The Ti*k*Z-Extensions Package

Manual for version 0.6 (8)

`https://github.com/Qrrbrbirlbel/tikz-extensions`

Qrrbrbirlbel

October 30, 2024

Contents

Part I
Introduction

1 Usage

This package is called `tikz-ext`, however, one can't load it via `\usepackage`.[1] Instead, this package consists mostly of PGF and Ti*k*Z 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 solutions of my answers in this package.

3 Having problems?

Note however, that most of these extensions haven't been stress-tested properly and might be considered experimental.

Don't hesitate to open an issue on GitHub. You probably found a bug.

4 Namespaces and Ti*k*Z-Extensions macros

Since some parts of this package have existed in some form since 2013, the choice for key names and in which PGFkeys namespace they reside is not always optimal. They often reside in the main `/tikz` or `/pgf` path. Similar applies to macro names.

For future versions, it is planned to move those in the `/tikz/ext` namespace. For keys in the `/pgf` namespace, this will probably not happen since it makes it not very intuitive to use them in Ti*k*Z.

Starting from version 0.6, Ti*k*Z-Extensions provides commands that return the current version for compatibility testing. The second simply increments with every release so that the first doesn't need to be parsed.

`\tikzextversion`

Returns `0.6`.

`\tikzextversionnumber`

Returns `8`.

Also starting from version 0.6, there's `\tikzextset` and `\pgfextset`.

`\tikzextset{`⟨*options*⟩`}`

This command will process the ⟨*options*⟩ using the `\pgfkeys` command with the default path set to `/tikz/ext`.

`\pgfextset{`⟨*options*⟩`}`

This command will process the ⟨*options*⟩ using the `\pgfkeys` command with the default path set to `/pgf/ext`.

[1] Except for `pgfcalendar-ext` and `pgffor-ext`.

5 Compatibility with older versions

As discussed in the previous section, keys and commands of extensions that existed before version 0.6 that do not appear in this manual are considered deprecated.

`/tikz/ext/compat=pre 0.6|0.6|warn|newest` (default `pre 0.6`)

This sets the global compatibility setting for every extension of this package (whether already loaded or not).

The choice `warn` gives out warning for deprecated keys or commands but still executes them if they were not not in use when an extension was loaded.

For version 0.6 this is actually the default settings so that active documents keep working – for now.

The following table shows the compatibility settings for each extension. A ✓ denotes an available setting where ✓✓ denotes the default compatibility setting. A – denotes that it is not different than the `newest` setting.

Extension	warn	pre 0.6	0.6
`pgfcalendar-ext` `ext.calendar-plus`	✓✓	✓	–
`ext.arrows`	✓✓	✓	–
`ext.layers`	✓✓	✓	–
`ext.node-families`	✓✓	✓	–
`ext.nodes`	✓✓	✓	–
`ext.paths.arcto`	✓✓	✓	–
`ext.paths.ortho`	✓✓	✓	–
`ext.paths.timer`	✓✓	✓	–
`ext.pgffor-ext`	✓✓	✓	–
`ext.pgfkeys-plus`	✓✓	✓	–
`ext.positioning-plus`	✓✓	✓	–
`ext.scalepicture`	✓✓	✓	–
`ext.shapes`	✓✓	✓	–
`ext.transformations.mirror`	✓✓	✓	–
`ext.topaths.arcthrough`	✓✓	✓	–

For each available extension the compatibility setting can be adjusted as well after the extension is loaded.

`/tikz/ext/compat/pgfcalendar-ext=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/arrows=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/layers=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/nodes=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/node-families=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/paths.arcto=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/paths.ortho=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/paths.timer=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/pgffor-ext=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/pgfkeys-plus=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/positioning-plus=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/scalepicture=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/shapes=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/transformations.mirror=`⟨*version*⟩ (default `pre 0.6`)
`/tikz/ext/compat/topaths.arcthrough=`⟨*version*⟩ (default `pre 0.6`)

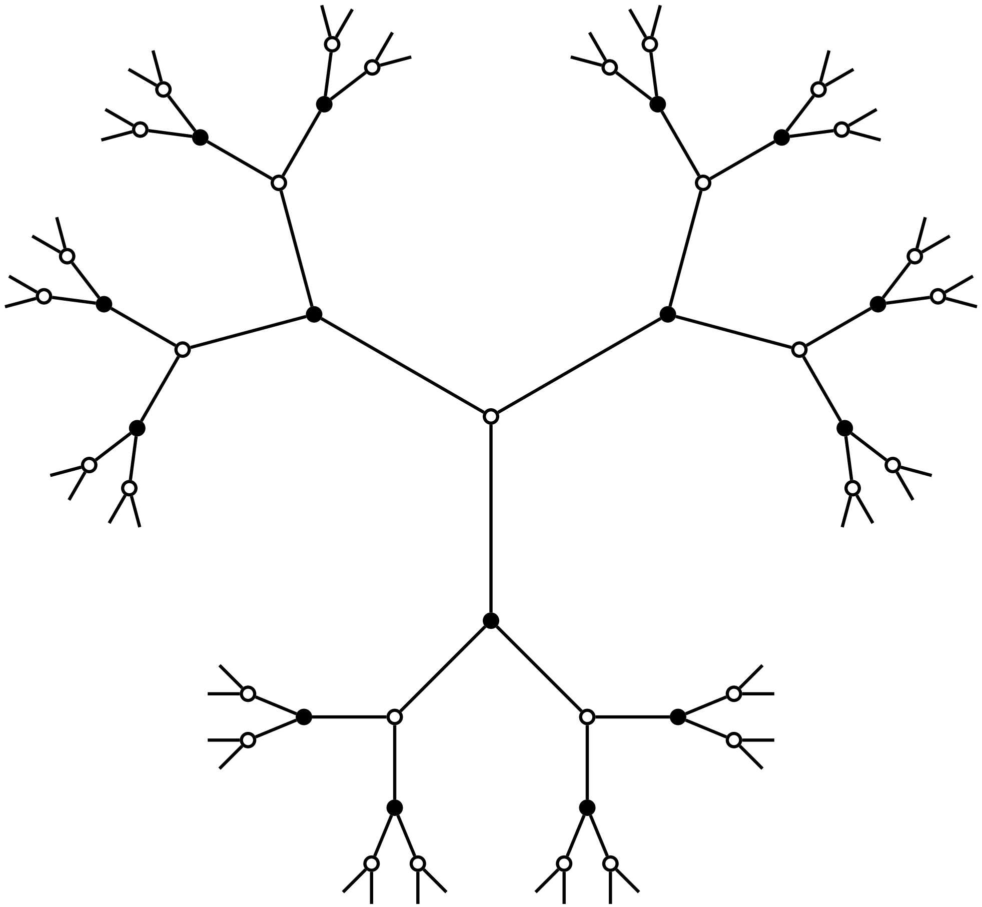
For ⟨*version*⟩ the same choices are valid as for the main `compat` key. It should be noted that at this point, a compatibility setting can't really be reversed since they only forward arguments from an old key or command to the new version.

