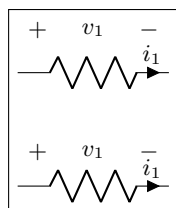


```
1 \begin{circuitikz}[]
2   \draw (0,0) to[R, v=$v_1$, i=$i_1$] (2,0);
3   \draw (0,-1) to[R, v=$v_1$, i=$i_1$,
4     voltage shift=0.5] (2,-1);
5   \draw (0,-2) to[R, v=$v_1$, i=$i_1$,
6     voltage shift=1.0] (2,-2);
7 \end{circuitikz}
```



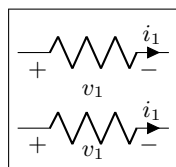
```
1 \begin{circuitikz}[american voltages, voltage shift=0.5]
2   \draw (0,0) to[R, v=$v_1$, i=$i_1$] (2,0);
3 \end{circuitikz}
```

Negative values do work as expected:



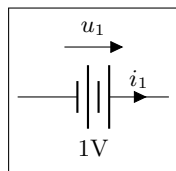
```
1 \begin{circuitikz}[raised voltages]
2   \draw (0,1.5) to[R, v^=$v_1$, i=$i_1$] ++(2,0);
3   \draw (0,0) to[R, v^=$v_1$, i=$i_1$,
4     voltage shift=-1.0] ++(2,0);
5 \end{circuitikz}
```

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.

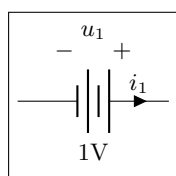


```
1 \begin{circuitikz}[american voltages]
2   \draw (0,1) to[R, v=$v_1$, i=$i_1$] ++(2,0);
3   % normally 1.4, make it tighter
4   \ctikzset{voltage/american label distance=0.5}
5   \draw (0,0) to[R, v=$v_1$, i=$i_1$] ++(2,0);
6 \end{circuitikz}
```

Notes that `american voltage` also affects batteries.

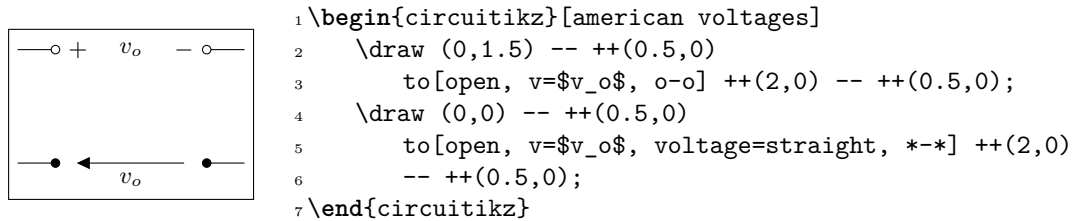


```
1 \begin{circuitikz}[voltage shift=0.5]
2   \draw (0,0) to[battery, l=1V, v=$u_1$, i=$i_1$] (2,0);
3 \end{circuitikz}
```



```
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}
```

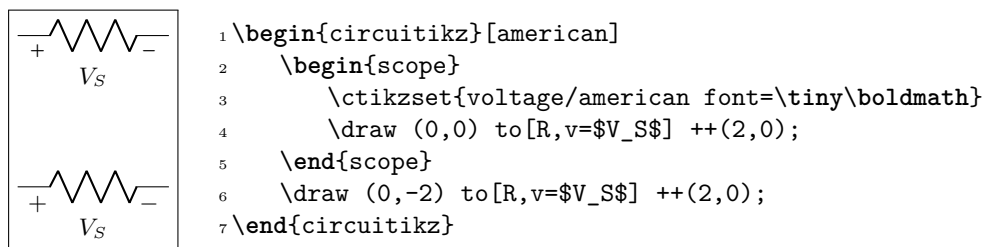
When using `american` or `straight` voltage style, the `open` component is treated differently, and the voltage is placed in the middle of the open space²⁸:



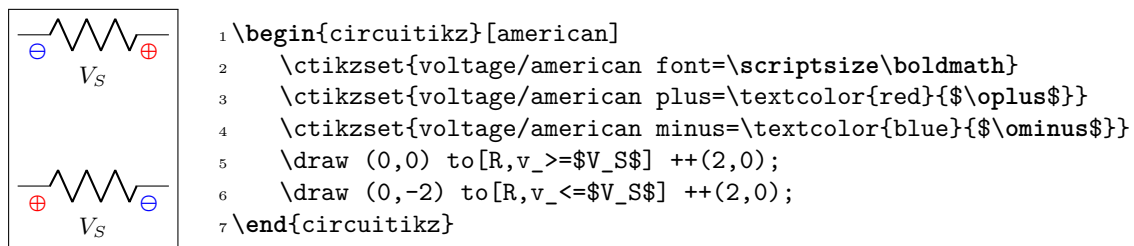
If you want or need to maintain the old behavior for `open` voltage, you can set the key `american open voltage` to `legacy` (the default is the new behavior, which correspond to the value `center`).

4.5.6 American voltages customization

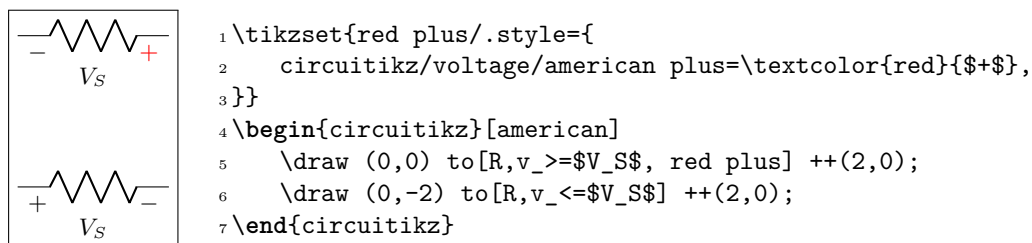
Since 0.9.0, you can change the font²⁹ used by the `american voltages` style, by setting to something different from nothing the key `voltage/american font` (default: nothing, using the current font) style:



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



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



²⁸Since v1.1.2, thank to an [issue opened by user rhandley on GitHub](#).

²⁹There 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!

4.6 Changing the style of labels and 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 are 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:

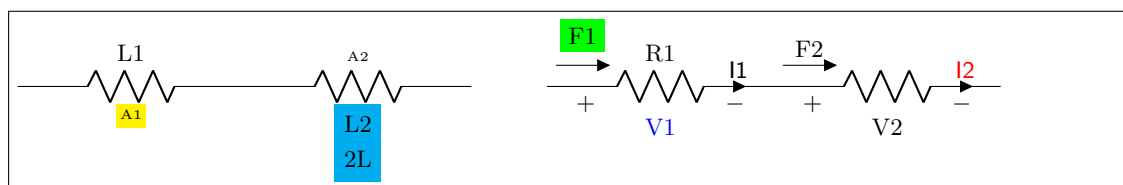
```
...to[bipole annotation style={color=red}, R, a={Red note}]...
```

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 effect³⁰.

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.



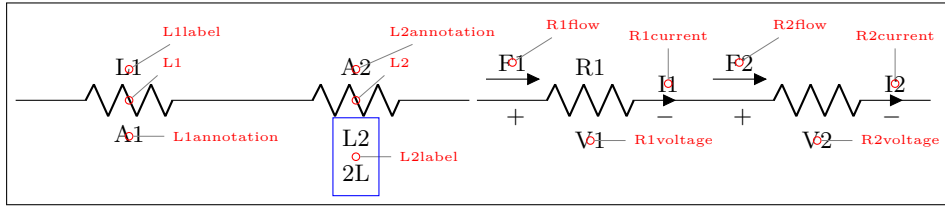
```
1 \begin{circuitikz}[american]
2   \ctikzset{bipole annotation style/.style={font=\tiny}}
3   \ctikzset{bipole current style/.style={font=\small\sffamily}}
4   \draw (0,0) to [bipole annotation append style={fill=yellow}, R=L1, a=A1] ++(3,0)
5     to [bipole label style={fill=cyan}, R, l2=L2 and 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}
```

4.7 Accessing labels text nodes

Since 0.9.5, you can access all the labels nodes³¹ using special node names. So, if you use `name` to give a name to the bipole node, you can access also 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.

