

The TikZ-Extensions Package
Manual for version 0.2
<https://github.com/Qrrbrbirlbel/tikz-extensions>

Qrrbrbirlbel

September 11, 2022

Contents

I	Introduction	4
1	Usage	4
2	Why do we need it?	4
3	Should these libraries be part of TikZ?	4
II	TikZ Libraries	5
4	Calendar	6
4.1	Value-keys and nestable if key	6
4.2	Week numbering (ISO 8601)	6
5	Node Families	7
6	Arc <i>to</i> a point	9

7	More Horizontal and Vertical Lines	11
7.1	Zig-Zag	11
7.2	Zig-Zig	13
7.3	Even more Horizontal and Vertical Lines	14
8	Extending the Path Timers	15
8.1	Rectangle	15
8.2	Parabola	15
8.3	Sine/Cosine	16
9	Using Images as a Pattern	17
10	Positioning Plus	18
10.1	Useful corner anchors	18
10.2	Useful placement keys for vertical and horizontal alignment	19
11	Arcs through Three Points	23
12	Mirror, Mirror on the Wall	24
12.1	Using the reflection matrix	24
12.2	Using built-in transformations	25
III	PGF Libraries	27
13	Transformations: Mirroring	28
13.1	Using the reflection matrix	28
13.2	Using built-in transformations	28
14	Shape: Circle Arrow	30
15	Shape: Circle Cross Split	33
16	Shape: Rectangle with Rounded Corners	36
17	Shape: Superellipse	38
IV	Utilities	41

18	Calendar: Weeknumbers and more conditionals	42
18.1	Extensions	42
18.2	Week numbering (ISO 8601)	42
19	And a little bit more	43
19.1	PGFmath	43
19.1.1	Postfix operator R	43
19.1.2	Functions	43
19.1.3	Functions: using coordinates	44
19.2	PGFkeys	44
19.2.1	Conditionals	44
19.2.2	Handlers	45
19.3	PGFfor	46
V	Changelog & Index	47
	Changelog	47
	Index	48

Part I

Introduction

1 Usage

This package is called `tikz-ext`, however, one can't load it via `\usepackage`. Instead, this package consists of multiple PGF and *TikZ* libraries which are loaded by either `\usepgflibrary` or `\usetikzlibrary`.

2 Why do we need it?

Since I have been answering questions on TeX.sx I've noticed that some questions come up again and again, every time with a slightly different approach on how to

solve them.

I don't like reinventing the wheel which is why I've gathered the code of my answers in this package.

And, yes, I am using them myself, too.

3 Should these libraries be part of *TikZ*?

I guess.

Part II

TikZ Libraries

These libraries only work with TikZ.



4 Calendar

TikZ Library `ext.calendar-plus`

```
\usetikzlibrary{ext.calendar-plus} % LATEX and plain TEX
\usetikzlibrary[ext.calendar-plus] % ConTEXt
```

This library extends the TikZ library calendar.

4.1 Value-keys and nestable if key

The values of following keys are originally stored in some macros that are not accessible by the user. These are now simple value-keys. The @-protected macros are still available, of course.

<code>/tikz/day xshift</code>	(initially 3ex)
<code>/tikz/day yshift</code>	(initially 3.5ex)
<code>/tikz/month xshift</code>	(initially 9ex)
<code>/tikz/month yshift</code>	(initially 9ex)

It is now also possible to nest `/tikz/if` occurrences.

`/tikz/if=(<conditions>)(<code or options>)else(<else code or options>)` (no default)

4.2 Week numbering (ISO 8601)

The actual week number algorithm is implemented by the `pgfcalendar-ext` package/module in section 18.2.

`/tikz/week code=<code>` (no default)

Works like `/tikz/day code` or `/tikz/month code`, only for weeks.

`/tikz/week text=<text>` (no default)

Works like `/tikz/day text` or `/tikz/month text`, only for weeks.

`/tikz/every week` (style, no value)

Works like `/tikz/every day` or `/tikz/every month`, only for weeks.

`/tikz/week label left` (style, no value)

Places the week label to the left of the first day of the month. (For `week list` and `month list` where a week does not start on a Monday, the position is chosen “as if” the week had started on a Monday – which is usually exactly what you want.)

July												
26				1	2	3						
27	4	5	6	7	8	9	10					
28	11	12	13	14	15	16	17					
29	18	19	20	21	22	23	24					
30	25	26	27	28	29	30	31					

```
\usetikzlibrary {ext.calendar-plus}
\tikz
  \calendar [week list, month label above
    centered,
              dates=2022-07-01 to 2022-07-31,
              week label left,
              every week/.append
style={gray!50!black, font=\sfamily}];
```

5 Node Families

TikZ Library `ext.node-families`

```
\usetikzlibrary{ext.node-families} % LATEX and plain TEX  
\usetikzlibrary[ext.node-families] % ConTEXt
```

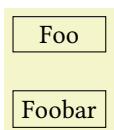
With this library the user can instruct multiple nodes to have the same width, height, text width, text height or text width. This uses the hook `/tikz/execute at end picture` to write the nodes' measurements to the AUX file.

Unfortunately, this does not work with the external library.¹

This library introduces two new shapes called `Circle` and `Rectangle` that are basically copies of the original shapes `circle` and `rectangle`. However, their dimension will be set to the same maximum minimum width and minimum height when one of the following $\langle name \rangle$ s are declared.

`/tikz/node family/width= $\langle name \rangle$` (no default, initially `{}`)

Nodes with the same $\langle name \rangle$ will have the same `/pgf/minimum width`. An empty $\langle name \rangle$ disables the evaluation by the library.



```
\usetikzlibrary {positioning,ext.node-families}  
\tikzexternaldisable % ext.node-families does not work with active externalization  
\begin{tikzpicture}[nodes={Rectangle, draw, node family/width=manual}]  
\node (a) {Foo};  
\node[below=of a] (b) {Foobar};  
\end{tikzpicture}
```

`/tikz/node family/height= $\langle name \rangle$` (no default, initially `{}`)

Nodes with the same $\langle name \rangle$ will have the same `/pgf/minimum height`. An empty $\langle name \rangle$ disables the evaluation by the library.

`/tikz/node family/size= $\langle name \rangle$` (no default)

Sets both height and width.

While `node family/width` and `node family/height` only work for the new shapes `Circle` and `Rectangle`, the following keys – when setup, see below – work with every shape with one single node part. Initially though, only `circle`, `rectangle`, `Circle` and `Rectangle` are set up that way.

`/tikz/node family/text height= $\langle name \rangle$` (no default, initially `{}`)

Nodes with the same $\langle name \rangle$ will have the same text height. An empty $\langle name \rangle$ disables the evaluation by the library.

`/tikz/node family/text depth= $\langle name \rangle$` (no default, initially `{}`)

Nodes with the same $\langle name \rangle$ will have the same text depth. An empty $\langle name \rangle$ disables the evaluation by the library.

¹First of all, I can't figure out how to use the AUX file during externalization since it gets written to the LOG instead. And then there's the question about how external would notice the need to export the picture again until it's stable ...

`/tikz/node family/text width=<name>`

(no default, initially {})

Nodes with the same `<name>` will have the same text width. An empty `<name>` disables the evaluation by the library.

`/tikz/node family/text=<name>`

(no default)

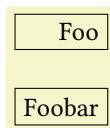
Sets text height, text depth and text width.

Since the width of the node's content's box is setup much earlier, the previous key only extends the width of that box which would make the text seem as if it were aligned to the left. With `text width family align` this can be changed.

`/tikz/node family/text width align=<alignment>`

(no default, initially center)

`<alignment>` is one of left, center or right.



```
\usetikzlibrary {positioning,ext.node-families}
\tikzexternaldisable % ext.node-families does not work with active externalization
\begin{tikzpicture}[nodes={Rectangle, draw, node family={text width=manual, text width align=right}}]
\node (a) {Foo};
\node[below=of a] (b) {Foobar};
\end{tikzpicture}
```

`/tikz/node family/prefix=<prefix>`

(no default, initially \pgfpictureid-)

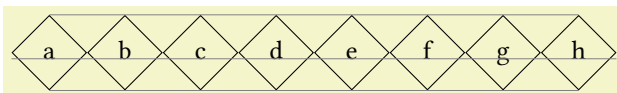
The family names are prefixed with the value of `/tikz/node family/prefix`.

`/tikz/node family/setup shape=<shape>`

(no default)

This adds instructions to the `<shape>`'s definition which adjust the text box's dimensions according to the family.

This should be only used once per shape.



```
\usetikzlibrary {ext.node-families,shapes.geometric}
\tikzexternaldisable % ext.node-families does not work with active externalization
\begin{tikzpicture}[node family/setup shape=diamond]
\foreach \cnt[count=\Cnt] in {a,...,h}
\node[draw, diamond, node family/text=aT0h] (\cnt)
at (right:\Cnt) {\cnt};
\draw[help lines] (a.south) -- (h.south) (a.north) -- (h.north) (a.base-|a.west) -- (h.base-|h.east);
\end{tikzpicture}
```

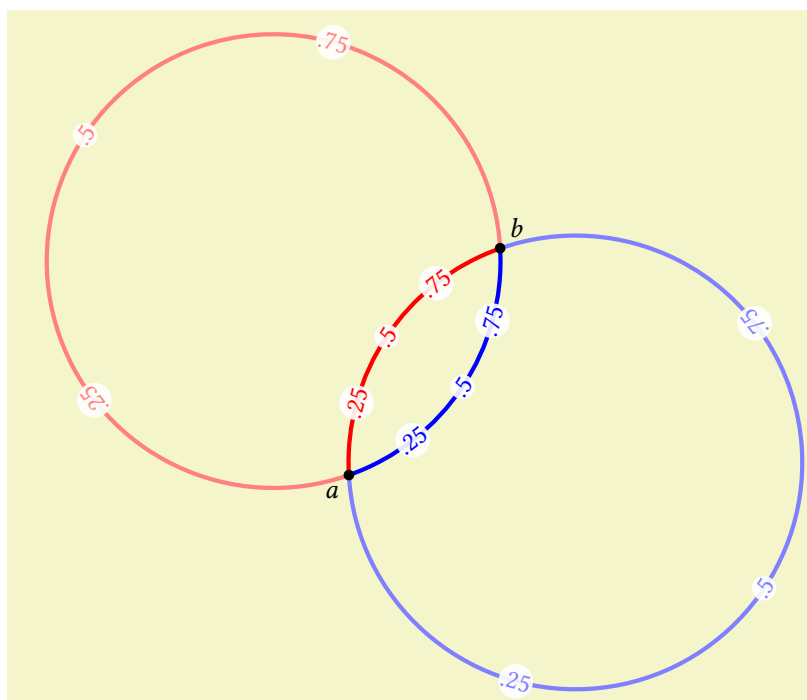

6 Arc to a point

TikZ Library `ext.paths.arcto`

`\usetikzlibrary{ext.paths.arcto}` % \LaTeX and plain \TeX

`\usetikzlibrary[ext.paths.arcto]` % Con \TeX t

This library adds the new path operation `arc to` to that specifies an arc *to* a point – without the user having to specify any angles.



```
\usetikzlibrary {ext.paths.arcto}
\begin{tikzpicture}[ultra thick,dot/.style={label={#1}}]
\coordinate[dot=below left:$a$] (a) at (0,0);
\coordinate[dot=above right:$b$] (b) at (2,3);
\begin{scope}[
  radius=3,
  nodes={
    shape=circle,
    fill=white,
    fill opacity=.9,
    text opacity=1,
    inner sep=+0pt,
    sloped,
    allow upside down
  }]
\draw[blue] (a) arc to[]
  node[near start] {.25} node {.5} node[near end] {.75} (b);
\draw[red] (a) arc to[clockwise]
  node[near start] {.25} node {.5} node[near end] {.75} (b);
\draw[blue!50] (a) arc to[large]
  node[near start] {.25} node {.5} node[near end] {.75} (b);
\draw[red!50] (a) arc to[large, clockwise]
  node[near start] {.25} node {.5} node[near end] {.75} (b);
\end{scope}

\fill[radius=2pt] (a) circle[] (b) circle[];
\end{tikzpicture}
```

`\path ... arc to[options](coordinate or cycle) ...;`

When this operation is used, the path gets extended by an arc that goes through the current point and *(coordinate)*.

For two points there exist two circles or four arcs that go through or connect these two points. Which one of these is constructed is determined by the following options that can be used inside of *(options)*.

`/tikz/arc to/clockwise`

(style, no value)

This constructs an arc that goes clockwise.