The old names are given as a subtitle to the new one in the sections that introduce them.

Part II

Ti*k*Z Libraries

These libraries only work with Ti*k*Z.



6 Arrow Pics

TikZ Library `ext.arrows-plus`

```
\usetikzlibrary{ext.arrows-plus} % LaTeX and plain TeX
\usetikzlibrary[ext.arrows-plus] % ConTeXt
```

This library defines pics and keys that can be used to place (bended) arrow tips on paths.

The `markings` decoration already provides the functionality to place arrow tips along the path. The pics and keys provided by this library serve as an alternative.

Many of the pics and keys share various keys that specify where and how the arrow tips are placed.

`/tikz/ext/pos <=`⟨*value*⟩ (no default, initially `0.0`)

If the pic type supports it and a start arrow tip sequence is provided this specifies the position of that sequence.

`/tikz/ext/pos >=`⟨*value*⟩ (no default, initially `0.5`)

This is an alias for `/tikz/pos` , if an end arrow tip sequence is provided, it is placed at this position.

`/tikz/ext/arrow shift mode=`⟨*shift mode*⟩ (no default, initially `total length`)

This key is used to set the ⟨*shift mode*⟩ for the arrow tip. It can be one of the following.

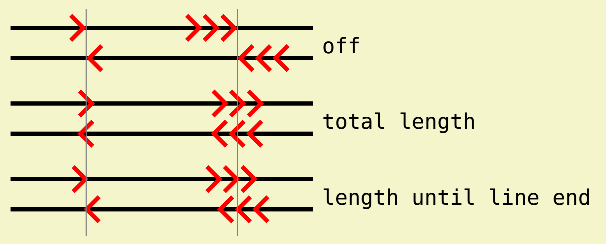
`arrow shift mode=off` This disables the shifting.

`arrow shift mode=total length` The total length of the whole arrow tip sequence will be used.

`arrow shift mode=total` This is an alias for `total length`.

`arrow shift mode=length until line end` The length of the whole arrow tip until the line end will be used – as reported by PGF which might not always be the expected one.

`arrow shift mode=line end` This is an alias for `length until line end`.



```
\usetikzlibrary {ext.arrows-plus}
\begin{tikzpicture}[>={Straight Barb[color=red]}, ultra thick]
\ttfamily
\foreach[count=\y] \shiftmode in {off, total length, length until line end}
  \draw[ext/arrow shift mode=\shiftmode] (0, -\y  )
                 -- pic {ext/arrow=>}    ++(right:2)
                 -- pic {ext/arrow=>.>>} ++(right:2) node[below right] {\shiftmode}
    ++(down:.4) -- pic {ext/arrow=>.>>} ++( left:2)
                 -- pic {ext/arrow=>}    ++( left:2);
\draw[thin, gray] (1,-.75) -- +(down:3) (3,-.75) -- +(down:3);
\end{tikzpicture}
```

For single arrow tips it might be better to use the Centered arrow tip variants of the `ext.arrows` library (see sec 20) and disabled `arrow shift mode`.

When an arrow tip sequence is to be drawn depending on the shift mode its total length or its length until the line end will be determined and multiplied with the `arrow shift factor`. The result of this evaluation is used to shift the arrow tip sequence in the tip's direction.

`/tikz/ext/arrow shift factor=`⟨*value*⟩ (no default, initially `0.5`)

This determines the shift factor.

The default value is probably good for most cases.

6.1 Arrow pic types

This library provides the following pics:

ext/arrow This is the simplest implementation to place an arrow tip along a path. It uses the current timer that is also used to place nodes.

It can be used without any adjustment for every path operation that provides such a timer. These do *not* include circle, ellipse, plot and grid. For rectangle, parabola, sin and cos, the ext.paths.timer library is recommended or even necessary (see section 13).

The arrow tips will never be bended. For this the following pic types or the /tikz/ext/arc arrows key will be necessary.

Due to [**PgfIssueSloped**] with an active transformation, the arrow tips won't be placed correctly in many cases. For this *and* bended arrow tips the following pics are necessary.

ext/softpath arrows This pic type places a possible bended arrow tip on the last segment of the path.

For the path operators --, |- and -| this works even with a non-identity transformation. If possible, the current timer will be used to take more segments into account so that the arrow tip can be placed along the recent path operation.

This won't work for arcs, for this the /tikz/ext/arc arrows key will be necessary.

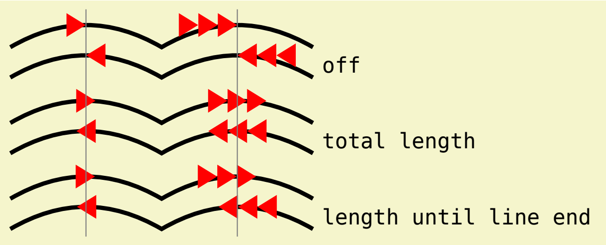
This pic type can place two tip specification, one at pos > and one at pos < in the reversed direction.

ext/softpath arrow This is an alias for softpath arrows with an empty start arrow tip specification.

Pic type ext/arrow=⟨*arrow tip specification*⟩

This pic draws the given ⟨*arrow tip specification*⟩ (defaults to the end tip specification of the path).

This obviously is best use as a pic along a path segment that supports it. It *does not* support bended arrow tips.



```
\usetikzlibrary {bending, ext.arrows-plus}
\begin{tikzpicture}[>={Triangle[color=red]}, arrows={[bend]}, ultra thick]
\ttfamily
\foreach[count=\y] \shiftmode in {off, total length, length until line end}
  \draw[ext/arrow shift mode=\shiftmode] (0, -\y  )
                to[bend  left] pic {ext/arrow=>}    ++(right:2)
                to[bend  left] pic {ext/arrow=>>.>} ++(right:2)
                                          node[below right] {\shiftmode}
    ++(down:.4) to[bend right] pic {ext/arrow=>>.>} ++( left:2)
                to[bend right] pic {ext/arrow=>}    ++( left:2);
\draw[thin, gray] (1,-.5) -- +(down:3) (3,-.5) -- +(down:3);
\end{tikzpicture}
```

Pic type ext/softpath arrows=⟨*start tip specification*⟩-⟨*end tip specification*⟩

This pic draws the given arrow tip specification (defaults to the already present tip specification of the path) along the previous path segment (a curve or a line). It supports the pos < key.