³⁰No, I do not know why. Hints and fixes are welcome.

³¹The access to `labels` and `annotations` was present before, but not documented.



```

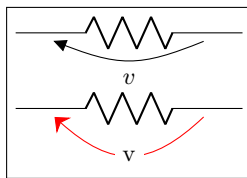
1 \newcommand{\marknode}[2][45]{%
2   \node[circle, draw, red, inner sep=1pt,
3     pin={red, font=\tiny}#1:#2] at (#2.center) {};
4 }
5 \begin{circuitikz}[american]
6   \draw (0,0) to [R=L1, a=A1, name=L1] ++(3,0)
7     to [R, l2_=L2 and 2L, a^=A2, name=L2] ++(3,0);
8   \marknode{L1} \marknode{L1label} \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}

```

4.8 Advanced voltages, currents and flows

Since version 1.2.1, 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 arrows³². Access to the anchors allows to do all this things, and more.

For example, you can do something like this:

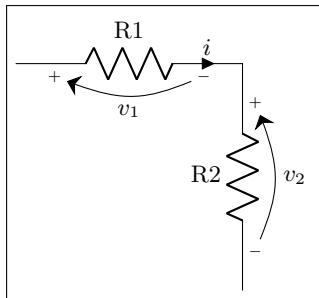


```

1 \begin{circuitikz}[]
2   \draw (0,1) to[R, v=$v$] ++(3,0);
3   \draw (0,0) to[R, v, name=R, voltage/bump b=3] ++(3,0);
4   \draw [thin, red, -{Stealth[width=8pt]}, ]
5     (R-Vfrom) .. controls (R-Vcont1) and (R-Vcont2).. (R-Vto)
6     node [black, pos=0.5, fill=white]{v};
7 \end{circuitikz}

```

Or, for example, to have a different voltage style; normally you would define a macro:



```

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}

```

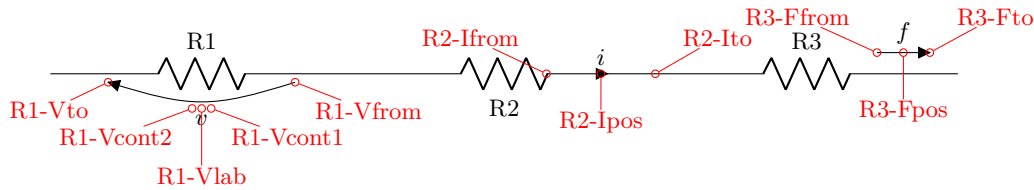
³²in regular voltages, the arrows are not real TikZ arrows, but the auxiliary arrow shapes of CircuiTikZ

4.8.1 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);
```

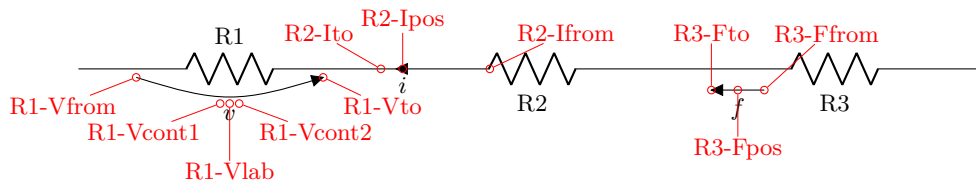


The meaning of the anchors is the following:

- **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`.³³

Changing the options of the elements, will change the anchors accordingly:

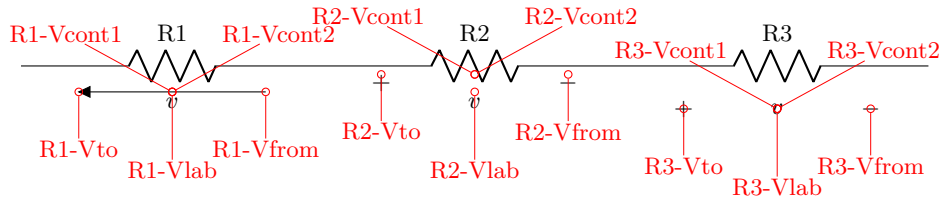
```
1 \ctikzset{current/distance=0.2}
2 \draw (0,0) to[R=R1, v>=$v$, name=R1, voltage=straight] ++(4,0)
3   to[R, l_=R2, i<=$i$, name=R2] ++(4,0)
4   to[R, l_=R3, f<=$f$, name=R3] ++(4,0);
```



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);
```

³³In 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.



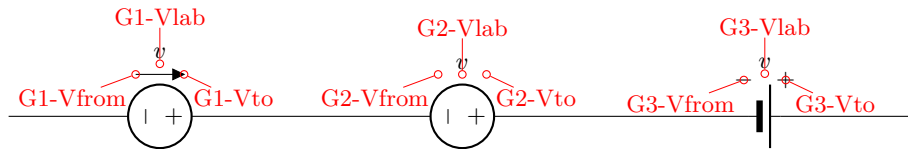
Notice the position of the control points, as well as the fact that the anchor available with `\ctikzgetanchor` is applied to `Vfrom` and `Vto` symbols, too.

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);

```

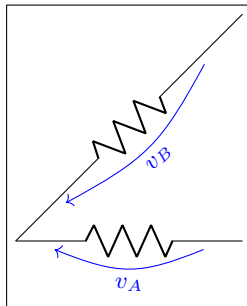


4.8.2 Auxiliary information

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* *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.

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}

```

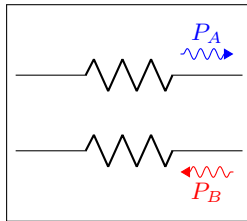
Or you could use the anchor to substitute the flow with a fancy one and still position automatically the label; suppose you have the following definition in your preamble (see TikZ manual, “Path decorations”):

```

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:



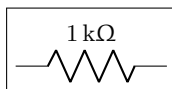
```

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}

```

4.9 Integration with siunitx

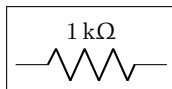
If the option `siunitx` is active (and *not* in ConT_EXt), then the following are equivalent:



```

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

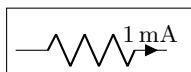
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, l=${\SI{1}{\kilo\ohm}}$] (2,0);
3 \end{circuitikz}

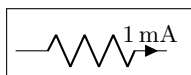
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, i=1<\milli\ampere>] (2,0);
3 \end{circuitikz}

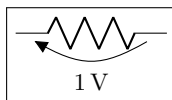
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, i=${\SI{1}{\milli\ampere}}$] (2,0);
3 \end{circuitikz}

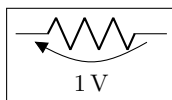
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[R, v=1<\volt>] (2,0);
3 \end{circuitikz}

```



```

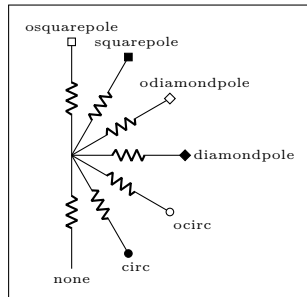
1 \begin{circuitikz}
2   \draw (0,0) to[R, v=${\SI{1}{\volt}}$] (2,0);
3 \end{circuitikz}

```

5 Using bipoles in circuits

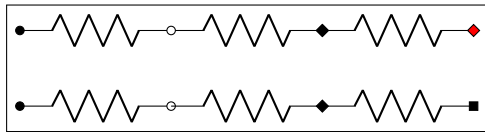
5.1 Nodes (also called poles)

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`³⁴ (see section 3.15).