`/tikz/arc to/counter clockwise`

(style, no value)

This constructs an arc that goes counter clockwise.

This is the default.

`/tikz/arc to/large`

(style, no value)

This constructs an arc whose angle is larger than 180°.

`/tikz/arc to/small`

(style, no value)

This constructs an arc whose angle is smaller than 180°.

`/tikz/arc to/rotate=<degree>`

(no default)

Rotates the arc by $\langle degree \rangle$. This is only noticeable when x radius and y radius are different.

`/tikz/arc to/x radius=<value>`

(no default)

This forwards the $\langle value \rangle$ to `/tikz/x radius`. Its $\langle value \rangle$ is used for the radius of the arc.

`/tikz/arc to/y radius=<value>`

(no default)

This forwards the $\langle value \rangle$ to `/tikz/y radius`. Its $\langle value \rangle$ is used for the radius of the arc.

`/tikz/arc to/radius=<value>`

(no default)

This forwards the $\langle value \rangle$ to both `/tikz/x radius` and `/tikz/y radius`. Its $\langle value \rangle$ is used for radius of the arc.

`/tikz/every arc to`

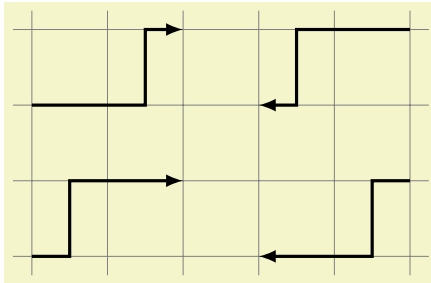
(style, no value)

After `/tikz/every arc to` this will also be applied before any $\langle options \rangle$ are set.

It should be noted that this uses `\pgfpatharcto` for which the TikZ manual warns:

The internal computations necessary for this command are numerically very unstable. In particular, the arc will not always really end at the $\langle target coordinate \rangle$, but may be off by up to several points. A more precise positioning is currently infeasible due to \TeX 's numerical weaknesses. The only case it works quite nicely is when the resulting angle is a multiple of 90°.

The `arc to` path operation will also work only in the canvas coordinate system. The lengths of the vectors (1, 0) and (0, 1) will be used for the calculation of the radii but no further consideration is done.



```
\usetikzlibrary {ext.paths.ortho}
\begin{tikzpicture}[very thick,-latex]
\draw[help lines,-] (-.25, -.25) grid (5.25, 3.25);
\draw (0, 0) -|-[distance=.5cm] ++(2, 1);
\draw (0, 2) -|-[distance=-.5cm] ++(2, 1);

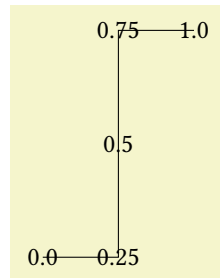
\tikzset{xshift=3cm}
\draw (2, 1) -|-[distance=.5cm] ++(-2, -1);
\draw (2, 3) -|-[distance=-.5cm] ++(-2, -1);
\end{tikzpicture}
```

`/tikz/hvvh/from center=<true or false>`

(default true)

When nodes get connected the placement of the middle part of the Zig-Zag and the Zig-Zig (see below) connections will be calculated from the border of these nodes. The middle part of the connections can be calculated from the nodes' center if this key is set to true.

New timers are setup for both the Zig-Zag and the Zig-Zig connections, these can be configured through the following keys.



```
\usetikzlibrary {paths.ortho}
\tikz \draw (0,0) -|-(2,3)
foreach \p in {0.0, 0.25, 0.5, 0.75, 1.0}{
node [pos=\p] {\p}};
```

`/tikz/hvvh/spacing=<number>`

(no default, initially 4)

Unless *<number>* = 0 is set

- pos = 0 will be at the start,
- pos = 1 will be at the end,
- pos = $\frac{1}{\langle number \rangle}$ will be at the first kink,
- pos = $\frac{\langle number \rangle - 1}{\langle number \rangle}$ will be at the second kink and
- pos = .5 will be in the middle of the middle part of the connection.

If $\langle number \rangle = 0$ then

- $pos = -1$ will be at the start,
- $pos = 2$ will be at the end,
- $pos = 0$ will be at the first kink,
- $pos = 1$ will be at the second kink and
- $pos = .5$ will still be in the middle of the middle part of the connection.

`/tikz/hvvh/middle 0 to 1`

(no value)

This is an alias for $spacing = 0$.

7.2 Zig-Zig

`\path ... r-ud[$\langle options \rangle$] $\langle coordinate or cycle \rangle$...;`

This operation means “first up, then horizontal and then down”.

`/tikz/udlr/ud distance= $\langle length \rangle$`

(no default, initially .5cm)

This sets the distance between the start and the horizontal line to $\langle length \rangle$.

`\path ... r-du[$\langle options \rangle$] $\langle coordinate or cycle \rangle$...;`

This operation means “first down, then horizontal and then up”.

`/tikz/udlr/du distance= $\langle length \rangle$`

(no default, initially .5cm)

This sets the distance between the start and the horizontal line to $\langle length \rangle$.

`\path ... r-lr[$\langle options \rangle$] $\langle coordinate or cycle \rangle$...;`

This operation means “left down, then vertical and then right”.

`/tikz/udlr/lr distance= $\langle length \rangle$`

(no default, initially .5cm)

This sets the distance between the start and the vertical line to $\langle length \rangle$.

`\path ... r-rl[$\langle options \rangle$] $\langle coordinate or cycle \rangle$...;`

This operation means “first right, then vertical and then down”.

`/tikz/udlr/rl distance= $\langle length \rangle$`

(no default, initially .5cm)

This sets the distance between the start and the vertical line to $\langle length \rangle$.

All distances can be set with on key.

`/tikz/udlr/distance=<length>` (no default)

Sets all distances in the `/tikz/udlr` namespace.

`/tikz/udlr/from center=<true or false>` (no default, initially false, default true)

This is an alias for `/tikz/hvvh/from center`.

7.3 Even more Horizontal and Vertical Lines

The following keys can be used to access vertical and horizontal line path operations.

`/tikz/horizontal vertical` (style, no value)

This installs to `path = -| (\tikztotarget) \tikztonodes` that can be used with the path operations `to` or `edge`.

`/tikz/vertical horizontal` (style, no value)

This installs to `path = |- (\tikztotarget) \tikztonodes` that can be used with the path operations `to` or `edge`.

`/tikz/horizontal vertical horizontal` (style, no value)

This installs to `path = -|- (\tikztotarget) \tikztonodes` that can be used with the path operations `to` or `edge`.

`/tikz/vertical horizontal vertical` (style, no value)

This installs to `path = |-| (\tikztotarget) \tikztonodes` that can be used with the path operations `to` or `edge`.

When connecting rectangular nodes, these keys could be useful as well. They all need to be given to a `to` or `edge` path operation.

`/tikz/only vertical second=<length>` (style, default 0pt)

This draws a vertical line from the start point to the target point so that it connects to the target point in the center (or at its border in case it is a node).

The optional `<length>` can be used to shift the line orthogonally to its direction.

`/tikz/only horizontal second=<length>` (style, default 0pt)

This draws a horizontal line from the start point to the target point so that it connects to the target point in the center (or at its border in case it is a node).

The optional `<length>` can be used to shift the line orthogonally to its direction.

`/tikz/only vertical first=<length>` (style, default 0pt)

This draws a vertical line from the start point to the target point so that it connects to the start point in the center (or at its border in case it is a node).

The optional `<length>` can be used to shift the line orthogonally to its direction.

`/tikz/only horizontal first=<length>` (style, default 0pt)

This draws a horizontal line from the start point to the target point so that it connects to the start point in the center (or at its border in case it is a node).

The optional `<length>` can be used to shift the line orthogonally to its direction.

8 Extending the Path Timers

TikZ Library `ext.paths.timer`

```
\usetikzlibrary{ext.paths.timer} % LATEX and plain TEX
\usetikzlibrary[ext.paths.timer] % ConTEXt
```

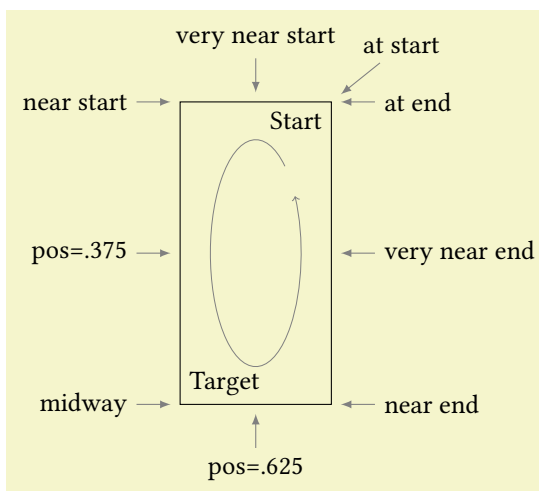
This library adds timers to the path specifications `rectangle`, `parabola`, `sin` and `cos`.

In TikZ, the path specification `rectangle`, `parabola`, `sin` and `cos` do not provide their own timer, i.e. a node placing algorithm that is dependent on the actual path. For `rectangle` the timer of the straight line between the rectangle's corners is used, for the other paths, nodes, coordinates, pics, etc. are placed on the last coordinate.

This library allows this.

8.1 Rectangle

For the `rectangle` path operator, the timer starts with `pos = 0` (= at start) from the starting coordinate in a counter-clockwise direction along the rectangle. The corners will be at positions 0.0, 0.25, 0.5, 0.75 and 1.0.



```
\usetikzlibrary {ext.paths.timer}
\begin{tikzpicture}[scale=2, every pin edge/.style={latex-, gray}]
\coordinate [label=above right:Target] (A) at (0,0);
\coordinate [label=below left:Start] (B) at (1,2);
\draw[->, help lines] ([shift=(50:.3 and .75)] .5,1)
  arc[start angle=50, delta angle=340, x radius=.3, y radius=.75];
\draw (B) rectangle (A)
  foreach \pos/\ang in {at start/60, very near start/90, near start/180, pos=.375/180,
    midway/180, pos=.625/270, near end/0, very near end/0, at end/0}{
    node[pin=\ang:\pos, style/.expanded=\pos]}{};
\end{tikzpicture}
```

8.2 Parabola

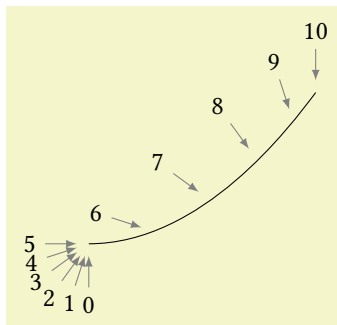
For the `parabola` path operator the timer is similar to the `.. controls ..` operator.

The position 0.5 will lie at the bend.



```
\usetikzlibrary {ext.paths.timer}
\begin{tikzpicture}
\draw[help lines] (-2.25, -1.25) grid (2.25, 3.25);
\draw (2,-1) parabola bend (0,0) (-1,3);
\draw[ultra thick] (-2,-1) parabola bend (0,0) (1,3)
  foreach \pos in {1,...,4,6,7,...,9}{
    node[
      pos=. \pos, sloped, fill=white, font=\small, inner sep=+0pt
    ] {\pos}
  };
\end{tikzpicture}
```

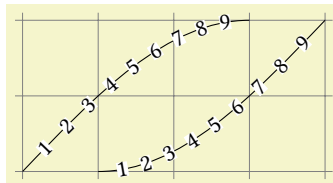
If no bend is specified half the positions will collapse into one end of the curve.



```
\usetikzlibrary {ext.paths.timer}
\begin{tikzpicture}[every pin edge/.style={latex-, shorten <=1pt, gray}]
\draw (-2,-2) parabola (1,0)
  foreach \pos in {0, 1, ..., 10} {
    node [pos=\pos/10, pin={[anchor=-18*\pos+90]-18*\pos+270:\pos]} {}
  };
\end{tikzpicture}
```

8.3 Sine/Cosine

The sin and cos path operators also allow placing of nodes along their paths.



```
\usetikzlibrary {ext.paths.timer}
\begin{tikzpicture}[mark nodes on line/.style={insert path={
  foreach \pos in {1, ..., 9} {node[
    sloped, fill=white, font=\small, inner sep=+0pt, pos=\pos/10] {\pos}}}}]
\draw[help lines] (-2.1,-2.1) grid (2.1,0.1);
\draw (-2,-2) sin (1,0) [mark nodes on line];
\draw[shift=(0:1)](-2,-2) cos (1,0) [mark nodes on line];
\end{tikzpicture}
```


9 Using Images as a Pattern

TikZ Library `ext.patterns.images`

```
\usetikzlibrary{ext.patterns.images} % LATEX and plain TEX
\usetikzlibrary[ext.patterns.images] % ConTEXt
```

This library allows to use an image to be used as a repeating pattern for a path.