Note: For arcs with an angle greater than 90° this will not work as expected. Use the arc arrows key instead.

Pic type softpath arrow=⟨*end tip specification*⟩

This pic type is an alias for softpath arrows = -⟨*end tip specification*⟩.

6.2 Arrow keys

The last pic type softpath arrows is also available as a key which is the preferred version.

/tikz/ext/softpath arrows=⟨*options*⟩ (default ->)

This key adds arrow tips to the previous path segment (a curve or a line).

`/tikz/ext/every softpath arrows` (style, initially `{}`)

This style will be applied for every instance of `softpath arrows` (key version, not the pic).

For `arcs` the following key needs to be used directly after the `arc` path operation.

`/tikz/ext/arc arrows=`⟨*options*⟩ (default `->`)

This key adds arrow tips to the previous arc segment.

`/tikz/ext/every softpath arrows` (style, initially `{}`)

This style will be applied for every instance of `arc arrows`.

Tip: Use an arc with the full 360° to place bended arrow tips along a circle or an ellipse.

6.3 Shifted and bended arrows for the `decorations.markings` library

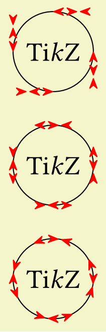
Many paths are not properly accessible by the previous methods. If this library is loaded *after* the `decorations.markings` library, both the `\arrow` and the `\arrowreversed` macros are enhanced.

`\arrow**[`⟨*options*⟩`]{`⟨*arrow end tip*⟩`}`

This macro works the same as before but the one-starred version applies the shifting as specified by `arrow shift mode` and `arrow shift factor` where as the two-starred version also bends the arrow tip.

`\arrowreversed**[`⟨*options*⟩`]{`⟨*arrow end tip*⟩`}`

As above, only the arrow end tip is flipped and points in the other direction.



```
\usetikzlibrary {bending, decorations.markings, ext.arrows-plus}
\tikzset{
  arr/.style={
    postaction=decorate,
    decoration={
      name=markings,
      mark={between positions .25 and 1 step .25 with
        \arrow#1[red]{> _ < _ >}}}}}
\tikz[y=1.5cm, >=Stealth, arrows={[round]}, nodes={circle, draw}]
  \path   node[arr=  ]{Ti\emph kZ} % \arrow
   (0,-1) node[arr=* ]{Ti\emph kZ} % \arrow*
   (0,-2) node[arr=**]{Ti\emph kZ} % \arrow**
  ;
```

7 Calendar

Ti*k*Z Library `ext.calendar-plus`

```
\usetikzlibrary{ext.calendar-plus} % LaTeX and plain TeX
\usetikzlibrary[ext.calendar-plus] % ConTeXt
```

This library extends the Ti*k*Z library `calendar`.

Q & A: [**WeekNum-Q**, **CalCond-Q**, **CalMath-Q**] & [**WeekNum-A**, **CalCond-A**, **CalMath-A**]

7.1 Value-keys and nestable `if` key

`/tikz/day xshift` (initially `3ex`)
`/tikz/day yshift` (initially `3.5ex`)
`/tikz/month xshift` (initially `9ex`)
`/tikz/month yshift` (initially `9ex`)

The values of these 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.

`/tikz/if=(`⟨*conditions*⟩`)`⟨*code or options*⟩`else`⟨*else code or options*⟩ (no default)

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

7.2 PGFmath functions

`ext_weeksinmonthofyear(`*first weekday*`, `*month*`, `*year*`)`
`\pgfmathextweeksinmonthofyear{`*first weekday*`}{`*month*`}{`*year*`}`

Returns the number of (partial) weeks in the month *month* of year *year* when this month begins on a *first weekday*.

`ext_lastdayinmonthofyear(`*month*`, `*year*`)`
`\pgfmathextlastdayinmonthofyear{`*month*`}{`*year*`}`

Returns the last day (28, 29, 30 or 31) of month *month* of year *year*.

7.3 Week numbering (ISO 8601)

The actual week number algorithm is implemented by the `pgfcalendar-ext` package/module in section 28.2.

`/tikz/week code=`⟨*code*⟩ (no default)

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

`/tikz/week text=`⟨*text*⟩ (no default)

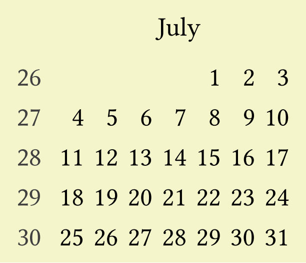
Works like `/tikz/day text` or `/tikz/month text`, only for weeks.

`/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.)



```
\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=\sffamily}];
```

8 Layers

TikZ Library ext.layers

```
\usetikzlibrary{ext.layers} % LaTeX and plain TeX
\usetikzlibrary[ext.layers] % ConTeXt
```

This library extends TikZ's functionalities to put nodes, edges, matrices and pics on a separate layer without having to use the `pgfonlayer` environment.

Consider this library experimental. If you can, avoid it and use the `pgfonlayer` environment or change the drawing order.

`/tikz/ext/layers/patch=node|matrix|pic|edge|all` (no default)

pre 0.6 `/tikz-ext/layers/patch`

Since this library is experimental, its functionality needs to be activated explicitly. Patches exist for

- `node`,
- `matrix`,
- `pic`[2],
- `edge` or
- `all` which applies all the patches at once.

`/tikz/ext/node on layer=`⟨*layer*⟩ (no default)

pre 0.6 `/tikz/node on layer`

If the `node` patch is applied, this key places a node on layer ⟨*layer*⟩.

`/tikz/ext/matrix on layer=`⟨*layer*⟩ (no default)

pre 0.6 `/tikz/matrix on layer`

If the `matrix` patch is applied, this key places the matrix on layer ⟨*layer*⟩.

`/tikz/ext/edge on layer=`⟨*layer*⟩ (no default)

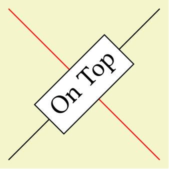
pre 0.6 `/tikz/edge on layer`

If the `edge` patch is applied, this key places the edge on layer ⟨*layer*⟩.