```
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}
```

³⁴You 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}
```

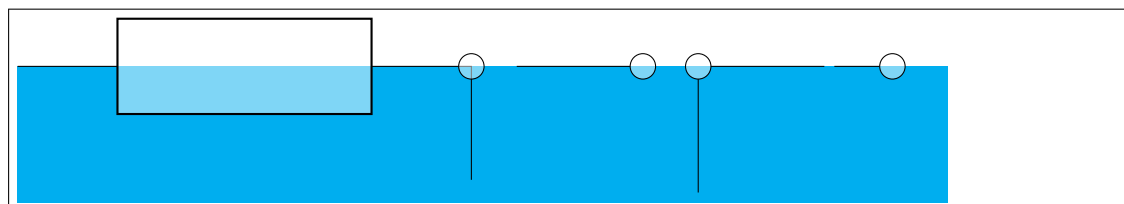


```
1 \begin{circuitikz}
2   \draw (0,0) to[R, d-*] (2,0);
3 \end{circuitikz}
```

5.1.1 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.



```

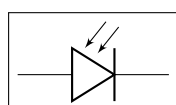
1 \begin{circuitikz}[scale=3, transform shape]
2   \fill[cyan] (0,0) rectangle (4.1,-0.6);
3   \tikzset{open poles opacity=0.5}
4   % automatic positioning when opacity is not 1.0 creates artifacts
5   % note that opacity must go on the draw command for path-style components
6   \draw[fill opacity=0.5] (0,0) to[generic, fill=white, -o] ++(2,0) --++(0,-0.5);
7   % you have to use manual positioning
8   \draw (2.2,0) -- ++(0.5,0) node[ocirc, anchor=180, fill opacity=0.5]{};
9   \draw (3,0) node[ocirc, fill opacity=0.5](B){} (B.0) --++(0.5,0) (B.-90)
10    --++(0,-0.5);
11   % maybe really useful only for terminals going out of the circuit...
12   % notice that in node commands you can specify the opacity directly
13   \draw (3.6,0) -- ++(0.2,0) node[ocirc, fill=white, fill opacity=0.5, anchor=180]{};
14 \end{circuitikz}

```

You also have the similar keys for the “full” poles (albeit they are probably not useful at all).

5.2 Mirroring and Inverting

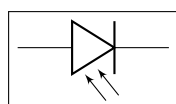
Bipole paths can also mirrored and inverted (or reverted) to change the drawing direction.



```

1 \begin{circuitikz}
2   \draw (0,0) to[pD] (2,0);
3 \end{circuitikz}

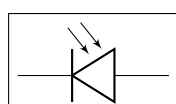
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[pD, mirror] (2,0);
3 \end{circuitikz}

```

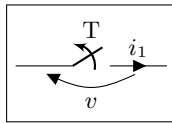


```

1 \begin{circuitikz}
2   \draw (0,0) to[pD, invert] (2,0);
3 \end{circuitikz}

```

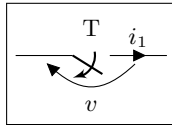
Placing labels, currents and voltages works also, please note, that mirroring and inverting does not influence the positioning of labels and voltages. Labels are by default above/right of the bipole and voltages below/left, respectively.



```

1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}

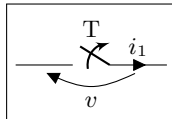
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T, mirror, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}

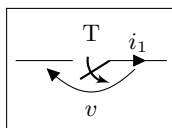
```



```

1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T, invert, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}

```

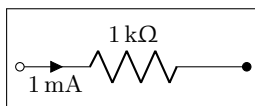


```

1 \begin{circuitikz}
2   \draw (0,0) to[ospst=T,mirror,invert, i=$i_1$, v=$v$] (2,0);
3 \end{circuitikz}

```

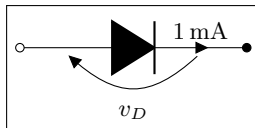
5.3 Putting them together



```

1 \begin{circuitikz}
2   \draw (0,0) to[R=1<\kilo\ohm>,
3     i>_1<\milli\ampere>, o-*] (3,0);
4 \end{circuitikz}

```



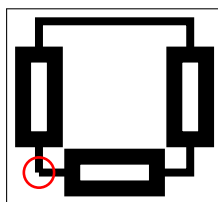
```

1 \begin{circuitikz}
2   \draw (0,0) to[D*, v=$v_D$,
3     i=1<\milli\ampere>, o-*] (3,0);
4 \end{circuitikz}

```

5.4 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!):

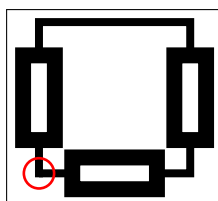


```

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}

```

6 Colors

Color support in CircuiTikZ is quite limited. You will have no problem if:

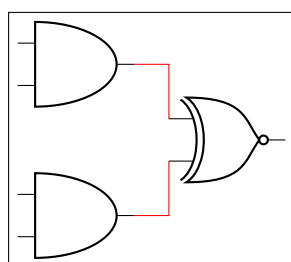
1. You stick to use styles (see 3.3.2) for filling your components, or using a direct `fill=...` option directly;
2. when coloring whole circuits, use the option `color=...` in your global picture options or in the `\draw` command (not just the color name as a shorthand);
3. forget about transparency.

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.

6.1 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. If you change color in the picture, please do not use just the color name as a style, like `[red]`, but rather assign the style `[color=red]`.

Compare for instance

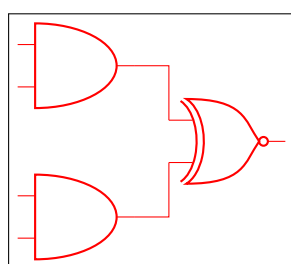


```

1 \begin{circuitikz} \draw[red]
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}

```

and

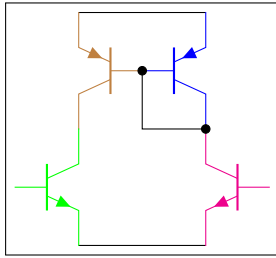


```

1 \begin{circuitikz} \draw[color=red]
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}

```

One can of course change the color *in medias res*:

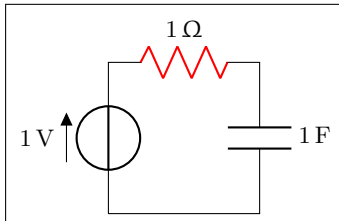


```

1 \begin{circuitikz} \draw
2   (0,0) node[pnp, color=blue] (pnp2) {}
3   (pnp2.B) node[pnp, xscale=-1, anchor=B, color=brown] (pnp1) {}
4   (pnp1.C) node[npn, anchor=C, color=green] (npn1) {}
5   (pnp2.C) node[npn, xscale=-1, anchor=C, color=magenta] (npn2) {}
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. Also, please notice the curly braces around the to:

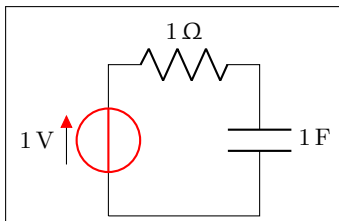


```

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}

```

Which, for some bipoles, can be frustrating:

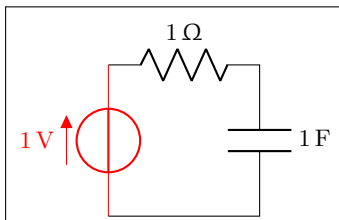


```

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}

```

The only way out is to specify different paths:



```

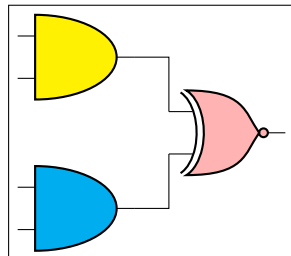
1 \begin{circuitikz} \draw[color=red]
2   (0,0) to[V=1<\volt>, color=red] (0,2);
3   \draw (0,2) to[R=1<\ohm>] (2,2)
4   to[C=1<\farad>] (2,0) -- (0,0)
5 ;\end{circuitikz}

```

And yes: this is a bug and *not* a feature...

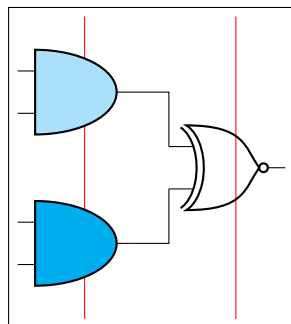
6.2 Fill colors

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:



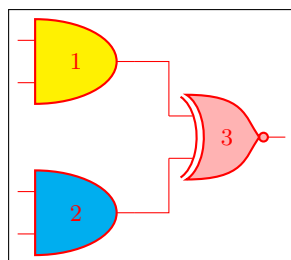
```
1 \begin{circuitikz} \draw
2   (0,2) node[and port, fill=yellow] (myand1) {}
3   (0,0) node[and port, fill=cyan] (myand2) {}
4   (2,1) node[xnor port, fill=red!30!white] (myxnor) {}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 ;\end{circuitikz}
```

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 it should be an exceptional thing anyway:



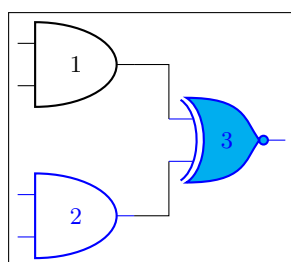
```
1 \begin{circuitikz}
2   \ctikzset{logic ports/fill=cyan!30!white}
3   \draw[red] (-0.5,3) -- (-0.5, -1);
4   \draw[red] (1.5,3) -- (1.5, -1);
5   \draw
6   (0,2) node[and port, ] (myand1) {}
7   (0,0) node[and port, fill=cyan] (myand2) {}
8   (2,1) node[xnor port, circuitikz/logic ports/fill=none] (
9     myxnor) {}
10  (myand1.out) -| (myxnor.in 1)
11  (myand2.out) -| (myxnor.in 2)
12 ;\end{circuitikz}
```

You can combine shape colors with fill colors, too, but you should use the `draw` color option style for this:



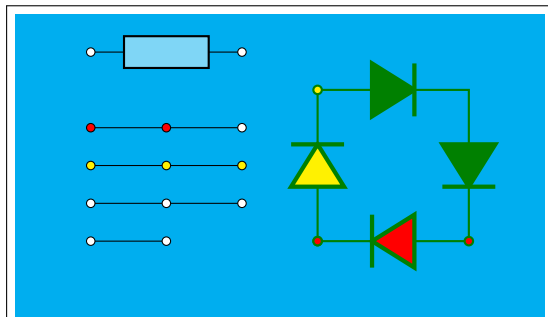
```
1 \begin{circuitikz} \draw[color=red]
2   (0,2) node[and port, fill=yellow] (myand1) {1}
3   (0,0) node[and port, fill=cyan] (myand2) {2}
4   (2,1) node[xnor port, fill=red!30!white] (myxnor) {3}
5   (myand1.out) -| (myxnor.in 1)
6   (myand2.out) -| (myxnor.in 2)
7 ;\end{circuitikz}
```

This is because, as you can see from the following example in port 2, you can't specify both a fill and a color in the node (yes, it's a bug too, but it's quite complex to solve given the current circuitTikZ architecture). A workaround is shown in port 3:



```
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)
4     {2}
5   (2,1) {[color=blue] node[xnor port, fill=cyan] (myxnor
6     ) {3}}
7   (myand1.out) -| (myxnor.in 1)
8   (myand2.out) -| (myxnor.in 2)
9 ;\end{circuitikz}
```


Notice also that the connection point are always filled, although the color *tries* to follow the color of the filling of the component (but look at section 5.1.1). 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=yellow]
9     ++(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.

7 FAQ

Q: When using `\tikzexternalize` I get the following error:

! Emergency stop.

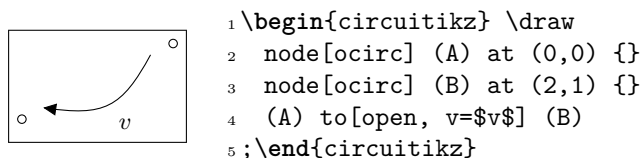
A: The TikZ manual states:

Furthermore, the library assumes that all \LaTeX pictures are ended with `\end{tikzpicture}`.

Just substitute every occurrence of the environment `circuitikz` with `tikzpicture`. They are actually pretty much the same.

Q: How do I draw the voltage between two nodes?

A: Between any two nodes there is an open circuit!



Q: I cannot write `to[R = $R_1=12V$]` nor `to[ospst = open, 3s]`: I get errors.

A: It is a limitation of the parser.

Use `\def{\eq}{=}` to `to[R = $R_1\eq 12V$]` and `to[ospst = open{,} 3s]` instead; see caveat in section 4.1.

Q: I tried to change the direction of the y axis with `yscale=-1`, but the circuit is completely messed up.

A: Yes, it's a known bug (or misfeature, or limitation). See section 1.7. Don't do that.

Q: I tried to put a diode in a `pic`, but it's coming out badly rotated.

A: Yes, it's a known bug (or misfeature, or limitation). See section 1.7. CircuiTikZ is not compatible with `pics` at this point.

8 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.³⁵

Big fat warning: this material is reserved to 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.

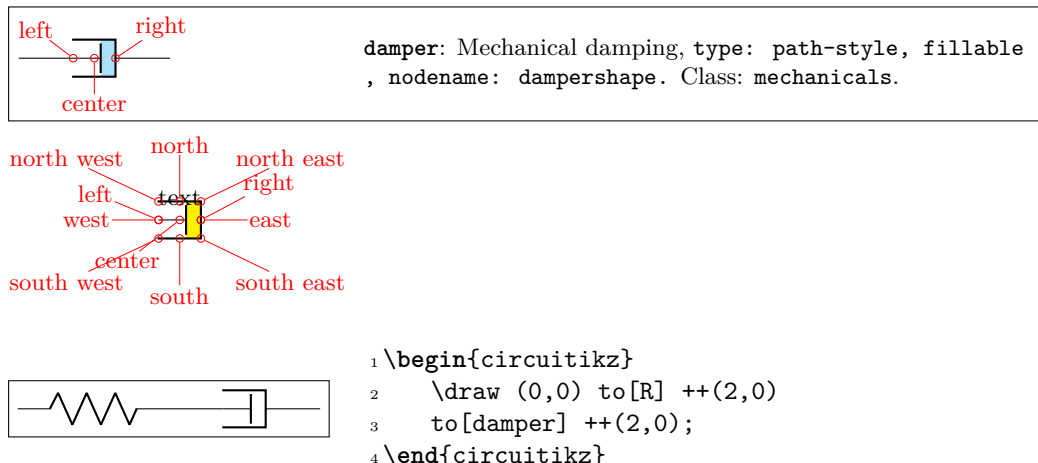
8.1 Suggested setup

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}
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{LTexample}[varwidth]
25 \begin{circuitikz}
26     \draw (0,0) to[R] ++(2,0)
27         to[damper] ++(2,0);
28 \end{circuitikz}
29 \end{LTexample}
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):

³⁵<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.

8.2 Path-style component

Let's define for example a path style component, like the one suggested by the user @alex on [TeX stack-exchange site](#). The component will be a mix of the **damper** and the **spring** components already present.

The first step is to check if we can use the definition already existing for similar elements (for coherence of size) or if we need to define new ones; for this you have to check the file `pgfcirc.defines.tex`: we find

```

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).