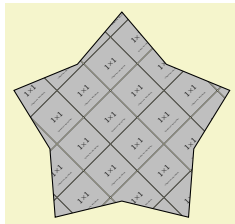
With this library arbitrary images (or indeed PDF documents) can be used as a repeating pattern for the background of a path. This is a two-step process:

1. Declaring an image as an “image-pattern”.
2. Using the “image-pattern”.

`\pgfsetupimageaspattern[⟨options⟩]{⟨name⟩}{⟨image⟩}`

`/tikz/image as pattern=⟨options⟩`

(default {})



```
\usetikzlibrary {ext.patterns.images,shapes.geometric}
\pgfsetupimageaspattern[width=.5cm]{grid}{example-image-1x1}
\tikz \node[star, minimum size=3cm, draw,
  image as pattern={name=grid,options={left, bottom, y=-.5cm, rotate=45}}] {};
```

`/tikz/image as pattern/name=⟨name⟩`

(no default)

Specifies the name of the “image-pattern” to be used.

`/tikz/image as pattern/option`

(style, no value)

Options that will be used by the internal `\pgftext`, only keys from `/pgf/text` should be used.

`/tikz/image as pattern/options=⟨style⟩`

(style, no default)

Appends style `/tikz/image as pattern/option`.

10 Positioning Plus

TikZ Library `ext.positioning-plus`

```
\usetikzlibrary{ext.positioning-plus} % LATEX and plain TEX  
\usetikzlibrary[ext.positioning-plus] % ConTEXt
```

With the help of the positioning and the fit library this extends the placement of nodes.

10.1 Useful corner anchors

The anchors `corner north east`, `corner north west`, `corner south west` and `corner south east` are defined as “generic anchors”, i. e. they are defined for all shapes. This is mostly useful for the placement of circular shapes.

`/tikz/corner above left=<specification>` (style, default 0pt)

Similar as `/tikz/above left` of the TikZ library positioning but uses the `corner north west` anchor.

`/tikz/corner below left=<specification>` (style, default 0pt)

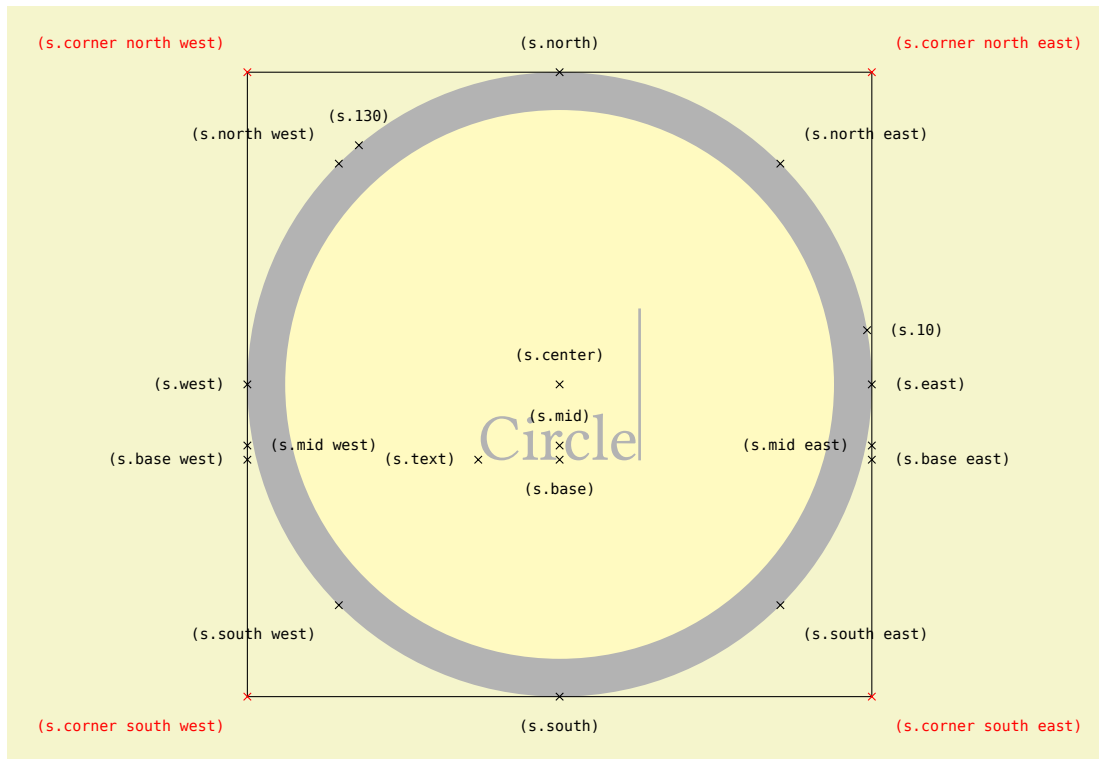
Similar as `/tikz/below left` of the TikZ library positioning but uses the `corner south west` anchor.

`/tikz/corner above right=<specification>` (style, default 0pt)

Similar as `/tikz/above right` of the TikZ library positioning but uses the `corner north east` anchor.

`/tikz/corner below right=<specification>` (style, default 0pt)

Similar as `/tikz/below right` of the TikZ library positioning but uses the `corner south east` anchor.



```
\usetikzlibrary {ext.positioning-plus}
\Huge
\begin{tikzpicture}
\node[name=s,shape=circle,shape example]
{Circle\vrule width 1pt height 2cm};
\foreach \anchor/\placement in {
north west/above left, north/above, north east/above right,
west/left, center/above, east/right,
mid west/right, mid/above, mid east/left,
base west/left, base/below, base east/right,
south west/below left, south/below, south east/below right,
text/left, 10/right, 130/above}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\draw (s.corner north west) rectangle (s.corner south east);
\foreach \anchor/\placement in {
corner north west/above left, corner north east/above right,
corner south west/below left, corner south east/below right}
\draw[red,shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

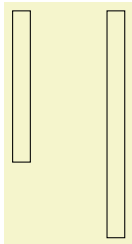
10.2 Useful placement keys for vertical and horizontal alignment

`/tikz/north left=<specification>`

(style, default 0pt)

Like `/tikz/left` but aligns the nodes at their north border.

This is basically the same as `left=of reference.north west, anchor=north east`.



```
\usetikzlibrary {ext.positioning-plus}
\begin{tikzpicture}[nodes=draw]
\node[minimum height=2cm] (a) {};
\node[minimum height=3cm, north right=of a] {};
\end{tikzpicture}
```

/tikz/north right=*<specification>*

(style, default 0pt)

Like /tikz/right but aligns the nodes at their north border.

This is basically the same as left=of reference.north east, anchor=north west.

/tikz/south left=*<specification>*

(style, default 0pt)

Like /tikz/left but aligns the nodes at their south border.

This is basically the same as left=of reference.south west, anchor=south east.

/tikz/south right=*<specification>*

(style, default 0pt)

Like /tikz/right but aligns the nodes at their south border.

This is basically the same as left=of reference.south east, anchor=south west.

/tikz/west above=*<specification>*

(style, default 0pt)

Like /tikz/above but aligns the nodes at their west border.

This is basically the same as left=of reference.north west, anchor=south west.

/tikz/west below=*<specification>*

(style, default 0pt)

Like /tikz/below but aligns the nodes at their west border.

This is basically the same as left=of reference.south west, anchor=north west.

/tikz/east above=*<specification>*

(style, default 0pt)

Like /tikz/above but aligns the nodes at their east border.

This is basically the same as left=of reference.north east, anchor=south east.

/tikz/east below=*<specification>*

(style, default 0pt)

Like /tikz/below but aligns the nodes at their east border.

This is basically the same as left=of reference.south east, anchor=north east.

The same exist for the recently introduces corner anchors, too.

`/tikz/corner north left=<specification>` (style, default 0pt)

The same as `/tikz/north left` but uses the new corner anchors.

`/tikz/corner north right=<specification>` (style, default 0pt)

The same as `/tikz/north right` but uses the new corner anchors.

`/tikz/corner south left=<specification>` (style, default 0pt)

The same as `/tikz/south left` but uses the new corner anchors.

`/tikz/corner south right=<specification>` (style, default 0pt)

The same as `/tikz/south right` but uses the new corner anchors.

`/tikz/corner west above=<specification>` (style, default 0pt)

The same as `/tikz/west above` but uses the new corner anchors.

`/tikz/corner west below=<specification>` (style, default 0pt)

The same as `/tikz/west below` but uses the new corner anchors.

`/tikz/corner east above=<specification>` (style, default 0pt)

The same as `/tikz/east above` but uses the new corner anchors.

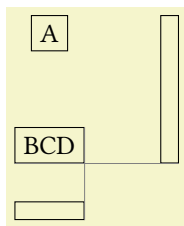
`/tikz/corner east below=<specification>` (style, default 0pt)

The same as `/tikz/east below` but uses the new corner anchors.

While the `<specification>` of all these keys still accept the same form as with TikZ, the `ext.positioning-plus` library extends this even more.

The specification after `of` can contain a list of coordinates (like the `fit` key of the `fit` library). This means that the new node will be placed in relation to a rectangular bounding box that fits around all this nodes in the list.

If this list is prefixed with `|`, `-` or `+`, the new node will also have the same height (`|`), the same width (`-`) or both as this bounding box.



```
\usetikzlibrary {ext.positioning-plus}
\begin{tikzpicture}[nodes=draw]
\node (A) {A};
\node[below=of A] (BCD) {BCD};
\node[right=of |(A)(BCD)] (c) {};
\node[below=.5:of -(A)(BCD)] (d) {};
\draw[help lines] (BCD.south west) -- (c.south east)
(BCD.north east) -- (d.south east);
\end{tikzpicture}
```

This functionality is also available without the placement:

`/tikz/fit bounding box=<list of coordinates>` (style, no default)

Creates a rectangular node with the name `fit bounding box` that encompasses the *<list of coordinates>*.

`/tikz/span vertical=<list of coordinates>` (style, no default)

Creates a rectangular node with the name `fit bounding box` that encompasses the *<list of coordinates>* and sets the `/pgfminimum height` to the height of this bounding box.

`/tikz/span horizontal=<list of coordinates>` (style, no default)

Creates a rectangular node with the name `fit bounding box` that encompasses the *<list of coordinates>* and sets the `/pgfminimum width` to the width of this bounding box.

`/tikz/span=<list of coordinates>` (style, no default)

Is a combination of `/tikz/span vertical` and `/tikz/span horizontal`.

As you maybe noticed in the example above, the *<specification>* also allows a prefix delimited by `:` which the node distance will be multiplied to with for the placement.²

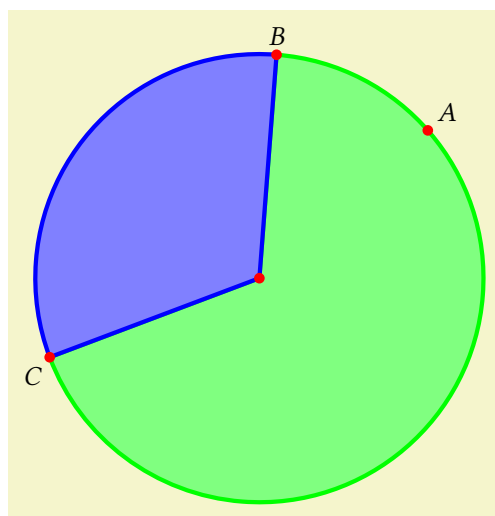
²This is probably more useful when `/tikz/on grid` is used.

11 Arcs through Three Points

TikZ Library `ext.topaths.arcthrough`

```
\usetikzlibrary{ext.topaths.arcthrough} % LATEX and plain TEX
\usetikzlibrary[ext.topaths.arcthrough] % ConTEXt
```

This library allows to use an arc defined by three points.