`/tikz/ext/pic on layer=`⟨*layer*⟩ (no default)

pre 0.6 `/tikz/pic on layer`

If the `pic` patch is applied, this key places the main code of a pic on layer ⟨*layer*⟩.



```
\usetikzlibrary {ext.layers}
\pgfdeclarelayer{front}
\begin{tikzpicture}[ext/layers/patch=node]
\pgfsetlayers{main,front}
\draw (0, -1) -- node[
                   ext/node on layer=front,
                   draw, fill=white, sloped
                 ] {On Top} (2,1);
\draw[red] (0, 1) -- (2, -1);
\end{tikzpicture}
```

[2] Only the normal `/tikz/pics/code` can be placed on different layers. Both `/tikz/pics/background code` and `/tikz/pics/foreground code` will not be affected.

9 Node Families

Ti*k*Z 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.

Q & A: [**NodeFam-Q**] & [**NodeFam-A**]

Before we get to the interesting keys, a common prefix can be set for the families' names. Initially this is `\pgfpictureid-` so that families of different pictures don't interact.

`/tikz/ext/node family/prefix`=⟨*prefix*⟩ (no default, initially `\pgfpictureid-`)

pre 0.6 `/tikz/node family/prefix`

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

9.1 Externalization

As this library usually needs multiple compilations to produce stable pictures it is incompatible with the `external` library.

However, the library provides support for the `memoize` [**memoize**] package.

9.2 Text Box

The following keys – when setup, see below – work with every shape with one single node part.[3] Initially though, only `circle` and `rectangle` are set up that way.

`/tikz/ext/node family/text height`=⟨*name*⟩ (no default, initially `{}`)

pre 0.6 `/tikz/node family/text height`

Nodes with the same ⟨*name*⟩ will have the same text height. An empty ⟨*name*⟩ disables the evaluation by the library.

`/tikz/ext/node family/text depth`=⟨*name*⟩ (no default, initially `{}`)

pre 0.6 `/tikz/node family/text depth`

Nodes with the same ⟨*name*⟩ will have the same text depth. An empty ⟨*name*⟩ disables the evaluation by the library.

`/tikz/ext/node family/text width`=⟨*name*⟩ (no default, initially `{}`)

pre 0.6 `/tikz/node family/text width`

Nodes with the same ⟨*name*⟩ will have the same text width. An empty ⟨*name*⟩ disables the evaluation by the library.

`/tikz/ext/node family/text`=⟨*name*⟩ (no default)

[3] Technically, it will also work with shapes with multiple node parts but it will only affect the main node part.

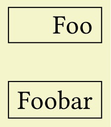
pre 0.6 /tikz/node family/text

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 where aligned to the left. With text width family align this can changed.

/tikz/ext/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, ext/node family={text width=manual, text width align=right}}]
\node (a) {Foo};
\node[below=of a] (b) {Foobar};
\end{tikzpicture}
```

/tikz/ext/node family/setup shape=⟨*shape*⟩ (no default)

pre 0.6 /tikz/node family/setup shape

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.

9.3 Minimum Width/Height

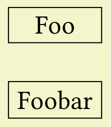
While the keys of the previous subsection work well enough for nodes of the same shape (and the same inner seps), for different node shapes the text box dimensions will be used differently for the node's total dimension.

For this, the following keys are necessary. When one of the keys are used the values of minimum width and/or minimum height are set to ext_nf_width or nf_height respectively.

/tikz/ext/node family/width=⟨*name*⟩ (no default, initially {})

pre 0.6 /tikz/node family/width

Nodes with the same ⟨*name*⟩ will have the same /pgf/minimum width . An empty ⟨*name*⟩ 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, ext/node family/width=manual}]
\node (a) {Foo};
\node[below=of a] (b) {Foobar};
\end{tikzpicture}
```

/tikz/ext/node family/height=⟨*name*⟩ (no default, initially {})

pre 0.6 `/tikz/node family/height`

Nodes with the same ⟨*name*⟩ will have the same `/pgf/minimum height` . An empty ⟨*name*⟩ disables the evaluation by the library.

`/tikz/ext/node family/size`=⟨*name*⟩ (no default)

pre 0.6 `/tikz/node family/size`

Sets both `height` and `width`.

9.4 More shapes that support the keys `width` and `height`

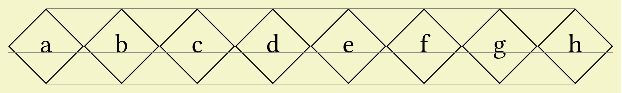
TikZ Library `ext.node-families.shapes.geometric`

```
\usetikzlibrary{ext.node-families.shapes.geometric} % LaTeX and plain TeX
\usetikzlibrary[ext.node-families.shapes.geometric] % ConTeXt
```

This library adds support for the keys `/tikz/ext/node family/width` and `/tikz/ext/node family/height` for the shapes of the PGF library `shapes.geometric`.

Q: [**NodeFam-Ellipse**]

The shapes are also setup for the keys from subsection 9.2.



```
\usetikzlibrary {ext.node-families.shapes.geometric}
\tikzexternaldisable % ext.node-families does not work with active externalization
\begin{tikzpicture}
\foreach \cnt[count=\Cnt] in {a,...,h}
  \node[draw, diamond, ext/node family/text=aTOh] (\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}
```

10 Nodes

Ti*k*Z Library `ext.nodes`

```
\usetikzlibrary{ext.nodes} % LaTeX and plain TeX
\usetikzlibrary[ext.nodes] % ConTeXt
```

This library extends Ti*k*Z's functionalities when it comes to nodes.

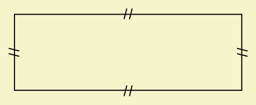
Q & A: [NodesOnLine-Q, NodesOnCurve-Q] & **[NodesOnLine-A, NodesOnCurve-A]**

10.1 Pic as a node

`/tikz/ext/pic=`⟨*boolean*⟩ (default `true, initially false`)

pre 0.6 `/tikz/pic`

This key allows one to use a pic where usually only nodes are accepted, for example as a label.



```
\usetikzlibrary {ext.nodes, ext.misc}
\begin{tikzpicture}[
  slsl/.pic={\draw(-2pt, 1.5pt)--( 2pt, .5pt)
                  ( 2pt,-1.5pt)--(-2pt,-.5pt);}]
\node[
  draw, minimum width=3cm, minimum height=1cm,
  label={[ext/pic            ] east:slsl},
  label={[ext/pic, rotate= 90]north:slsl},
  label={[ext/pic            ] west:slsl},
  label={[ext/pic, rotate=-90]south:slsl}]{};
\end{tikzpicture}
```

10.2 Nodes on paths

When nodes are placed along paths they don't interrupt the path at that place. The decoration `markings` and its `/pgf/decoration/mark connection node` key can help but only works for straight paths and doesn't play nicely with arrow tips.

This library provides alternatives. These are separated into straight paths, i. e. `--`, and everything else (including any `to path`).