To define the new component we will look into `pgfcircbipoles.tex` and we will copy, for example, the definition of the damper into our code, just changing the name:

```

1 %% mechanical resistor - damper
2 \pgfcircdeclarebipolescaled{mechanicals}
3 {}
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@right}{\pgf@circ@res@up}}
10  \pgf@circ@maybefill
11
12  % line into the damper

```

```

13 \pgfpathmoveto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@zero}}
14 \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
    pgf@circ@res@right}
15     {\pgf@circ@res@zero}}
16 \pgfusepath{stroke}
17
18 % damper box
19 \pgf@circ@setlinewidth{bipoles}{\pgfstartlinewidth}
20 \pgfpathmoveto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@down}}
21 \pgfpathlineto{\pgfpoint{\pgf@circ@res@right}{\pgf@circ@res@down}}
22 \pgfpathlineto{\pgfpoint{\pgf@circ@res@right}{\pgf@circ@res@up}}
23 \pgfpathlineto{\pgfpoint{\pgf@circ@res@left}{\pgf@circ@res@up}}
24
25 \pgfsetrectcap
26 \pgfsetmiterjoin
27 \pgfusepath{stroke}
28
29 % damper vertical element
30 \pgfpathmoveto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
    pgf@circ@res@right}
31     {.8\pgf@circ@res@down}}
32 \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}\
    pgf@circ@res@right}
33     {.8\pgf@circ@res@up}}
34 \pgfsetbuttcap
35 \pgfusepath{stroke}
36
37 }

```

This command will define a shape that is named `viscoeshape`, with all the correct geographical anchors based on the depth, height and width defined in the parameters of `\pgfcircdeclarebipolescaled`. Moreover, the element is assigned to the class `mechanicals` for styling.

To be coherent with the styling, you should use (when needed) the length `\pgf@circ@scaled@Rlen` as the “basic” length for drawing, using the fill functions defined at the start of `pgfcirc.defines.tex` to fill and stroke — so that the operation will follow the style parameters and, finally, use the macro `\pgf@circ@setlinewidth` to set the line thickness /the first argument is the “legacy” class, if you do not want to assign one you can use the pseudo-legacy class `none`.

The anchors for the bipole (which then set the lengths `\pgf@circ@res@left`) are already scaled for your use. You can use these 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 with (this commands are normally in `pgfcircpath.tex`):

```

1 \def\pgf@circ@viscoe@path#1{\pgf@circ@bipole@path{viscoe}{#1}}
2 \compattikzset{viscoe/.style = {\circuitikzbasekey,
3     /tikz/to path=\pgf@circ@dviscoe@path, l=#1}}

```

And now you can show it with:

```

1 \circuitdescrip*{viscoe}{Mechanical viscoelastic element}{\left/135/0.2,
    right/45/0.2, center/-90/0.3}
2
3 \geolrcoord{viscoeshape, fill=yellow}
4
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, the `\pgfcircdeclarebipole` will setup the lengths `\pgfcirc@res@right` and `\pgfcirc@res@up` as the x - y coordinates of the upper right corner, and `\pgfcirc@res@left` and `\pgfcirc@res@down` as the x - y coordinates of the lower left corner of your shape. The `center` coordinate is usually at $(0pt, 0pt)$.

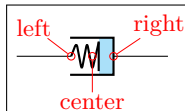
Looking at the implementation of the `spring` element, a possible implementation is changing the lines between lines 12 and 16 with:

```

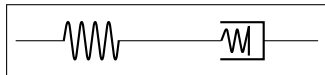
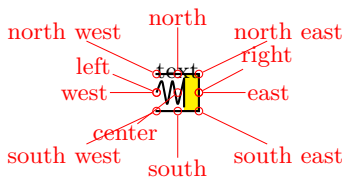
1  % spring into the damper
2  \pgfscope
3    \pgfpathmoveto{\pgfpoint{\pgfcirc@res@left}{\pgfcirc@res@zero}}
4    \pgf@circ@setlinewidth{bipoles}{\pgfstartlinewidth}
5    \pgfsetcornersarced{\pgfpoint{.25\pgfcirc@res@up}{.25\pgfcirc@res@up}}
6    \pgfpathlineto{\pgfpoint{.75\pgfcirc@res@left}{.75\pgfcirc@res@up}}
7    \pgfpathlineto{\pgfpoint{.5\pgfcirc@res@left}{-.75\pgfcirc@res@up}}
8    \pgfpathlineto{\pgfpoint{.25\pgfcirc@res@left}{.75\pgfcirc@res@up}}
9    \pgfpathlineto{\pgfpoint{0pt}{-.75\pgfcirc@res@up}}
10   \pgfpathlineto{\pgfpoint{\ctikzvalof{bipoles/damper/length}}{\pgfcirc@res@right}{.75\pgfcirc@res@up}}
11   \pgfusepath{stroke}
12 \endpgfscope

```

which leads to:



`viscoe`: Mechanical viscoelastic element, type: path-style, fillable, nodename: viscoeshape. Class: mechanicals.

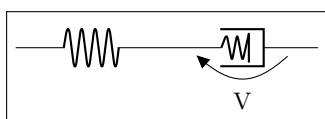


```

1 \begin{circuitikz}
2   \draw (0,0) to[spring] ++(2,0)
3     to[viscoe] ++(2,0);
4 \end{circuitikz}

```

Now you can check if the voltage labels are correct for your new component:

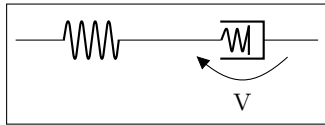


```

1 \begin{circuitikz}[]
2   \draw (0,0) to[spring] ++(2,0)
3     to[viscoe, v=V] ++(2,0);
4 \end{circuitikz}

```

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).

As a final note, notice that the `viscoe` element is already added to the standard package.

8.3 Node-style component

Adding a node-style component is much more straightforward. Just define it by following examples in, for example, `pgfcirctripoles.tex` or the other files; be careful that you should define all the geographical anchors of the shape if you want that the 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, in the drawing part, you should start with:

```

1   \pgf@circ@scaled@Rlen=\scaledRlen

```

and then use `\pgf@circ@scaled@Rlen` (or the anchors) as default lengths while you draw it.

8.3.1 Finishing your work

Once you have a satisfactory element, you should

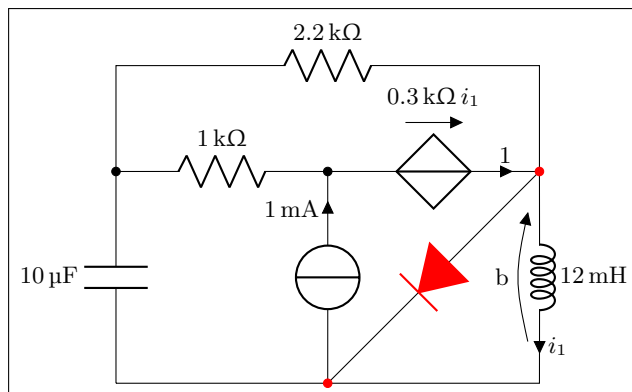
- Clean up your code;
- write a piece of documentation explaining its use, with an example;
- Propose the element for inclusion in the GitHub page of the project (you will have to license this as explained in that page, of course).

The best way of contributing is forking the project, adding your component in the correct files, modifying the manual and creating a pull request for the developers to merge. Anyway, if this is a problem, just open an issue and someone (when they have time...) will answer.

9 Examples

Here a series of example, contributed by several people, is shown with their code.

9.1 A red diode

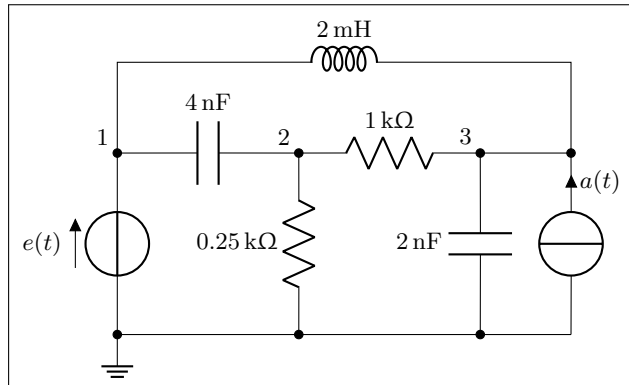