```
\usetikzlibrary {ext.topaths.arcthrough}
\begin{tikzpicture}
\coordinate[label=above right:$A$] (A) at ( 3, 1);
\coordinate[label=above:$B$] (B) at ( 1, 2);
\coordinate[label=below left:$C$] (C) at (-2,-2);

\draw[ultra thick, draw=green, fill=green!50]
(B) to[arc through={clockwise,(A)}] (C)
-- (arc through center) -- cycle;
\draw[ultra thick, draw=blue, fill=blue!50]
(B) to[arc through=(A)] (C)
-- (arc through center) -- cycle;

\foreach \p in {A,B,C, arc through center} \fill[red] (\p) circle[radius=2pt];
\end{tikzpicture}
```

This can only be used for circles in the canvas coordinate system.

`/tikz/arc through/through=<coordinate>` (no default, initially (0,0))

The coordinate on the circle that defines – together with the starting and target point – a circle.

`/tikz/arc through/center suffix=<suffix>` (no default, initially)

The arc through will define a coordinate named arc through center<suffix> so that it can be referenced later.

`/tikz/arc through/clockwise` (no value)

The resulting arc will go clockwise from the starting point to the target point.

This will not necessarily go through the through point.

`/tikz/arc through/counter clockwise` (no value)

The resulting arc will go counter clockwise from the starting point to the target point. This will not necessarily go through the through point.

`/tikz/arc through=<key-value>` (no default)

This key should be used with to or edge. A parameter other than center suffix, clockwise or counter clockwise will be assumed to be the through coordinate.

12 Mirror, Mirror on the Wall

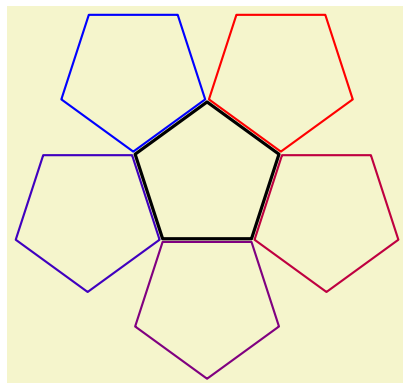
TikZ Library `ext.transformations.mirror`

```
\usetikzlibrary{ext.transformations.mirror} % LATEX and plain TEX
\usetikzlibrary[ext.transformations.mirror] % ConTEXt
```

This library adds more transformations to TikZ.

As explained in section 13, there are two approaches to setting a mirror transformation. As with the commands in PGF, we'll be using a lowercase `m` for the reflection matrix and an uppercase `M` for the built-in approach.

12.1 Using the reflection matrix

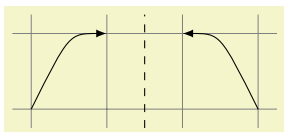


```
\usetikzlibrary {shapes.geometric,ext.transformations.mirror}
\begin{tikzpicture}[line join=round, thick, reg poly/.style={
  shape=regular polygon, regular polygon sides={#1}}]
\node[reg poly=5, minimum size=+2cm, draw, very thick] (a) {};
\foreach \i[evaluate={\col=(\i-1)/.04}] in {1,...,5}
  \node [mirror=(a.corner \i)--(a.side \i), transform shape,
    reg poly=5, minimum size=+2cm, draw=red!\col!blue] {};
\end{tikzpicture}
```

`/tikz/xmirror=<value or coordinate>`

(no default)

Sets up a transformation that mirrors along a horizontal line that goes through point $(\langle value \rangle, 0)$ or $\langle coordinate \rangle$.



```
\usetikzlibrary {ext.transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-0.25, -.25) grid (3.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);
\draw[xmirror=(m),-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```


`/tikz/ymirror=<value or coordinate>`

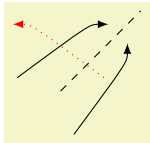
(no default)

Sets up a transformation that mirrors along a vertical line that goes through point $(0, \langle value \rangle)$ or $\langle coordinate \rangle$.

`/tikz/mirror x=<coordinate>`

(no default)

Similar to `/tikz/xmirror`, this however uses the xyz coordinate system instead of the canvas system.



```
\usetikzlibrary {ext.transformations.mirror}
\begin{tikzpicture}[x=.5cm, y=(45:1cm)]

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);

\draw[xmirror=(m), -latex, red, dotted] (0,0) .. controls (.5,1) .. (1,1);
\draw[mirror x=(m), -latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`/tikz/mirror y=<coordinate>`

(no default)

Similar to `/tikz/ymirror`, this however uses the xyz coordinate system instead of the canvas system.

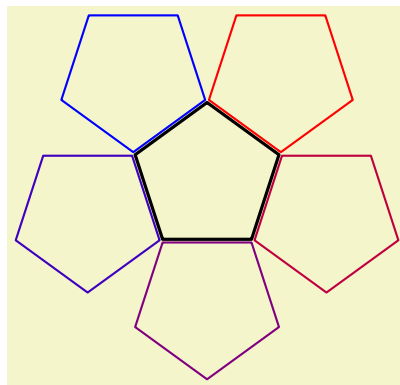
`/tikz/mirror=<point A>--<point B>`

(no default)

Sets up a transformation that mirrors along a line that goes through $\langle point A \rangle$ and $\langle point B \rangle$.

When only $\langle point A \rangle$ is given that line goes through $\langle point A \rangle$ and the origin.

12.2 Using built-in transformations

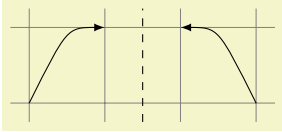


```
\usetikzlibrary {shapes.geometric,ext.transformations.mirror}
\begin{tikzpicture}[line join=round, thick, reg poly/.style={
  shape=regular polygon, regular polygon sides={#1}}]
\node[reg poly=5, minimum size=+2cm, draw, very thick] (a) {};
\foreach \i[evaluate={\col=(\i-1)/.04}] in {1,...,5}
  \node [Mirror=(a.corner \i)--(a.side \i), transform shape,
    reg poly=5, minimum size=+2cm, draw=red!\col!blue] {};
\end{tikzpicture}
```

/tikz/xMirror= $\langle value \text{ or coordinate} \rangle$

(no default)

Sets up a transformation that mirrors along a horizontal line that goes through point $(\langle value \rangle, 0)$ or $\langle coordinate \rangle$.



```
\usetikzlibrary {ext.transformations.mirror}  
\begin{tikzpicture}  
\draw[help lines] (-0.25, -.25) grid (3.25, 1.25);  
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);  
  
\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);  
\draw[xMirror=(m),-latex] (0,0) .. controls (.5,1) .. (1,1);  
\end{tikzpicture}
```

/tikz/yMirror= $\langle value \text{ or coordinate} \rangle$

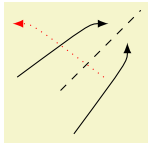
(no default)

Sets up a transformation that mirrors along a vertical line that goes through point $(0, \langle value \rangle)$ or $\langle coordinate \rangle$.

/tikz/Mirror x= $\langle coordinate \rangle$

(no default)

Similar to **/tikz/xMirror**, this however uses the xyz coordinate system instead of the canvas system.



```
\usetikzlibrary {ext.transformations.mirror}  
\begin{tikzpicture}[x=.5cm, y=(45:1cm)]  
  
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);  
  
\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);  
  
\draw[xMirror=(m), -latex, red, dotted] (0,0) .. controls (.5,1) .. (1,1);  
\draw[Mirror x=(m), -latex] (0,0) .. controls (.5,1) .. (1,1);  
\end{tikzpicture}
```

/tikz/Mirror y= $\langle coordinate \rangle$

(no default)

Similar to **/tikz/yMirror**, this however uses the xyz coordinate system instead of the canvas system.

/tikz/Mirror= $\langle point A \rangle$ -- $\langle point B \rangle$

(no default)

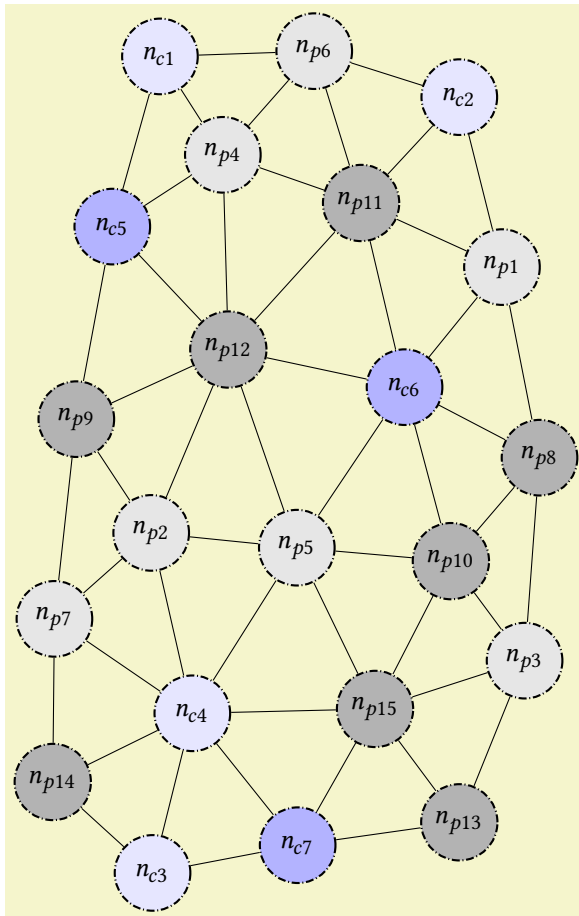
Sets up a transformation that mirrors along a line that goes through $\langle point A \rangle$ and $\langle point B \rangle$.

When only $\langle point A \rangle$ is given that line goes through $\langle point A \rangle$ and the origin.

Part III

PGF Libraries

These libraries (should) work with both PGF and TikZ.



```
\usetikzlibrary {graphs,graphdrawing,ext.misc} \usegdlibrary {force}
\tikzset{
  mynode/.style={
    circle, minimum size=10mm, draw, densely dashdotted, thick,
    decide color/.expand once=#1,
    decide color/.style 2 args={
      /utils/TeX/if=c#1
        {/utils/TeX/ifnum={#2<5}{blue!light}{blue!dark}}
        {/utils/TeX/ifnum={#2<8}{light}{dark}}},
    light/.style={fill=gray!20}, blue!light/.style={fill=blue!10},
    dark/.style={fill=gray!60}, blue!dark/.style={fill=blue!30}}
\tikz\graph[
  spring electrical layout, vertical=c2 to p13,
  node distance=1.5cm, typeset=$n_{\tikzgraphnodetext}$,
  nodes={mynode=\tikzgraphnodetext}] {
  % outer ring
  c2 -- {p1, p11, p6};
  p1 -- {p8, c6, p11};
  p8 -- {p3, p10, c6};
  p3 -- {p13, p15, p10};
  p13 -- {p15, c7};
  c7 -- {c3, c4, p15};
  c3 -- {p14, c4};
  p14 -- {p7, c4};
  p7 -- {p9, p2, c4};
  p9 -- {c5, p12, p2};
  c5 -- {c1, p4, p12};
  c1 -- {p6, p4};
  p6 -- {p11, p4};
  % inner ring
  p11 -- {c6, p12, p4};
  p5 -- {c6 -- {p10, p12}, p10 -- p15, p15 -- c4, c4 -- p2, p2 -- p12, p12 -- p4};
};
```

13 Transformations: Mirroring

PGF Library `ext.transformations.mirror`

```
\usepgflibrary{ext.transformations.mirror} % LATEX and plain TEX
\usepgflibrary[ext.transformations.mirror] % ConTEXt
```

This library adds mirror transformations to PGF.

Two approaches to mirror transformation exist:

1. Using the reflection matrix (see left column).

This depends on `\pgfpointnormalised` which involves the sine and the cosine functions of PGFmath.

2. Using built-in transformations (see right column).

This depends on `\pgfmathanglebetween` which involves the arctangent (`atan2`) function of PGFmath.

Which one is better? I don't know. Choose one you're comfortable with.

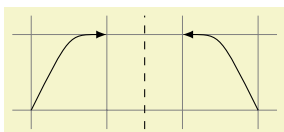
13.1 Using the reflection matrix

The following commands use the reflection matrix that sets the transformation matrix following

$$A = \frac{1}{\|\vec{z}\|^2} \begin{bmatrix} l_x^2 - l_y^2 & 2l_x l_y \\ 2l_x l_y & l_y^2 - l_x^2 \end{bmatrix}.$$

`\pgftransformxmirror{⟨value⟩}`

Sets up a transformation that mirrors along a vertical line that goes through point $(⟨value⟩, 0)$.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-0.25, -
.25) grid (3.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) -- (1.5, 1.25);
\pgftransformxmirror{1.5}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

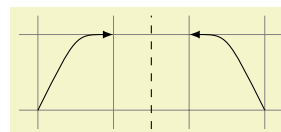
13.2 Using built-in transformations

The following commands use a combination of shifting, rotating, -1 scaling, rotating back and shifting back to reach the mirror transformation.

The commands are named the same as on the left side, only the `m` in `mirror` is capitalized.

`\pgftransformxMirror{⟨value⟩}`

Sets up a transformation that mirrors along a vertical line that goes through point $(⟨value⟩, 0)$.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-0.25, -
.25) grid (3.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) -- (1.5, 1.25);
\pgftransformxMirror{1.5}

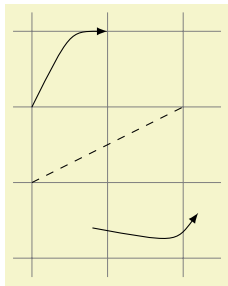
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`\pgftransformymirror{⟨value⟩}`

Sets up a transformation that mirrors along a horizontal line that goes through point $(0, \langle value \rangle)$.

`\pgftransformmmirror{⟨point A⟩}{⟨point B⟩}`

Sets up a transformation that mirrors along the line that goes through $\langle point A \rangle$ and $\langle point B \rangle$.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -2.25) grid (2.5, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, -1) -- (2, 0);
\pgftransformmmirror{\pgfpointxy{0}{-1}}
{\pgfpointxy{2}{0}}

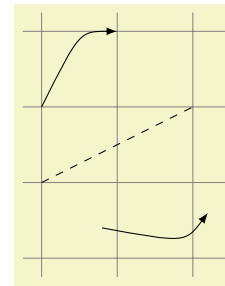
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`\pgftransformyMirror{⟨value⟩}`

Sets up a transformation that mirrors along a horizontal line that goes through point $(0, \langle value \rangle)$.

`\pgftransformMirror{⟨point A⟩}{⟨point B⟩}`

Sets up a transformation that mirrors along the line that goes through $\langle point A \rangle$ and $\langle point B \rangle$.



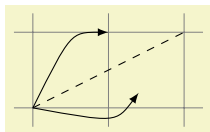
```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -2.25) grid (2.5, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, -1) -- (2, 0);
\pgftransformMirror{\pgfpointxy{0}{-1}}
{\pgfpointxy{2}{0}}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`\pgfqtransformmmirror{⟨point A⟩}`