10.2.1 Nodes on Lines

`/tikz/ext/node on line=`⟨*anchor specification*⟩ (style, default {})

pre 0.6 `/tikz/node on line`

This installs a `/tikz/to path` that places *one* node along a straight line but connect the line with it.

This allows a node to be placed *on* a straight line without having to use `fill = white` or similar tricks to make the line disappear beneath the node.

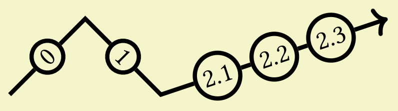
The optional ⟨*anchor specification*⟩ allows to specify the anchors to which the line should connect. It allows one or two anchors divided by `and` to be specified.

`/tikz/ext/nodes on line` (style, no value)

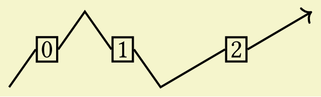
pre 0.6 `/tikz/nodes on line`

This is similar to the previous key but allows multiple nodes to be placed on a straight line *if* they are in the correct order (from start to target), don't overlap with each other, the start or the target.

It allows *no* anchor specification.



```
\usetikzlibrary {ext.nodes, quotes}
\tikz[inner sep=.15em, circle, nodes=draw, sloped]
  \draw[ultra thick, ->, ext/node on line] (0,0) to["0"] (1,1)
                                                 to["1"] (2,0)
    to[ext/nodes on line, "2.1" near start, "2.2", "2.3" near end] (5,1);
```



```
\usetikzlibrary {ext.nodes, quotes}
\tikz[inner sep=.15em, nodes=draw]
  \draw[thick, ->, ext/node on line=west and east]
     (0,0) to["0"] (1,1)
           to["1"] (2,0)
           to["2"] (4,1);
```

10.2.2 Nodes on Curves

The following keys need the `intersections` and the `spath3` [**spath3**] library to be loaded. They will not be automatically loaded by this library.

Any `/pgf/outer sep` will be ignored.

If you can, use `fill=`⟨*bg color*⟩ instead of these keys, it will be much faster and easier.

`/tikz/ext/nodes on curve=`⟨*to path*⟩ (style, default `line to`)

pre 0.6 `/tikz/nodes on curve`

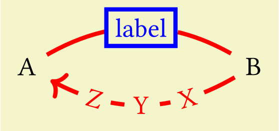
Similar to `nodes on line`, this key allows to have nodes on arbitrary paths.

This is not suitable for paths connecting nodes.

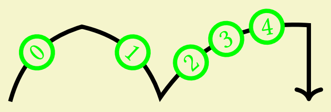
`/tikz/ext/nodes on curve'=`⟨*to path*⟩ (style, default `line to`)

pre 0.6 `/tikz/nodes on curve'`

As above but suitable for connecting nodes.



```
\usetikzlibrary {ext.nodes, intersections, quotes, spath3}
\begin{tikzpicture}[ultra thick]
  \node (A) at (0, 0) {A} ;
  \node (B) at (3, 0) {B} ;
  \draw [red, ->, ext/nodes on curve'=bend left]
    (A) to node[blue,draw]{label} (B)
        to ["X" {sloped, near start},
            "Z" {sloped, near end},
            "Y"] (A);
\end{tikzpicture}
```



```
\usetikzlibrary {ext.nodes, intersections, quotes, spath3}
\tikz[inner sep=.15em, circle, nodes={draw, green}, sloped, ultra thick]
  \draw[->, ext/nodes on curve=bend left] (0,0) to["0"] (1,1)
                                                 to["1"] (2,0)
                    to["2" near start, "3", "4" near end] (4,1)
                                                 -- ++(down:1);
```

10.3 Automatic placement of nodes

The `/tikz/auto` key allows automatic placement of nodes along a path segment. This library extends this in various ways.

10.3.1 More than left and right

Besides `left` and `right` that are provided by Ti*k*Z the following placement mechanism are provided:

- `ext/left` will place a node to the left of the direction of the line,
- `ext/right` will place a node to the right of the direction of the line,
- `ext/above` will place a node towards the direction of the line,
- `ext/below` will place a node against the direction of the line,
- `ext/west` will place a node towards the left side of the paper,
- `ext/east` will place a node towards the right side of the paper,
- `ext/north` will place a node towards the upper side of the paper and
- `ext/south` will place a node twoards the lower side of the paper.

The placement mechanisms `ext/left` and `ext/right` are like the original `left` and `right` mechanisms but don't swap sides when `/tikz/sloped` is used.

Certain cases exist for `ext/west`, `ext/east`, `ext/north` and `ext/south` placements where it is not clear how a node should be placed. These cases and their behavior can be seen in figure 1.

10.3.2 Offset

Nodes are usually placed with their border (including any `outer sep`) on the line. With the following option, a node will be shifted a certain offset distance.

`/tikz/ext/auto with offset=`⟨*true or false*⟩ (default `true`)

This key activates the offset function.

`/tikz/ext/auto offset` (initially `1cm`)

The offset distance itself.

For the `brace` decoration, the following keys are provided which needs the `decorations.pathreplacing` loaded before they can be used.

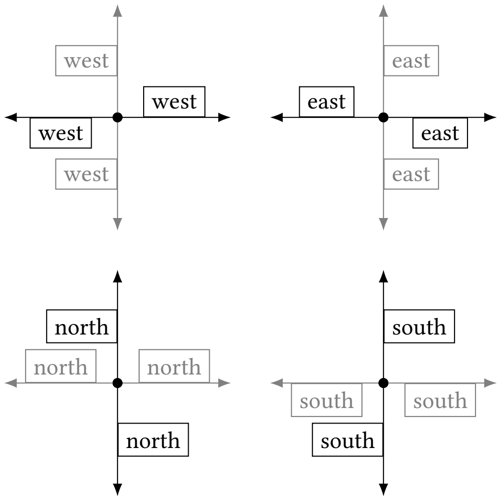


Figure 1: Behavior of `ext/est`, `ext/east`, `ext/north` and `ext/south` in certain cases

`/tikz/ext/nodes/install auto offset for brace decoration=`⟨*distance*⟩ (default `0pt`)

This key installs the necessary customizations for the `/pgf/decoration/raise` key so that the given value is available as an offset.

It also makes available the following keys.

`/tikz/ext/auto offset for brace decoration` (style, no value)

This sets `/tikz/ext/auto offset` to `\pgfdecorationsegmentamplitude+(\pgfkeysvalueof/pgf/decoration/raise)`.

`/tikz/ext/every brace node` (style, no value)

Using this key on a node along a path that's decorated by the `brace` decoration will offset the node so that it will be placed at the tip of the brace.