```

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}

```

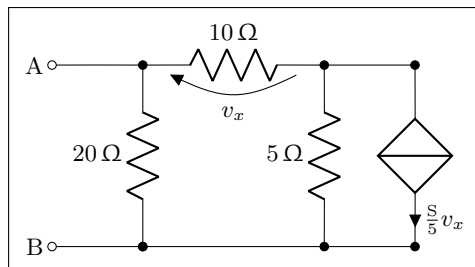

9.2 Using the (experimental) siunitx syntax



```

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}

```

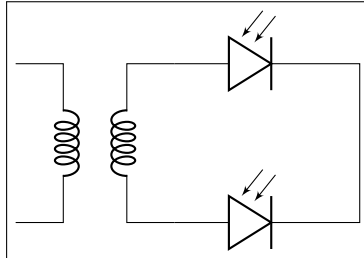


```

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=$\frac{\siemens}{5} v_x$, *-] (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}

```

9.3 Photodiodes

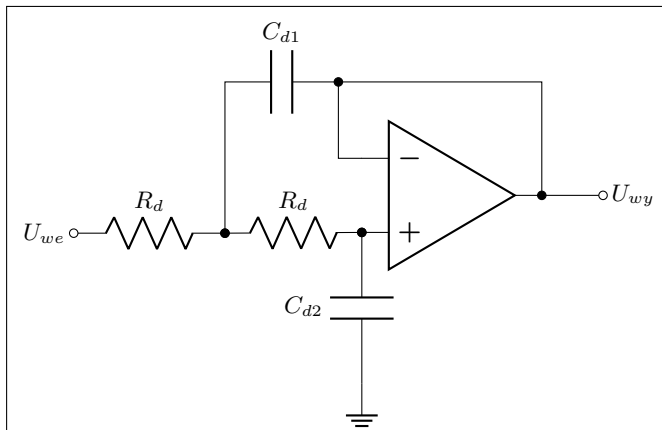


```

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}

```

9.4 A Sallen-Key cell

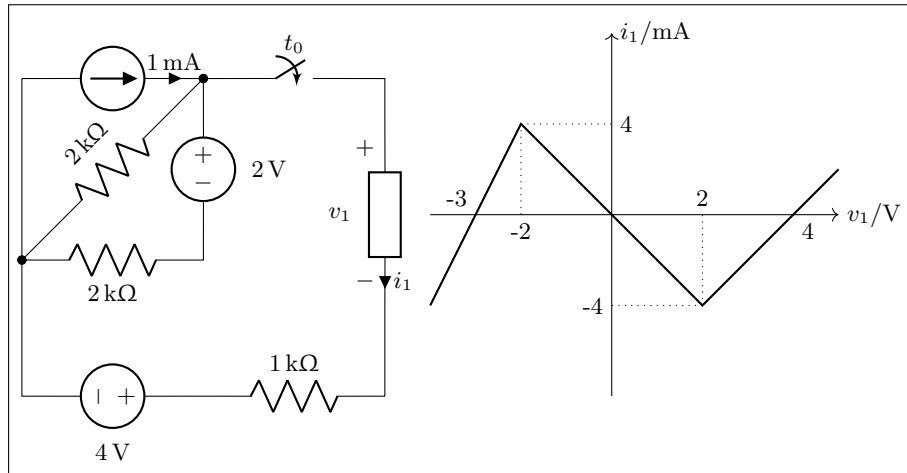


```

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}

```

9.5 Mixing circuits and graphs

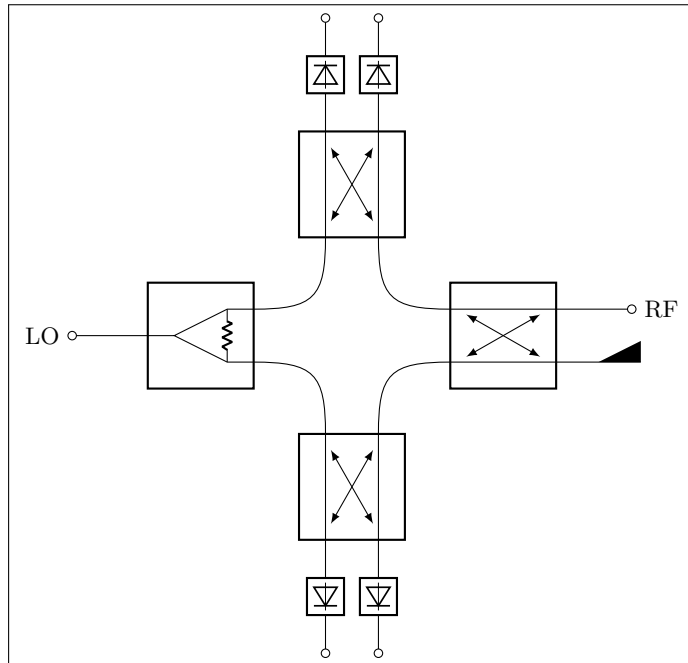


```

1 \begin{circuitikz}[scale=1.2, american]\draw
2   (0,2) to[I=1<\milli\ampere>] (2,2)
3     to[R, l=2<\kilo\ohm>, *-] (0,0)
4     to[R, l=2<\kilo\ohm>] (2,0)
5     to[V, v=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<\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] {$v_1/\text{volt}$};
13   \draw [->] (0,-2) -- (0,2) node[anchor=west] {$i_1/\text{SI}\{\}\text{milli}\text{ampere}\}$} ;
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}

```

9.6 RF circuit

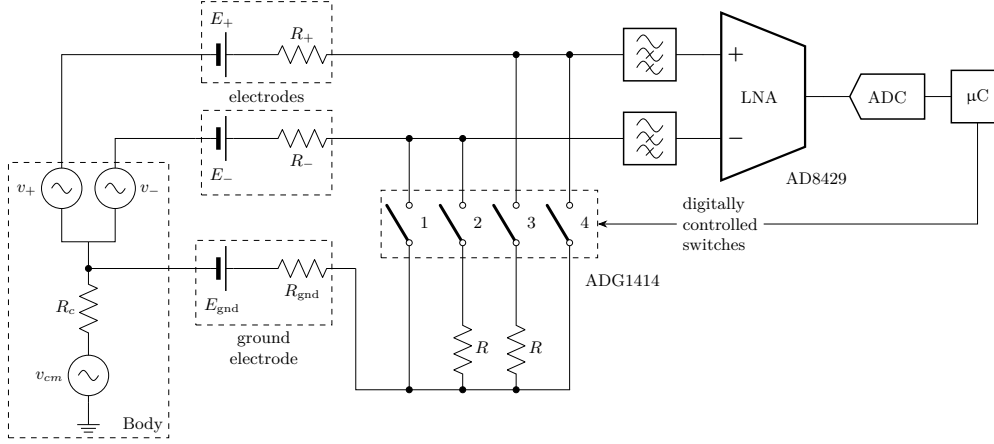