Sets up a transformation that mirrors along the line that goes through the origin and $\langle point A \rangle$.



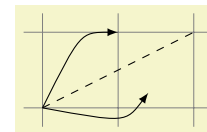
```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -.25) grid (2.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, 0) -- (2, 1);
\pgfqtransformmmirror{\pgfpointxy{2}{1}}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`\pgfqtransformMirror{⟨point A⟩}`

Sets up a transformation that mirrors along the line that goes through the origin and $\langle point A \rangle$.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -.25) grid (2.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, 0) -- (2, 1);
\pgfqtransformMirror{\pgfpointxy{2}{1}}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

14 Shape: Circle Arrow

TikZ Library `ext.shapes.circulararrow`

```
\usepgflibrary{ext.shapes.circulararrow} % LATEX and plain TEX and pure pgf
\usepgflibrary[ext.shapes.circulararrow] % ConTEXt and pure pgf
\usetikzlibrary{ext.shapes.circulararrow} % LATEX and plain TEX when using TikZ
\usetikzlibrary[ext.shapes.circulararrow] % ConTEXt when using TikZ
```

A circular shape that has an arc as its background path that can have an arrow tip.

`/pgf/circle arrow start angle=(start angle)` (no default, initially {})

Sets the start angle.

`/pgf/circle arrow end angle=(end angle)` (no default, initially {})

Sets the end angle.

`/pgf/circle arrow delta angle=(delta angle)` (no default, initially {})

Sets the delta angle.

`/pgf/circle arrow arrows=(start arrow tip specification)-(end arrow tip specification)` (no default, initially -)

The specification will be forwarded to `\pgfsetarrows`.

`/pgf/circle arrow turn left north` (no value)

Sets circle arrow start angle = 100, circle arrow delta angle = 340 and circle arrow arrows = ->.

`/pgf/circle arrow turn left east` (no value)

As above but circle arrow start angle = 10.

`/pgf/circle arrow turn left west` (no value)

As above but circle arrow start angle = 280.

`/pgf/circle arrow turn left south` (no value)

As above but circle arrow start angle = 190.

`/pgf/circle arrow turn right north` (no value)

Sets circle arrow start angle = 100, circle arrow delta angle = 340 and circle arrow arrows = <-.

`/pgf/circle arrow turn right east` (no value)

As above but circle arrow start angle = 10.

`/pgf/circle arrow turn right west`

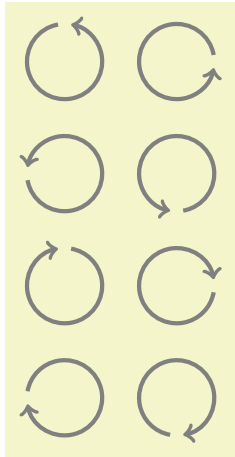
(no value)

As above but circle arrow start angle = 280.

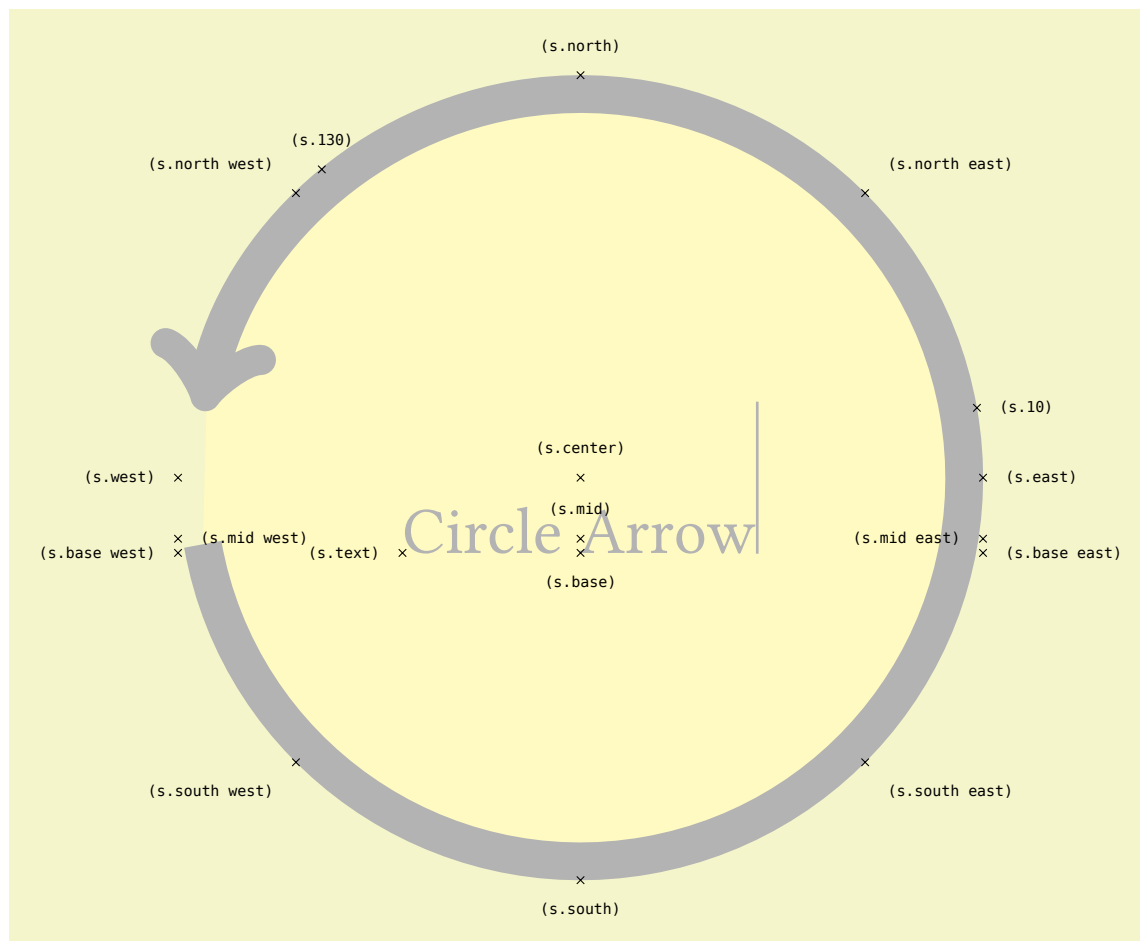
`/pgf/circle arrow turn right south`

(no value)

As above but circle arrow start angle = 190.



```
\usetikzlibrary {ext.shapes.circlearrow,matrix}
\begin{tikzpicture}
\matrix[matrix of nodes, draw=none, row sep=1em, column sep=1em,
every node/.style={draw=gray, shape=circle arrow, ultra thick, inner sep=1em}]
] (m) {
|[circle arrow turn left north]| & |[circle arrow turn left east]| & \\
|[circle arrow turn left west]| & |[circle arrow turn left south]| & \\
|[circle arrow turn right north]| & |[circle arrow turn right east]| & \\
|[circle arrow turn right west]| & |[circle arrow turn right south]| & \\
};
\end{tikzpicture}
```



```
\usetikzlibrary {ext.shapes.circlearrow}
\begin{tikzpicture}\Huge
\node[name=s, shape=circle arrow,
circle arrow turn left west, shape example]
{Circle Arrow\vrule width 1pt height 2cm};
\foreach \anchor/\placement in
{north west/above left, north/above, north east/above right,
west/left, center/above, east/right,
mid west/right, mid/above, mid east/left,
base west/left, base/below, base east/right,
south west/below left, south/below, south east/below right,
text/left, 10/right, 130/above}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```


15 Shape: Circle Cross Split

TikZ Library `ext.shapes.circlecrosssplit`

```
\usepgflibrary{ext.shapes.circlecrosssplit} % LATEX and plain TEX and pure pgf
\usepgflibrary[ext.shapes.circlecrosssplit] % ConTEXt and pure pgf
\usetikzlibrary{ext.shapes.circlecrosssplit} % LATEX and plain TEX when using TikZ
\usetikzlibrary[ext.shapes.circlecrosssplit] % ConTEXt when using TikZ
```

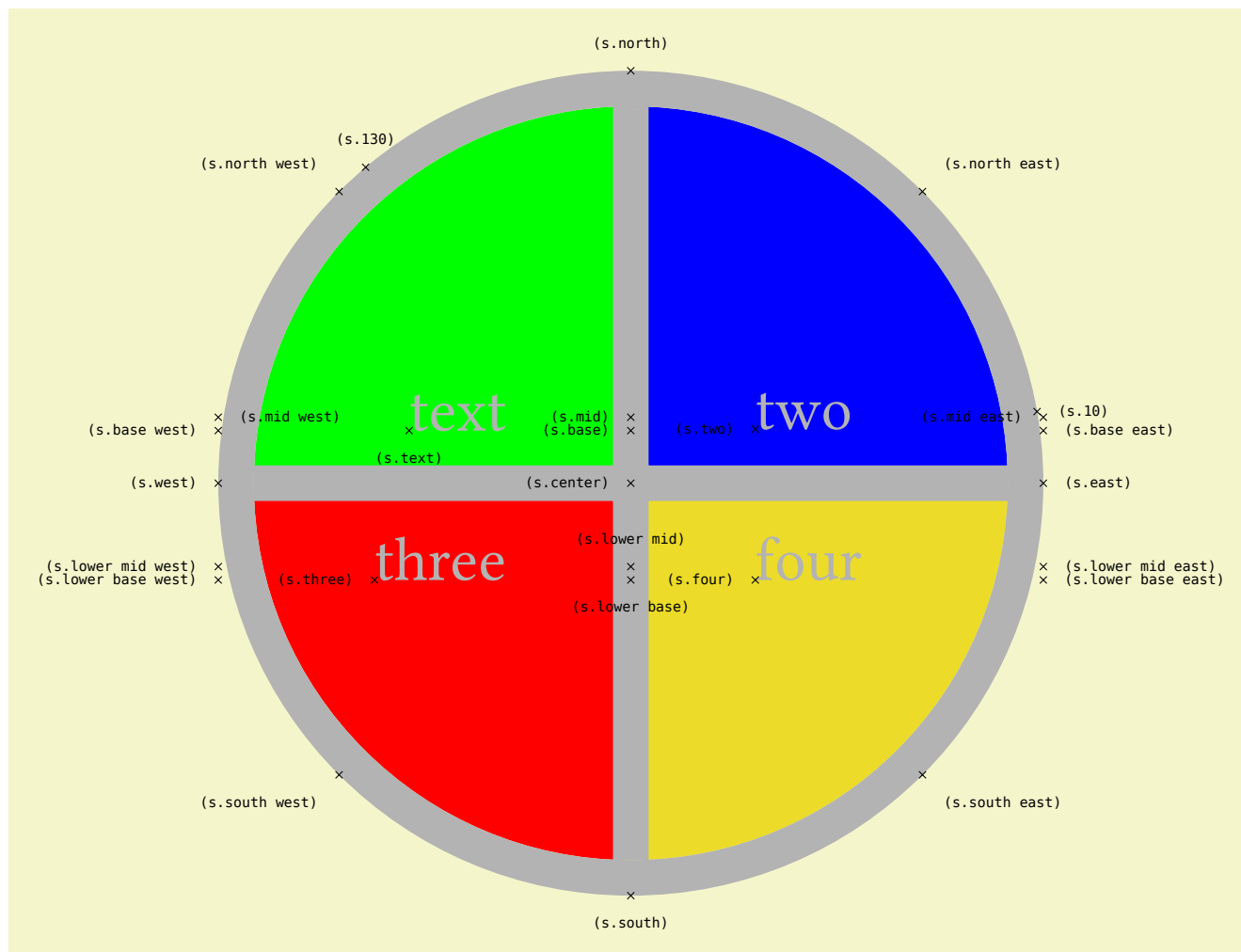
A circular shape with four parts that can be individually filled.