10.3.3 Precise placement

The default behavior of the `auto` placement mechanism is to snap to one of the eight compass directions.

`/tikz/ext/precise auto angle=`⟨*true or false*⟩ (default `true`)

With this option set to `true`, the `auto` placement won't snap to one of the eight compass directions.

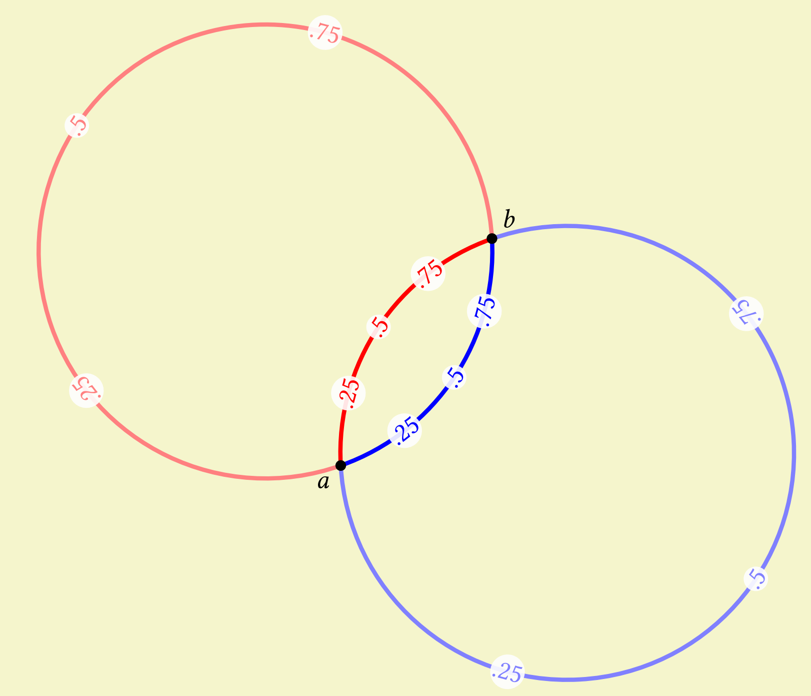
This key disables the `/tikz/sloped` option which in turn will disable this option.

11 Arc *to* a point

Ti*k*Z Library `ext.paths.arcto`

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

This library adds the new path operation `arc 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/ext/arc to/clockwise` (style, no value)

This constructs an arc that goes clockwise.

`/tikz/ext/arc to/counter clockwise` (style, no value)

This constructs an arc that goes counter clockwise.

This is the default.

`/tikz/ext/arc to/large` (style, no value)

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

`/tikz/ext/arc to/small` (style, no value)

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

`/tikz/ext/arc to/rotate=`⟨*degree*⟩ (no default)

Rotates the arc by ⟨*degree*⟩. This is only noticeable when `x radius` and `y radius` are different.

`/tikz/ext/arc to/x radius=`⟨*value*⟩ (no default)

This forwards the ⟨*value*⟩ to `/tikz/x radius` . Its ⟨*value*⟩ is used for the radius of the arc.

`/tikz/ext/arc to/y radius=`⟨*value*⟩ (no default)

This forwards the ⟨*value*⟩ to `/tikz/y radius` . Its ⟨*value*⟩ is used for the radius of the arc.

`/tikz/ext/arc to/radius=`⟨*value*⟩ (no default)

This forwards the ⟨*value*⟩ to both `/tikz/x radius` and `/tikz/y radius`. Its ⟨*value*⟩ is used for radius of the arc.

`/tikz/ext/every arc to` (style, no value)

After `/tikz/every arc` this will also be applied before any ⟨*options*⟩ are set.

It should be noted that this uses `\pgfpatharcto` for which the Ti*k*Z manual warns:

> *The internal computations necessary for this command are numerically very unstable. In particular, the arc will not always really end at the* ⟨*target coordinate*⟩*, 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.

12 More Horizontal and Vertical Lines

Ti*k*Z Library `ext.paths.ortho`

```
\usetikzlibrary{ext.paths.ortho} % LaTeX and plain TeX
\usetikzlibrary[ext.paths.ortho] % ConTeXt
```

This library adds new path specifications `|-|`, `-|-` as well as `r-ud`, `r-du`, `r-lr` and `r-rl`.

12.1 Zig-Zag

Similar to the path operations `|-` and `-|` this library adds the path operations `|-|` and `-|-`.

`\path` … `|-|[`⟨*options*⟩`]`⟨*coordinate or cycle*⟩ …;

This operation means “first vertical, then horizontal and then vertical again”.

`\path` … `-|-[`⟨*options*⟩`]`⟨*coordinate or cycle*⟩ …;

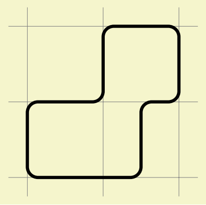
This operation means “first horizontal, then vertical and then horizontal again”.

`/tikz/ext/ortho/ratio=`⟨*ratio*⟩ (no default, initially `0.5`)

pre 0.6 `/tikz/ortho/ratio`

This sets the ratio for the middle part of the Zig-Zag connection.

For values ⟨*ratio*⟩ < 0 and ⟨*ratio*⟩ > 1 the Zig-Zag lines will look more like Zig-Zig lines.



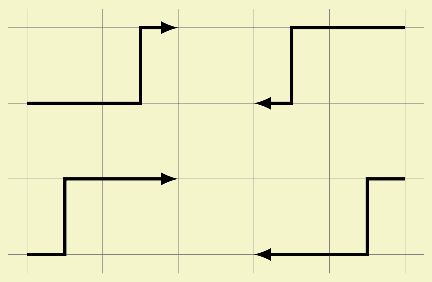
```
\usetikzlibrary {ext.paths.ortho}
\begin{tikzpicture}[very thick, rounded corners]
\draw[help lines] (-.25, -1.25) grid (2.25, 1.25);
\draw (0, 0) -|-            (2, 1) --
      (2, 0) -|-[ratio=.25] (0,-1) -- cycle;
\end{tikzpicture}
```

`/tikz/ext/ortho/distance=`⟨*distance*⟩ (no default)

pre 0.6 `/tikz/ortho/distance`

This sets the distance between the start point and the middle part of the Zig-Zag connection.

For values ⟨*distance*⟩ < 0 the distance will be used for the target coordinate.