```

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}

```

9.7 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){};

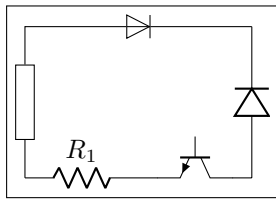
```

```

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

```

9.8 An example with the compatibility option

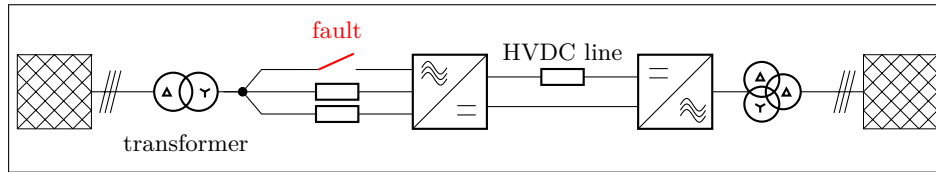


```

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}

```

9.9 3-phases block schematic



```

1 \begin{circuitikz}[smallR/.style={european resistor, resistors/scale=0.5}]
2   \draw (0,0) node[tacdcshape, anchor=ac2] (acdc){} to[smallR] ++(-2,0) --
      node[circ] (point){} ++(-.5,0);
3   \draw (acdc.ac1) to[nos, invert, mirror, name=switch,color=red] ++(-2,0) --
      (point);
4   \draw (acdc.ac3) to[smallR] ++(-2,0)
5     -- (point)
6     to[oosourcetrans,prim=wye,sec=delta,l=transformer] ++(-1.5,0)
7     to[tmultiwire] ++(-.5,0)
8     node[gridnode, anchor=right]{};
9   \node[above=.3cm,color=red] at (switch) {fault};
10  \draw (acdc.dc1) to[smallR,l=HVDC line] ++(2,0) node[tdcacshape, anchor=dc
      1] (dcac){};
11  \draw (acdc.dc2) -- (dcac.dc2);
12  \draw (dcac.right) to[ooosource,prim=delta,sec=delta,tert=wye,invert]
      ++(1.5,0)
13    to[tmultiwire] ++(.5,0) node[gridnode,anchor=left]{};
14 \end{circuitikz}

```

10 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 1.2.3 (unreleased)
 - Fixed size of “not circle” in flip-flops to match european style not circle when used.
 - added a Mach-Zehnder-Modulator block symbol as node mzm by user @dl1chb
 - Block anchors: add border anchors for round elements and deprecate old 1, 2, 3, 4 anchors.
- 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.

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 globel option “strokediode” or stroke instead of full/empty, or D-. Use this option for compliance with DIN standard EN-60617
 - Improved Shape of Diodes:tunnel diode, Zener diode, schottky diode (bit longer lines at cathode)
 - Reworked igtbt: New anchors G,gate and new L-shaped form Lnigtbt, Lpigtbt
 - 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, nifete, nigfetd, pfet, pigfete, pigfetd (contribution of Clemens Helfmeier and Theodor Borsche)
 - njfet, pjfet (contribution of Danilo Piazzalunga)
 - pigbt, nigbt
 - *backward incompatibility* potentiometer is now the standard resistor-with-arrow-in-the-middle; the old potentiometer is now known as variable resistor (or vR), similarly to variable inductor and variable capacitor
 - triac, thyristor, memristor
 - new property “name” for bipoles
 - fixed voltage problem for batteries in american voltage mode
 - european logic gates
 - *backward incompatibility* new american standard inductor. Old american inductor now called “cute inductor”
 - *backward incompatibility* transformer now linked with the chosen type of inductor, and version with core, too. Similarly for variable inductor
 - *backward incompatibility* styles for selecting shape variants now end are in the plural to avoid conflict with paths
 - new placing option for some tripoles (mostly transistors)
 - mirror path style

- Version 0.2.2 - 20090520
 - Added the shape for lamps.
 - Added options `europeanresistor`, `europeaninductor`, `americanresistor` and `americaninductor`, with corresponding styles.
 - FIXED: error in transistor arrow positioning and direction under negative `xscale` and `yscale`.
- Version 0.2.1 - 20090503
 - Op-amps added
 - added options `arrowmos` and `noarrowmos`, to add arrows to pmos and nmos
- Version 0.2 - 20090417 First public release on CTAN
 - *Backward incompatibility*: labels ending with `:angle` are not parsed for positioning anymore.
 - Full use of 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, 66
adder, 65
afuse, 60
ageneric, 39
ALU, 126
american and port, 108
american buffer port, 108
american controlled current source, 50
american controlled voltage source, 50
american current source, 49
american gas filled surge arrester, 60
american inductive sensor, *see* sL
american inductor, *see* L
american nand port, 108
american nor port, 108
american not port, 108
american or port, 108
american potentiometer, *see* pR, *see* pR
american resistive sensor, *see* sR
american resistor, *see* resistor, *see* R
american voltage source, 48
american xnor port, 108
american xor port, 108
ammeter, 28, 54
amp, 67
antenna, 88
asymmetric fuse, *see* afuse

bandpass, 66
bandstop, 66
bare7seg, 131
bareantenna, 87
bareRXantenna, 87
bareTXantenna, 87
barrier, 60
battery, 48
battery1, 48
battery2, 48
biD*, *see* full bidirectionaldiode
biDo, *see* empty bidirectionaldiode
bjtnpn, collectors=1, emitters=2, 73
bjtnpn, collectors=2, emitters=2, bjt pins
width=0, bjt multi height=0.8, 79
bjtpnp, collectors=3, emitters=2, 73
bmultiwire, 61, *see* bmultiwire
bnc, 65
buffer, 96
bulb, 60

C, *see* capacitor
capacitive sensor, 41
capacitor, 41
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, 36
circ, 64
circulator, 66
csource, *see* european controlled current source
csourceAM, *see* american controlled current
source
csourceC, *see* cute european controlled current
source
csourcesin, *see* controlled sinusoidal current
source
closing switch, 102
cogsw, *see* cute opening switch
controlled isourcesin, *see* controlled sinusoidal
current source
controlled sinusoidal current source, 51
controlled sinusoidal voltage source, 50
controlled vsourcesin, *see* controlled sinusoidal
voltage source
cosw, *see* cute open switch
coupler, 68
coupler2, 68
crossing, 62
csI, *see* controlled sinusoidal current source
cspst, *see* closing switch
csV, *see* controlled sinusoidal voltage source
curarrow, 63
curved capacitor, 41
cute choke, 42
cute closed switch, 103
cute closing switch, 103
cute european controlled current source, 50
cute european controlled voltage source, 50
cute european current source, 49
cute european voltage source, 48
cute inductive sensor, *see* sL
cute inductor, *see* L
cute open switch, 103
cute opening switch, 103
cute spdt down, 103
cute spdt down arrow, 30, 103
cute spdt mid, 103
cute spdt mid arrow, 103
cute spdt up, 103
cute spdt up arrow, 103
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, 66
damper, 59, 162
dcisource, 53
dcvsource, 53
demux, 125
detector, 67
diamondpole, 64
diodetube, 83
diodetube,filament, 84
diodetube,filament,nocathode, 84
diodetube,fullcathode, 85
dipchip, 128
Do, *see* empty diode
dsp, 67