`/pgf/circle cross split part fill={⟨list⟩}` (no default, initially none)

Sets the custom fill color for each node part shape. The items in *⟨list⟩* should be separated by commas (so if there is more than one item in *⟨list⟩*, it must be surrounded by braces). If *⟨list⟩* has less entries than node parts, then the remaining node parts use the color from the last entry in the list. This key will automatically set `/pgf/circle cross split uses custom fill`.

`/pgf/circle cross split uses custom fill=⟨boolean⟩` (default true)

This enables the use of a custom fill for each of the node parts (including the area covered by the inner sep). The background path for the shape should not be filled (e. g., in TikZ, the `fill` option for the node must be implicitly or explicitly set to none). Internally, this key sets the T_EX-if `\ifpgfcirclecrosssplitcustomfill` appropriately.



```

\usepgflibrary {ext.shapes.circlecrosssplit}
\begin{tikzpicture}\Huge
\node[name=s, shape=circle cross split, shape example, inner xsep=1.5cm, fill=none,
circle cross split part fill={green,blue,red,yellow!90!black}]
{\nodepart{text}text\nodepart{two}two
\nodepart{three}three\nodepart{four}four};
\foreach \anchor/\placement in
{north west/above left, north/above, north east/above right,
west/left, center/left, east/right,
mid west/right, mid/left, mid east/left,
base west/left, base/left, base east/right,
lower base west/left, lower base/below, lower base east/right,
lower mid west/left, lower mid/above, lower mid east/right,
south west/below left, south/below, south east/below right,
text/below, 10/right, 130/above, two/left, three/left, four/left}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}

```

16 Shape: Rectangle with Rounded Corners

TikZ Library `ext.shapes.rectangleroundedcorners`

```
\usepgflibrary{ext.shapes.rectangleroundedcorners} % LATEX and plain TEX and pure pgf
\usepgflibrary[ext.shapes.rectangleroundedcorners] % ConTEXt and pure pgf
\usetikzlibrary{ext.shapes.rectangleroundedcorners} % LATEX and plain TEX when using TikZ
\usetikzlibrary[ext.shapes.rectangleroundedcorners] % ConTEXt when using TikZ
```

A rectangle with rounded corners.

`/pgf/rectangle with rounded corners north west radius=(dimen)` (no default, initially `.5\pgflinewidth`)

Sets the north west radius to $\langle dimen \rangle$.

`/pgf/rectangle with rounded corners north east radius=(dimen)` (no default, initially `.5\pgflinewidth`)

Sets the north east radius to $\langle dimen \rangle$.

`/pgf/rectangle with rounded corners south west radius=(dimen)` (no default, initially `.5\pgflinewidth`)

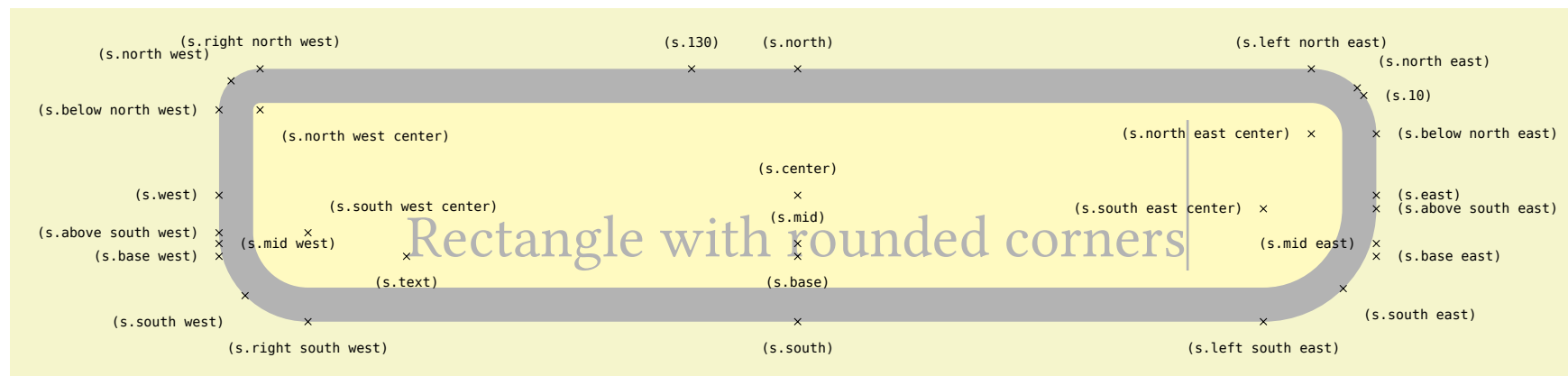
Sets the south west radius to $\langle dimen \rangle$.

`/pgf/rectangle with rounded corners south east radius=(dimen)` (no default, initially `.5\pgflinewidth`)

Sets the south east radius to $\langle dimen \rangle$.

`/pgf/rectangle with rounded corners radius=dimen` (no default)

Sets all radii to $\langle dimen \rangle$.



```

\usepgflibrary {ext.shapes.rectangleroundedcorners}
\begin{tikzpicture}\Huge
\node[name=s, shape=rectangle with rounded corners, shape example,
rectangle with rounded corners north west radius=10pt,
rectangle with rounded corners north east radius=20pt,
rectangle with rounded corners south west radius=30pt,
rectangle with rounded corners south east radius=40pt] {Rectangle with rounded corners\vrule width 1pt height 2cm};
\foreach \anchor/\placement in
{north west/above left, north/above, north east/above right,
west/left, center/above, east/right,
mid west/right, mid/above, mid east/left,
base west/left, base/below, base east/right,
south west/below left, south/below, south east/below right,
text/below, 10/right, 130/above,
north west center/below right, north east center/left,
south west center/above right, south east center/left,
below north west/left, above south west/left, above south east/right, below north east/right,
right north west/above, right south west/below, left south east/below, left north east/above}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}

```

17 Shape: Superellipse

TikZ Library `ext.shapes.superellipse`

```
\usepgflibrary{ext.shapes.superellipse} % LATEX and plain TEX and pure pgf
\usepgflibrary[ext.shapes.superellipse] % ConTEXt and pure pgf
\usetikzlibrary{ext.shapes.superellipse} % LATEX and plain TEX when using TikZ
\usetikzlibrary[ext.shapes.superellipse] % ConTEXt when using TikZ
```

Shape in the form of a “superellipse”.

This shape is defined by formula

$$\left| \frac{x}{r_x} \right|^m + \left| \frac{y}{r_y} \right|^n = 1$$

and will be plotted by

$$x(t) = |\cos t|^{\frac{2}{m}} \cdot r_x \operatorname{sgn}(\cos t)$$

$$y(t) = |\sin t|^{\frac{2}{n}} \cdot r_y \operatorname{sgn}(\sin t)$$

where r_x is half the node’s width and r_y is half the node’s height.

`/pgf/superellipse x exponent=<x exponent>`

(no default, initially 2.5)

This sets m .

`/pgf/superellipse y exponent=<y exponent>`

(no default, initially 2.5)

This sets n .

`/pgf/superellipse step=<step>`

(no default, initially 5)

This specifies the step of the underlying plot handler. The smaller $\langle \text{step} \rangle$ is, the slower computation will be.

Sensible values for $\langle \text{step} \rangle$ are integer dividers of 90, i. e. 2, 3, 5, 6, 9, 10, 15, 18, 30 and 45.

`/pgf/superellipse exponent=<exponent>`

(no default)

Sets both `superellipse x exponent` and `superellipse y exponent` to $\langle \text{exponent} \rangle$.

Notes on Implementation For implementing this shape, additional mathematical functions were declared.

`superellipse(x(t), 2/m, r_x)`

`\pgfmathsuperellipse{t}{2/m}{r_x}`

Returns the x value on a point of the superellipse with its center on the origin following

$$x = r_x \cos^{2/m} t$$

for values of $0 \leq t \leq 90$.

`superellipse(t, 2/n, r_y)`

`\pgfmathsuperellipse{t}{2/n}{r_y}`

Returns the y value on a point of the superellipse with its center on the origin following

$$y = r_y \cos^{2/n} t$$

for values of $0 \leq t \leq 90$.

Both PGFmath functions can be used at once with the following macro.

`\pgfmathsuperellipseXY{t}{2/m}{2/n}{a}{b}`

Returns the x value (in `\pgfmathresultX`) and the y value (in `\pgfmathresultY`) of the superellipse with its center on the origin following

$$x = a \cos^{2/m} t$$

$$y = b \cos^{2/n} t$$

for values of $0 \leq t \leq 90$.

Note: all arguments must be a valid number since they will not be parsed by PGFmath.

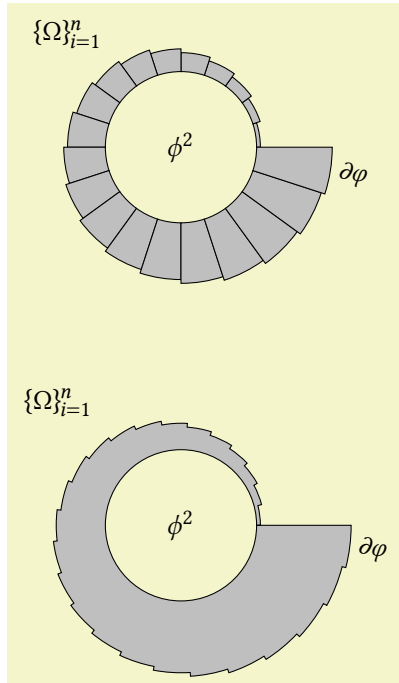
And additional internal macro was defined following the original naming scheme.

`\pgfutil@prefix@macrotomacro{macro 1}{macro 2}`

Adds the once-expansion of `macro 2` in front of `macro 1`.

Part IV

Utilities



```
\usetikzlibrary {ext.misc}
\begin{tikzpicture}[
  declare function={bigR(\n)=smallR+.05*\n;},
  declare constant={smallR=1; segments=20;},
  full arc=segments]
\foreach \iN[evaluate={\endRadius=bigR(\iN+1);}, use int=0 to segments-1]
\filldraw[fill=gray!50] (\iN R:\endRadius)
  arc [radius=\endRadius, start angle=\iN R, delta angle=+IR] -- (\iN R+1R:smallR)
  arc [radius=smallR, end angle=\iN R, delta angle=-IR] -- cycle;

\node                                {${\phi}^2$};
\node at (north west:{sqrt 2 * bigR(segments/2)}) {${\Omega}_{i=1}^n$};
\node[rotate=-.5R, right] at (-.5R: bigR segments) {${\partial} \varphi$};

\tikzset{yshift=-.5cm, declare constant={segments=25;}, full arc=segments}
\filldraw[fill=gray!50] (right:smallR)
  \foreach \iN[evaluate={\endRadius=bigR(\iN+1);}, use int=0 to segments-1] {
    -- (\iN R:\endRadius) arc[radius=\endRadius, start angle=\iN R, delta angle=IR]}
    -- (right:smallR) arc[radius=smallR, start angle=0, delta angle=-360];

\node                                {${\phi}^2$};
\node at (north west:{sqrt 2 * bigR(segments/2)}) {${\Omega}_{i=1}^n$};
\node[rotate=-.5R, right] at (-.5R: bigR segments) {${\partial} \varphi$};
\end{tikzpicture}
```

18 Calendar: Weeknumbers and more conditionals

```
\usepackage{calendar-ext} % LATEX
\input calendar-ext.tex   % plain TEX
\usemodule{calendar-ext}  % ConTEXt
```

This package adds week numbers and more conditionals to the PGF package `pgfcalendar`. (Despite the code example above, this package is not set up to work with ConT_EXt.)