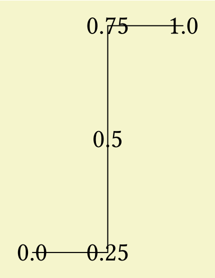
```
\usetikzlibrary {ext.paths.ortho}
\begin{tikzpicture}[very thick,-latex]
\draw[help lines,-] (-.25, -.25) grid (5.25, 3.25);
\draw (0, 0) -|-[distance= .5cm] ++(2, 1);
\draw (0, 2) -|-[distance=-.5cm] ++(2, 1);

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

`/tikz/ext/ortho/from center`=⟨*true or false*⟩ (default `true`)

pre 0.6 `/tikz/ortho/from center`

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 {ext.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/ext/ortho/spacing`=⟨*number*⟩ (no default, initially 4)

pre 0.6 `/tikz/ortho/spacing`

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 ⟨*number*⟩ = 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/ext/ortho/middle 0 to 1` (no value)

pre 0.6 `/tikz/ortho/middle 0 to 1`

This is an alias for `spacing = 0`.

12.2 Zig-Zig

`\path` … `r-ud[`⟨*options*⟩`]`⟨*coordinate or cycle*⟩ …`;`

This operation means "first up, then horizontal and then down".

`/tikz/ext/ortho/ud distance=`⟨*length*⟩ (no default, initially `.5cm`)

pre 0.6 `/tikz/ortho/ud distance`

This sets the distance between the start and the horizontal line to ⟨*length*⟩.

`\path` … `r-du[`⟨*options*⟩`]`⟨*coordinate or cycle*⟩ …`;`

This operation means "first down, then horizontal and then up".

`/tikz/ext/ortho/du distance=`⟨*length*⟩ (no default, initially `.5cm`)

pre 0.6 `/tikz/ortho/du distance`

This sets the distance between the start and the horizontal line to ⟨*length*⟩.

`\path` … `r-lr[`⟨*options*⟩`]`⟨*coordinate or cycle*⟩ …`;`

This operation means "left down, then vertical and then right".

`/tikz/ext/ortho/lr distance=`⟨*length*⟩ (no default, initially `.5cm`)

pre 0.6 `/tikz/ortho/lr distance`

This sets the distance between the start and the vertical line to ⟨*length*⟩.

`\path` … `r-rl[`⟨*options*⟩`]`⟨*coordinate or cycle*⟩ …`;`

This operation means "first right, then vertical and then down".

`/tikz/ext/ortho/rl distance`=⟨*length*⟩ (no default, initially `.5cm`)

pre 0.6 `/tikz/ortho/rl distance`

This sets the distance between the start and the vertical line to ⟨*length*⟩.

All distances can be set with one key.

`/tikz/ext/ortho/udlr distance`=⟨*length*⟩ (no default)

pre 0.6 `/tikz/ortho/udlr distance`

Sets all the previous distances to the same value ⟨*length*⟩.

12.3 Even more Horizontal and Vertical Lines

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

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

pre 0.6 `/tikz/horizontal vertical`

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

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

pre 0.6 `/tikz/vertical horizontal`

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

`/tikz/ext/horizontal vertical horizontal`=⟨*options*⟩ (style, no default)

pre 0.6 `/tikz/horizontal vertical horizontal`

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

`/tikz/ext/vertical horizontal vertical`=⟨*options*⟩ (style, no default)

pre 0.6 `/tikz/vertical horizontal vertical`

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

`/tikz/ext/up horizontal down`=⟨*options*⟩ (style, no default)

pre 0.6 `/tikz/up horizontal down`

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

`/tikz/ext/down horizontal up`=⟨*options*⟩ (style, no default)

pre 0.6 `/tikz/down horizontal up`

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

`/tikz/ext/left vertical right`=⟨*options*⟩ (style, no default) pre 0.6 `/tikz/left vertical right`

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

`/tikz/ext/right vertical left=`⟨*options*⟩ (style, no default)

pre 0.6 `/tikz/right vertical left`

This installs `to path = r-rl (\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/ext/only vertical second=`⟨*length*⟩ (style, default `0pt`)

pre 0.6 `/tikz/only vertical second`

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/ext/only horizontal second=`⟨*length*⟩ (style, default `0pt`)

pre 0.6 `/tikz/only horizontal second`

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/ext/only vertical first=`⟨*length*⟩ (style, default `0pt`)

pre 0.6 `/tikz/only vertical first`

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/ext/only horizontal first=`⟨*length*⟩ (style, default `0pt`)

pre 0.6 `/tikz/only horizontal first`

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.

Since all previous key are rather cumbersome, one can install shortcuts for these.

`/tikz/ext/ortho/install shortcuts` (style, no value)

pre 0.6 `/tikz/ortho/install shortcuts`

Installs the following shortcuts:

```
|-    ⟶  vertical horizontal
-|    ⟶  horizontal vertical
-|-   ⟶  horizontal vertical horizontal
|-|   ⟶  vertical horizontal vertical

|*    ⟶  only vertical first
*|    ⟶  only vertical second
-*    ⟶  only horizontal first
*-    ⟶  only horizontal second

r-ud  ⟶  up horizontal down
r-du  ⟶  down horizontal up
r-lr  ⟶  left vertical right
r-rl  ⟶  right vertical left
```

13 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`.

Q & A: [**TimerRect-Q**, **TimerPara-Q**] & [**TimerRect-A**, **TimerPara-A**]

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.

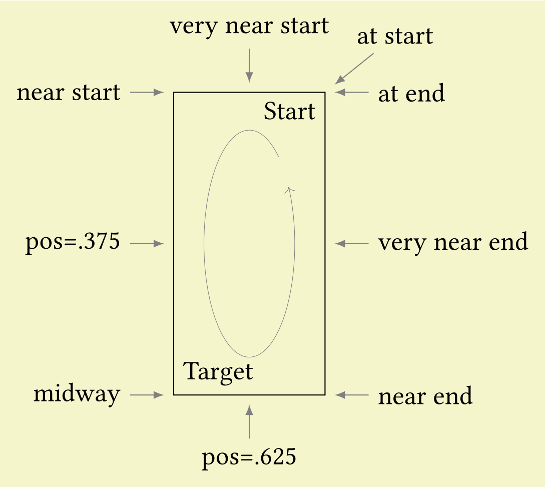
13.1 Rectangle

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.

`/tikz/ext/rectangle timer=line` or `rectangle` (no default)

pre 0.6 `/tikz/rectangle timer`

By default, the library activates the new (correct) timer for `rectangle`. With `rectangle timer = line` the original line timer can be reinstated.

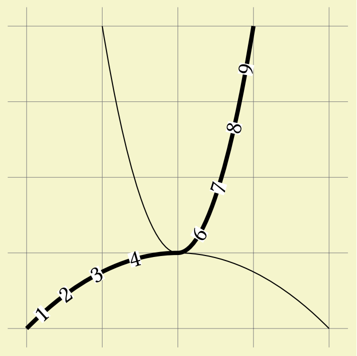