eC, *see* ecapacitor
ecapacitor, 41
ecsource, *see* empty controlled source
eground, 36
eground2, 36
elko, *see* ecapacitor
elmech, 89
empty bidirectionaldiode, 45
empty controlled source, 50
empty diode, 44
empty led, 45
empty photodiode, 45
empty Schottky diode, 44
empty thyristor, 47
empty triac, 46
empty tunnel diode, 44
empty varcap, 45
empty Zener diode, 44
empty ZZener diode, 44
en amp, 95
esource, 52
european and port, 110
european buffer port, 110
european controlled current source, 50
european controlled voltage source, 50
european current source, 49
european gas filled surge arrester, 60
european inductive sensor, *see* sL
european inductor, *see* L
european nand port, 110
european nor port, 110
european not port, 110
european or port, 110
european potentiometer, *see* pR
european resistive sensor, *see* sR
european resistor, *see* R
european voltage source, 48
european xnor port, 110
european xor port, 110

fd inst amp, 96
fd op amp, 95

fft, 67
flipflop, 120
flipflop AB, 120
flipflop D, 121
flipflop JK, 121
flipflop JK, add async SR, 122
flipflop JK, add async SR, external pins
width=0, 123
flipflop JK, dot on notQ, 121
flipflop myJK, 122
flipflop SR, 121
flipflop T, 121
flowarrow, 63
fourport, 68
full bidirectionaldiode, 45
full diode, 45
full led, 45
full photodiode, 45
full Schottky diode, 45
full thyristor, 47
full triac, 46
full tunnel diode, 45
full varcap, 45
full Zener diode, 45
full ZZener diode, 45
fuse, 60

generic, 38
gm amp, 95
gridnode, 66
ground, 36
gyrator, 91

hemt, 73
highpass, 66

ieeestd and port, 109
ieeestd buffer port, 109
ieeestd invschmitt port, 109
ieeestd nand port, 109
ieeestd nor port, 109
ieeestd not port, 109
ieeestd or port, 109
ieeestd schmitt port, 109
ieeestd xnor port, 109
ieeestd xor port, 109
iloop, 55
iloop2, 55
inertor, 59
inline buffer, 111
inline invschmitt, 111
inline not, 111
inline schmitt, 111
inputarrow, 63
inst amp, 95
inst amp ra, 96
invschmitt, 108
ioosource, 52
isfet, 74

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, 62

 L, 42
 lamp, 60
 latch, 121
 leD*, *see* full led
 leD-, *see* stroke led
 leDo, *see* empty led
 Lnigbt, 72
 loudspeaker, 61
 lowpass, 66
 Lpigt, 72
 Lpigt, bodydiode, 72

 magnetron, 86
 mass, 59
 match, 88
 memristor, 39
 mic, 61
 mixer, 65
 Mr, *see* memristor
 mslstub, 87
 msport, 88
 msrstub, 88
 mstline, 87
 multiwire, 61, *see* multiwire
 muxdemux, 125
 mzm, 66

 ncpb, *see* normally closed push button
 ncs, *see* normal closed switch
 nfet, 73
 nground, 36
 nI, *see* noise current source
 nigbt, 72
 nigfetd, 74
 nigfete, 73
 nigfete,solderdot, 74
 nigfetebulk, 74
 njfet, 74
 nmos, 73, 75
 nmos, bulk, 78
 noise current source, 51
 noise voltage source, 51
 nopb, *see* push button
 normal closed switch, 102
 normal open switch, 102
 normally closed push button, 102
 normally open push button, *see* push button
 nos, *see* normal open switch
 notcirc, 110
 npn, 30, 71
 npn, bodydiode, 71

npn, schottky base, 71
 npn,photo, 72
 nV, *see* noise voltage source

 ocirc, 64
 odiamondpole, 64
 ohmmeter, 54
 one bit adder, 125
 ooosource, 52
 oosourcetrans, 52
 op amp, 95
 open, 38
 openbarrier, 60
 opening switch, 102
 oscillator, 65
 oscope, 54
 ospst, *see* opening switch
 osquarepole, 64

 pC, *see* polar capacitor
 pD*, *see* full photodiode
 pD-, *see* stroke photodiode
 pDo, *see* empty photodiode
 pentode, 84
 pentode suppressor to cathode, 84
 pfet, 74
 pground, 36
 phaseshifter, 67
 photoresistor, *see* phR
 phR, 40
 piattenuator, 67
 piezoelectric, 41
 pigbt, 72
 pigfetd, 74
 pigfete, 74
 pigfetebulk, 74
 pjfet, 74
 plain amp, 30, 96
 plain crossing, 62
 plain mono amp, 96
 pmos, 73, 75
 pmos, bulk, 78
 pmos,emptycircle, 78
 pmos,nocircle,arrowmos, 78
 pnp, 71
 pnp, schottky base, 71
 pnp,photo, 72
 polar capacitor, 42
 pR, 28, *see* pR, 39
 push button, 102
 pvsource, 52
 PZ, *see* piezoelectric

 qfpchip, 128
 qiprobe, 54
 qpprobe, 54
 qvprobe, 54

 R, *see* resistor, 39

- reed, 102
- resistor, 28
- rground, 36
- rmeter, 54
- rmeterwa, 54
- rotaryswitch, 30, 105
- rxantenna, 88

- sacdc, 67
- sC, *see* capacitive sensor
- schmitt, 108
- schmitt symbol, 110
- sD*, *see* full Schottky diode
- sD-, *see* stroke Schottky diode
- sdcac, 67
- sDo, *see* empty Schottky diode
- sground, 36
- short, 38
- sI, *see* sinusoidal current source
- sinusoidal current source, 49
- sinusoidal voltage source, 49
- sL, 42, 43
- smeter, 54
- spdt, 102
- spring, 59
- spst, *see* switch
- square voltage source, 52
- squarepole, 64
- squid, 60
- sqV, *see* square voltage source
- sR, 39
- stroke diode, 46
- stroke led, 46
- stroke photodiode, 46
- stroke Schottky diode, 46
- stroke thyristor, 47
- stroke tunnel diode, 46
- stroke varcap, 46
- stroke Zener diode, 46
- stroke ZZener diode, 46
- sV, *see* sinusoidal voltage source
- switch, 102

- tacdc, 67
- tattenuator, 67
- tD*, *see* full tunnel diode
- tD-, *see* stroke tunnel diode
- tdcac, 67
- tDo, *see* empty tunnel diode
- tetrode, 83
- tgeneric, 39
- tground, 36
- thermistor, *see* thR
- thermistor ntc, *see* thRn
- thermistor ptc, *see* thRp
- thermocouple, 60
- thR, 40
- thRn, 40
- thRp, 40

- thyristor, 46
- TL, 88
- tlground, 36
- tline, *see* TL
- tlinestub, 88
- tmultiwire, 61, *see* tmultiwire
- toggle switch, 102
- Tr, *see* triac
- Tr*, *see* full triac
- transformer, 90, 91
- transformer core, 91
- transmission line, *see* TL
- trarrow, 63
- triac, 46
- triode, 83
- Tro, *see* empty triac
- tV, *see* vsourcetri
- twoport, 66
- txantenna, 88
- Ty, *see* thyristor
- Ty*, *see* full thyristor
- Ty-, *see* stroke thyristor
- Tyo, *see* empty thyristor

- vamp, 67
- variable american inductor, *see* vL
- variable american resistor, *see* vR
- variable capacitor, 41
- variable cute inductor, *see* vL
- variable european inductor, *see* vL
- variable european resistor, *see* vR
- varistor, 39
- vC, *see* variable capacitor
- VC*, *see* full varcap
- VC-, *see* stroke varcap
- vcc, 37
- VCo, *see* empty varcap
- vco, 66
- vee, 37
- viscoe, 59, 165
- vL, 42, 43
- voltmeter, 54
- voosource, 52
- vphaseshifter, 67
- vpiattenuator, 67
- vR, 39
- 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, 52
- vtattenuator, 67

- waves, 87
- wilkinson, 66

- xgeneric, 38

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