18.1 Extensions

The following tests are added.

- **Jan** This test is passed by all dates that are in the month of January.
- **Feb** as above.
- **Mar** as above.
- **Apr** as above.
- **May** as above.
- **Jun** as above.
- **Jul** as above.
- **Aug** as above.
- **Sep** as above.
- **Oct** as above.
- **Nov** as above.
- **Dec** as above.
- **leap year**= $\langle year \rangle$ This test checks whether the given year is a leap year. If $\langle year \rangle$ is omitted, it checks the year of the current date.
- **and**= $\{\langle tests \rangle\}$ This test passes when all $\langle tests \rangle$ pass.
- **not**= $\{\langle tests \rangle\}$ This test passes when $\langle tests \rangle$ do not pass.
- **yesterday**= $\{\langle tests \rangle\}$ This test passes when the previous day passes $\langle tests \rangle$.

- **week**= $\langle num \rangle$ This test passes when the current week of the year equals $\{\langle num \rangle\}$.

The shorthands for `d-` and `m-` are slightly changed so that they are expandable. This makes it possible to use these shorthands inside of PGFmath. The shorthands for the week (see section 18.2) are added. These are

- `n-` (shortest numerical representation),
- `n=` (shortest but added horizontal space) and
- `n0` (leading zero when below 10).

18.2 Week numbering (ISO 8601)

`\pgfcalendarjulianyeartoweek` $\{\langle julian\ day \rangle\}\{\langle year \rangle\}\{\langle week\ counter \rangle\}$

This command calculates the week for the $\langle julian\ day \rangle$ of $\langle year \rangle$. The $\langle week\ counter \rangle$ must be a T_EX counter.

The calculation follows the rule of ISO 8601 where the first week has that year's first Thursday in it.

Inside of `\pgfcalendar` the command `\pgfcalendarcurrentweek` will be available.

`\pgfcalendarcurrentweek`

This command returns the current week number (always two digits – use shorthand `n.` to strip the leading zero).

Inside of `\ifdate` the command `\pgfcalendarifdateweek` will be available.

`\pgfcalendarifdateweek`

This command returns the week number (always two digits).

19 And a little bit more

TikZ Library `ext.misc`

```
\usetikzlibrary{ext.misc} % LATEX and plain TEX
\usetikzlibrary[ext.misc] % ConTEXt
```

This library adds miscellaneous utilities to PGFmath, PGF or TikZ.

19.1 PGFmath

19.1.1 Postfix operator R

Similar to `\segments[<num>]` in PSTricks, the postfix operator R allows the user to use an arbitrary number of segments of a circle to be used instead of an angle.

`/tikz/full arc=<num>` (default)

The number $\langle num \rangle$ of segments will be set up. Using `full arc` with an empty value disables the segmentation and 1R equals 1°.

The given value $\langle num \rangle$ is evaluated when the key is used and doesn't change when $\langle num \rangle$ contains variables that change.

The R operator can then be used.

`xR` (postfix operator; uses the `fullarc` function)

Multiplies x with $\frac{360}{\langle num \rangle}$.

19.1.2 Functions

`strrepeat("Text", x)`

`\pgfmathstrrepeat{"Text"}{x}`

Returns a string with *Text* repeated x times.

```
foofoofoofoofoo \pgfmathparse{strrepeat("foo", 5)}
\pgfmathresult
```

`isInString("String", "Text")`

`\pgfmathisInString{"String"}{"Text"}`

Returns 1 (true) if *Text* contains *String*, otherwise 0 (false).

0 and 1

```
\pgfmathparse{isInString("foo", "bar")}
\pgfmathresult \ and\
\pgfmathparse{isInString("foo", "foobar")}
\pgfmathresult
```

`strcat("Text A", "Text B", ...)`

`\pgfmathstrcat{"Text A"}{"Text B"}{...}`

Returns the concatenation of all given parameters.

blue!21!green

```
\pgfmathparse{strcat("blue!", int(7*3), "!green")}
\pgfmathresult
```

`isEmpty("Text")`

`\pgfmathisEmpty{"Text"}`

Returns 1 (true) if *Text* is empty, otherwise 0 (false).

0 and 1 and 1

```
\pgfmathparse{isEmpty("foo")} \pgfmathresult\ and\
\pgfmathparse{isEmpty("")} \pgfmathresult\ and\
\def\emptyText{}
\pgfmathparse{isEmpty("\emptyText")} \pgfmathresult
```

`atanXY(x,y)`

`\pgfmathatanXY{x}{y}`

Arctangent of $y \div x$ in degrees. This also takes into account the quadrant. This is just a argument-swapped version of `atan2` which makes it easier to use the `\p` commands of the `calc` library.

53.13011

```
\pgfmathparse{atanXY(3,4)} \pgfmathresult
```

atanYX(y, x)

`\pgfmathatanYX{y}{x}`

Arctangent of $y \div x$ in degrees. This also takes into account the quadrant.

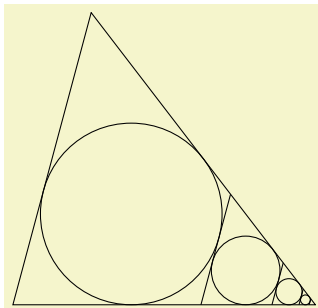
53.13011 `\pgfmathparse{atanYX(4,3)} \pgfmathresult`

19.1.3 Functions: using coordinates

The following functions can only be used with PGF and/or TikZ. Since the arguments are usually plain text (and not numbers) one has to wrap them in `"`.

anglebetween(`"p1"`, `"p2"`)

`\pgfmathanglebetween{"p1"}{"p2"}`



```
\usetikzlibrary {calc,ext.misc,through}
\begin{tikzpicture}
\path (0,0) coordinate (A) + (0:4) coordinate (B) +(75:4) coordinate (C);
\draw (A) -- (B) -- (C) -- cycle;
\foreach \cnt in {1,...,4}{
  \pgfmathsetmacro\triA{distancebetween("B","C")}
  \pgfmathsetmacro\triB{distancebetween("C","A")}
  \pgfmathsetmacro\triC{distancebetween("A","B")}
  \path (barycentric cs:A=\triA,B=\triB,C=\triC) coordinate (M)
    node [draw, circle through=($(A)!(M)!(C)$)] (M) {};
  \draw ($(C)-(A)$) coordinate (vecB)
    (M.75-90) coordinate (@)
    (intersection of @--[shift=(vecB)]@ and B--C) coordinate (C) --
    (intersection of @--[shift=(vecB)]@ and B--A) coordinate (A);}
\end{tikzpicture}
```

Return the angle between the centers of the nodes `p1` and `p2`.

qanglebetween(`"p"`)

`\pgfmathqanglebetween{"p"}`

Return the angle between the origin and the center of the node `p`.

distancebetween(`"p1"`, `"p2"`)

`\pgfmathdistancebetween{"p1"}{"p2"}`

Return the distance (in pt) between the centers of the nodes `p1` and `p2`.

qdistancebetween(`"p"`)

`\pgfmathqdistancebetween{"p"}`

Return the distance (in pt) between the origin and the center of the node `p`.

19.2 PGFkeys

19.2.1 Conditionals

`/utils/if=<cond><true><false>`

(no default)

This key checks the conditional `<cond>` and applies the styles `<true>` if `<cond>` is true, otherwise `<false>`. `<cond>` can be anything that PGFmath understands.

As a side effect on how PGFkeys parses argument, the `<false>` argument is actu-

ally optional.

The following keys use TeX' macros `\if`, `\ifx`, `\ifnum` and `\ifdim` for faster executions.

`/utils/TeX/if=<token A><token B><true><false>`

(no default)

This key checks via `\if` if $\langle token A \rangle$ matches $\langle token B \rangle$ and applies the styles $\langle true \rangle$ if it does, otherwise $\langle false \rangle$.

As a side effect on how PGFkeys parses argument, the $\langle false \rangle$ argument is actually optional.

`/utils/TeX/iffx= $\langle token A \rangle \langle token B \rangle \langle true \rangle \langle false \rangle$` (no default)

As above.

`/utils/TeX/ifnum= $\langle num cond \rangle \langle true \rangle$
opt $\langle false \rangle$` (no default)

This key checks `\ifnum $\langle num cond \rangle$` and applies the styles $\langle true \rangle$ if true, otherwise $\langle false \rangle$. A delimiting `\relax` will be inserted after $\langle num cond \rangle$.

As a side effect on how PGFkeys parses argument, the $\langle false \rangle$ argument is actually optional.

`/utils/TeX/ifdim= $\langle dim cond \rangle \langle true \rangle \langle false \rangle$` (no default)

As above.

`/utils/TeX/isempty= $\langle Text \rangle \langle true \rangle \langle false \rangle$` (no default)

This checks whether $\langle Text \rangle$ is empty and applies styles $\langle true \rangle$ if true, otherwise $\langle false \rangle$.

19.2.2 Handlers

While already a lot of values given to keys are evaluated by PGFmath at some point, not all of them are.

Key handler $\langle key \rangle/.pgfmath= $\langle eval \rangle$$

This handler evaluates $\langle eval \rangle$ before it is handed to the key.

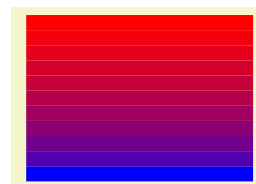
Key handler $\langle key \rangle/.pgfmath int= $\langle eval \rangle$$

As above but truncates the result.

Key handler $\langle key \rangle/.pgfmath strcat= $\langle eval \rangle$$

As above but uses the `strcat` function.

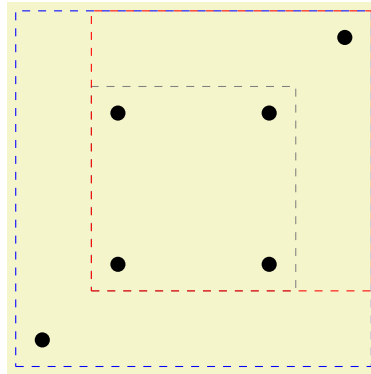
In the example below, one could have used the `/pgf/foreach/evaluate` key from `\foreach`.



```
\usetikzlibrary {misc}
\tikz\foreach \i in {0,10,...,100}
\draw[
  line width=+.2cm,
  color/.pgfmath strcat={"red!",sqrt(\i)*10,"!blue"}
]
(0,\i/50) -- +(right:3);
```

Key handler $\langle key \rangle/.List= $\langle e1 \rangle, \langle e2 \rangle, ..., \langle en \rangle$$

This handler evaluates the given list with `\foreach` and concatenates the element and the result is then given to the used key.



```
\usetikzlibrary {fit,ext.misc}
\begin{tikzpicture}[nodes={draw, dashed, inner sep=+10pt}]
\foreach \point [count=\cnt] in {(0,0), (0,2), (2,0), (2,2), (3,3), (-1,-1)}
\fill \point circle[radius=.1] coordinate (point-\cnt);
\node[gray, fit/.List={(point-1),(point-...),(point-4)}] {};
\node[red, fit/.List={(point-1),(point-...),(point-5)}] {};
\node[blue, fit/.List={(point-1),(point-...),(point-6)}] {};
\end{tikzpicture}
```

19.3 PGFfor

Instead of `\foreach \var in {start, start + delta, ..., end}` one can use `\foreach \var[use int=start to end step delta]`.

`/pgf/foreach/use int=<start>to<end>step<delta>`

(no default)

The values $\langle start \rangle$, $\langle end \rangle$ and $\langle delta \rangle$ are evaluated by PGFmath at initialization. The part step $\langle delta \rangle$ is optional ($\langle delta \rangle = 1$).

`/pgf/foreach/use float=<start>to<end>step<delta>`

(no default)

Same as above, however the results are not truncated.

Part V

Changelog & Index

Changelog

Version 0.3

- Added shape `superellipse` (PGF library `ext.shapes.superellipse`).

Version 0.2

- Added TikZ library `ext.positioning-plus`.
- Added TikZ library `ext.node-families`.

Version 0.1

- Added TikZ library `ext.calendar-plus`.
- Added TikZ library `ext.misc`.

- Added TikZ library `ext.paths.arcto`.
- Added TikZ library `ext.paths.ortho`.
- Added TikZ library `ext.paths.timer`.
- Added TikZ library `ext.patterns.images`.
- Added TikZ library `ext.topaths.arctthrough`.
- Added TikZ library `ext.transformations.mirror`.
- Added PGF library `ext.transformations.mirror`.

Index

This index contains automatically generated entries as well as *references* to original functionalities of PGF/TikZ.

- | - | path operation, 11
- | - path operation, 11
- | path operation, 11
- | - path operation, 11

- above key, 20
- above left key, 18
- above right key, 18
- and date test, 42
- anglebetween math function, 44
- Apr date test, 42
- arc through key, 23
- arc to path operation, 9
- atan2 math function, 28, 43
- atanXY math function, 43
- atanYX math function, 44
- Aug date test, 42

- below key, 20
- below left key, 18
- below right key, 18

- calc library, 43
- calendar library, 6
- calendar-ext package, 42
- center suffix key, 23
- circle shape, 7
- circle arrow arrows key, 30
- circle arrow delta angle key, 30
- circle arrow end angle key, 30
- circle arrow start angle key, 30
- circle arrow turn left east key, 30
- circle arrow turn left north key, 30
- circle arrow turn left south key, 30
- circle arrow turn left west key, 30
- circle arrow turn right east key, 30
- circle arrow turn right north key, 30
- circle arrow turn right south key, 31
- circle arrow turn right west key, 31
- circle cross split part fill key, 33
- circle cross split uses custom fill key, 33
- clockwise key, 9, 23
- corner above left key, 18
- corner above right key, 18
- corner below left key, 18
- corner below right key, 18
- corner east above key, 21
- corner east below key, 21
- corner north left key, 21
- corner north right key, 21
- corner south left key, 21
- corner south right key, 21
- corner west above key, 21
- corner west below key, 21
- cos math function, 28
- cos path operation, 16
- counter clockwise key, 10, 23

- Date tests
 - and, 42
 - Apr, 42
 - Aug, 42
 - Dec, 42
 - Feb, 42
 - Jan, 42
 - Jul, 42
 - Jun, 42
 - leap year, 42
 - Mar, 42
 - May, 42
 - not, 42
 - Nov, 42

- Oct, 42
- Sep, 42
- week, 42
- yesterday, 42
- day code key, 6
- day text key, 6
- day xshift key, 6
- day yshift key, 6
- Dec date test, 42
- distance key, 11, 14
- distancebetween math function, 44
- du distance key, 13
- east above key, 20
- east below key, 20
- edge path operation, 23
- every arc to key, 10
- every day key, 6
- every month key, 6
- every week key, 6
- execute at end picture key, 7
- ext.calendar-plus library, 6
- ext.misc library, 43
- ext.node-families library, 7
- ext.paths.arcto library, 9
- ext.paths.ortho library, 11
- ext.paths.timer library, 15
- ext.patterns.images library, 17
- ext.positioning-plus library, 18
- ext.shapes.circulararrow library, 30
- ext.shapes.circlecrosssplit library, 33
- ext.shapes.rectangleroundedcorners library, 36
- ext.shapes.superellipse library, 38
- ext.topaths.arctthrough library, 23
- ext.transformations.mirror library, 24, 28
- external library, 7
- Feb date test, 42
- fit library, 18
- fit bounding box key, 22
- from center key, 12, 14

- full arc key, 43
- height key, 7
- horizontal vertical key, 14
- horizontal vertical horizontal key, 14
- if key, 6, 44
- \ifdate, 42
- ifdim key, 45
- ifempty key, 45
- ifnum key, 45
- ifx key, 45
- image as pattern key, 17
- isEmpty math function, 43
- isInString math function, 43
- Jan date test, 42
- Jul date test, 42
- Jun date test, 42
- Key handlers
 - .List, 45
 - .pgfmath, 45
 - .pgfmath int, 45
 - .pgfmath strcat, 45
- large key, 10
- leap year date test, 42
- left key, 19, 20
- Libraries
 - calc, 43
 - calendar, 6
 - ext.calendar-plus, 6
 - ext.misc, 43
 - ext.node-families, 7
 - ext.paths.arcto, 9
 - ext.paths.ortho, 11
 - ext.paths.timer, 15
 - ext.patterns.images, 17
 - ext.positioning-plus, 18
 - ext.shapes.circulararrow, 30
 - ext.shapes.circlecrosssplit, 33

- ext.shapes.rectangleroundedcorners, 36
- ext.shapes.superellipse, 38
- ext.topaths.arctthrough, 23
- ext.transformations.mirror, 24, 28
- external, 7
- fit, 18
- positioning, 18
- .List handler, 45
- lr distance key, 13
- Mar date test, 42
- Math functions
 - anglebetween, 44
 - atan2, 28, 43
 - atanXY, 43
 - atanYX, 44
 - cos, 28
 - distancebetween, 44
 - isEmpty, 43
 - isInString, 43
 - qanglebetween, 44
 - qdistancebetween, 44
 - sin, 28
 - strcat, 43
 - strrepeat, 43
 - superellipse, 38
 - superellipse, 39
- Math operators
 - R, 43
- May date test, 42
- middle 0 to 1 key, 13
- minimum height key, 7
- minimum width key, 7
- Mirror key, 26
- mirror key, 25
- Mirror x key, 26
- mirror x key, 25
- Mirror y key, 26
- mirror y key, 25
- month code key, 6
- month text key, 6

- month xshift key, 6
- month yshift key, 6

- name key, 17
- north left key, 19
- north right key, 20
- not date test, 42
- Nov date test, 42

- Oct date test, 42
- on grid key, 22
- only horizontal first key, 14
- only horizontal second key, 14
- only vertical first key, 14
- only vertical second key, 14
- option key, 17
- options key, 17

Packages and files

- calendar-ext, 42
- parabola path operation, 15

Path operations

- |-, 11
- |-, 11
- |, 11
- |-, 11
- arc to, 9
- cos, 16
- edge, 23
- parabola, 15
- r-du, 13
- r-lr, 13
- r-rl, 13
- r-ud, 13
- rectangle, 15
- sin, 16
- to, 23

/pgf/

- circle arrow arrows, 30
- circle arrow delta angle, 30
- circle arrow end angle, 30

circle arrow start angle, 30
 circle arrow turn left east, 30
 circle arrow turn left north, 30
 circle arrow turn left south, 30
 circle arrow turn left west, 30
 circle arrow turn right east, 30
 circle arrow turn right north, 30
 circle arrow turn right south, 31
 circle arrow turn right west, 31
 circle cross split part fill, 33
 circle cross split uses custom fill, 33
 foreach/
 use float, 46
 use int, 46
 minimum height, 7
 minimum width, 7
 rectangle with rounded corners north east radius, 36
 rectangle with rounded corners north west radius, 36
 rectangle with rounded corners radius, 36
 rectangle with rounded corners south east radius, 36
 rectangle with rounded corners south west radius, 36
 superellipse exponent, 38
 superellipse step, 38
 superellipse x exponent, 38
 superellipse y exponent, 38
 text, 17
 \pgfcalendar, 42
 \pgfcalendarcurrentweek, 42
 \pgfcalendarifdateweek, 42
 \pgfcalendarjulianyeartoweek, 42
 .pgfmath handler, 45
 .pgfmath int handler, 45
 .pgfmath strcat handler, 45
 \pgfmathanglebetween, 44
 \pgfmathanglebetween, 28
 \pgfmathatanXY, 43
 \pgfmathatanYX, 44
 \pgfmathdistancebetween, 44
 \pgfmathisEmpty, 43
 \pgfmathisInString, 43
 \pgfmathqanglebetween, 44
 \pgfmathqdistancebetween, 44
 \pgfmathstrcat, 43
 \pgfmathstrrepeat, 43
 \pgfmathsuperellipse, 39
 \pgfmathsuperellipseXY, 39
 \pgfmathsuperellipsey, 39
 pgfminimum height, 22
 pgfminimum height key, 22
 pgfminimum width, 22
 pgfminimum width key, 22
 \pgfpatharcto, 10
 \pgfpointnormalised, 28
 \pgfqtransformMirror, 29
 \pgfqtransformmirror, 29
 \pgfsetarrows, 30
 \pgfsetupimageaspattern, 17
 \pgftext, 17
 \pgftransformMirror, 29
 \pgftransformmirror, 29
 \pgftransformxMirror, 28
 \pgftransformxmirror, 28
 \pgftransformyMirror, 29
 \pgftransformymirror, 29
 \pgfutil@prefix@macrotomacro, 39
 pos key, 12, 15
 positioning library, 18
 prefix key, 8

 qanglebetween math function, 44
 qdistancebetween math function, 44

 R postfix math operator, 43
 r-du path operation, 13
 r-lr path operation, 13
 r-rl path operation, 13
 r-ud path operation, 13
 radius key, 10
 ratio key, 11
 rectangle path operation, 15
 rectangle shape, 7
 rectangle with rounded corners north east radius key, 36

- rectangle with rounded corners north west radius key, 36
- rectangle with rounded corners radius key, 36
- rectangle with rounded corners south east radius key, 36
- rectangle with rounded corners south west radius key, 36
- right key, 20
- rl distance key, 13
- rotate key, 10
- Sep date test, 42
- setup shape key, 8
- Shapes
 - circle, 7
 - rectangle, 7
- sin math function, 28
- sin path operation, 16
- size key, 7
- small key, 10
- south left key, 20
- south right key, 20
- spacing key, 12
- span key, 22
- span horizontal key, 22
- span vertical key, 22
- strcat math function, 43
- strrepeat math function, 43
- superellipse exponent key, 38
- superellipse step key, 38
- superellipse x exponent key, 38
- superellipse y exponent key, 38
- superellipse x math function, 38
- superellipse y math function, 39
- text key, 17
- text key, 8
- text depth key, 7
- text height key, 7
- text width key, 8
- text width align key, 8
- through key, 23
- /tikz/
 - above, 20

- above left, 18
- above right, 18
- arc through/
 - center suffix, 23
 - clockwise, 23
 - counter clockwise, 23
 - through, 23
- arc through, 23
- arc to/
 - clockwise, 9
 - counter clockwise, 10
 - large, 10
 - radius, 10
 - rotate, 10
 - small, 10
 - x radius, 10
 - y radius, 10
- below, 20
- below left, 18
- below right, 18
- corner above left, 18
- corner above right, 18
- corner below left, 18
- corner below right, 18
- corner east above, 21
- corner east below, 21
- corner north left, 21
- corner north right, 21
- corner south left, 21
- corner south right, 21
- corner west above, 21
- corner west below, 21
- day code, 6
- day text, 6
- day xshift, 6
- day yshift, 6
- east above, 20
- east below, 20
- every arc to, 10
- every day, 6
- every month, 6

- every week, 6
- execute at end picture, 7
- fit bounding box, 22
- full arc, 43
- horizontal vertical, 14
- horizontal vertical horizontal, 14
- hvvh/
 - distance, 11
 - from center, 12
 - middle 0 to 1, 13
 - ratio, 11
 - spacing, 12
- if, 6
- image as pattern/
 - name, 17
 - option, 17
 - options, 17
- image as pattern, 17
- left, 19, 20
- Mirror, 26
- mirror, 25
- Mirror x, 26
- mirror x, 25
- Mirror y, 26
- mirror y, 25
- month code, 6
- month text, 6
- month xshift, 6
- month yshift, 6
- node family/
 - height, 7
 - prefix, 8
 - setup shape, 8
 - size, 7
 - text, 8
 - text depth, 7
 - text height, 7
 - text width, 8
 - text width align, 8
 - width, 7
- north left, 19

- north right, 20
- on grid, 22
- only horizontal first, 14
- only horizontal second, 14
- only vertical first, 14
- only vertical second, 14
- pos, 12, 15
- right, 20
- south left, 20
- south right, 20
- span, 22
- span horizontal, 22
- span vertical, 22
- to path, 14
- udlr/
 - distance, 14
 - du distance, 13
 - from center, 14
 - lr distance, 13
 - rl distance, 13
 - ud distance, 13
- vertical horizontal, 14
- vertical horizontal vertical, 14
- week code, 6
- week label left, 6
- week text, 6
- west above, 20
- west below, 20
- x radius, 10
- xMirror, 26
- xmirror, 24
- y radius, 10
- yMirror, 26
- ymirror, 25
- to path operation, 23
- to path key, 14
- ud distance key, 13
- use float key, 46
- use int key, 46
- /utils/

- if, 44
- TeX/
 - if, 44
 - ifdim, 45
 - ifempty, 45
 - ifnum, 45
 - ifx, 45

- vertical horizontal key, 14
- vertical horizontal vertical key, 14

- week date test, 42
- week code key, 6
- week label left key, 6
- week text key, 6
- west above key, 20
- west below key, 20
- width key, 7

- x radius key, 10
- x radius key, 10
- xMirror key, 26
- xmirror key, 24

- y radius key, 10
- y radius key, 10
- yesterday date test, 42
- yMirror key, 26
- ymirror key, 25