```
\usetikzlibrary {ext.paths.timer}
\begin{tikzpicture}[scale=2, every pin edge/.style={latex-, gray}]
\coordinate [label=above right:Target] (A) at (0,0);
\coordinate [label=below left:Start]   (B) at (1,2);
\draw[->, help lines] ([shift=(50:.3 and .75)] .5,1)
  arc[start angle=50, delta angle=340, 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}
```

13.2 Parabola

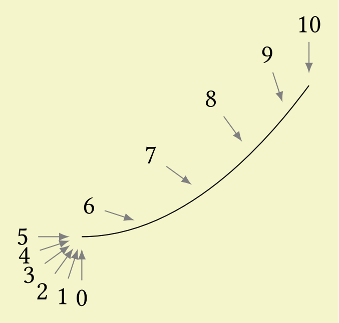
For the `parabola` path operator the timer is similar to the `.. controls ..` operator.

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}
```

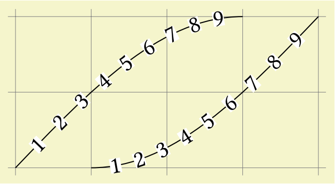
If no `bend` is specified half the positions will collapse into one end of the curve.



```
\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}
```

13.3 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}
```

14 Using Images as a Pattern

Ti*k*Z 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.

Q & A: [Pattern-Q] & **[Pattern-A]**

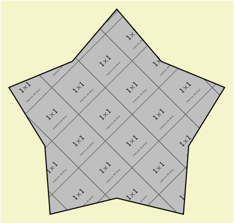
With this library arbitrary images (or indeed PDF documents) can be used as a repeating pattern for the background of a path. This is a two-step process:

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`.

15 Positioning Plus

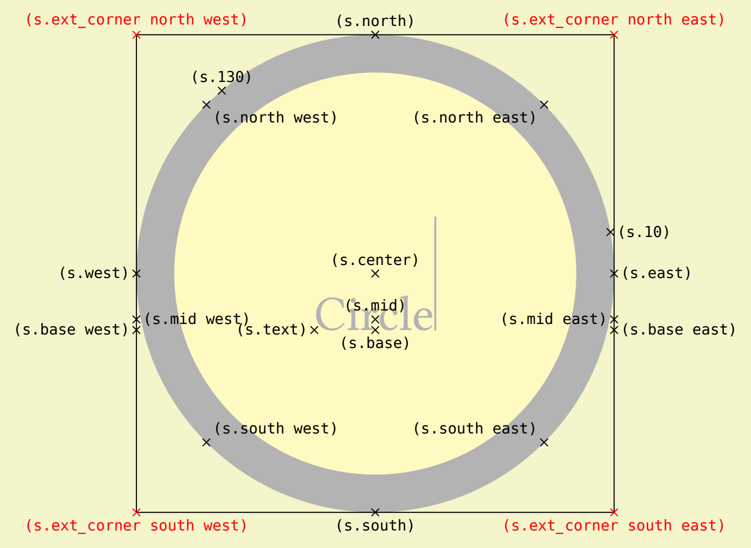
Ti*k*Z Library ext.positioning-plus

```
\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.

15.1 Useful corner anchors

The anchors ext_corner north east, ext_corner north west, ext_corner south west and ext_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.



```
\usetikzlibrary {ext.positioning-plus}
\Huge
\begin{tikzpicture}
\node[name=s,shape=circle,shape example,scale=.75,outer sep=auto]
  {Circle\vrule width 1pt height 2cm};
\foreach \anchor/\placement in {
  north west/below right, north/above, north east/below left,
  west/left, center/above, east/right,
  mid west/right, mid/above, mid east/left,
  base west/left, base/below, base east/right,
  south west/above right, south/below, south east/above left,
  text/left, 10/right, 130/above}
  \draw[node font=\scriptsize, shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
    node[\placement] {\texttt{(s.\anchor)}};
\draw (s.ext_corner north west) rectangle (s.ext_corner south east);
\foreach \anchor/\placement in {
  corner north west/above, corner north east/above,
  corner south west/below, corner south east/below}
  \draw[node font=\scriptsize, red, shift=(s.ext_\anchor)] plot[mark=x] coordinates{(0,0)}
    node[\placement] {\texttt{(s.ext\textunderscore\anchor)}};
\end{tikzpicture}
```

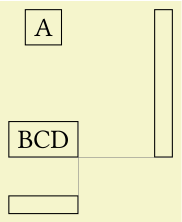
15.2 Useful placement keys for vertical and horizontal alignment

/tikz/ext/left=⟨*specification*⟩ (default 0pt)
/tikz/ext/right=⟨*specification*⟩ (default 0pt)
/tikz/ext/above=⟨*specification*⟩ (default 0pt)
/tikz/ext/below=⟨*specification*⟩ (default 0pt)
/tikz/ext/above left=⟨*specification*⟩ (default 0pt)
/tikz/ext/below left=⟨*specification*⟩ (default 0pt)
/tikz/ext/above right=⟨*specification*⟩ (default 0pt)
/tikz/ext/below right=⟨*specification*⟩ (default 0pt)
/tikz/ext/mid left=⟨*specification*⟩ (default 0pt)
/tikz/ext/mid right=⟨*specification*⟩ (default 0pt)
/tikz/ext/base left=⟨*specification*⟩ (default 0pt)
/tikz/ext/base right=⟨*specification*⟩ (default 0pt)

While the ⟨*specification*⟩ of all these keys still accept the same form as with Ti*k*Z, 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[ext/right=   of |(A)(BCD)] (c) {};
\node[ext/below=.5:of -(A)(BCD)] (d) {};
\draw[help lines] (BCD.south west) -- (c.south east)
                  (BCD.north east) -- (d.south east);
\end{tikzpicture}
```

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.[4]

The `fitting` functionality is also available without the placement.

/tikz/ext/fit bounding box=⟨*list of coordinates*⟩ (no default)
/tikz/ext/span vertical=⟨*list of coordinates*⟩ (no default)
/tikz/ext/span horizontal=⟨*list of coordinates*⟩ (no default)
/tikz/ext/span=⟨*list of coordinates*⟩ (no default)

These all create a rectangular node with the name `ext_fit bounding box` that encompasses the ⟨*list of coordinates*⟩.

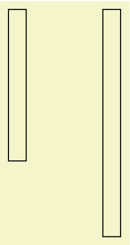
The `span vertical` key will also set `/pgf/minimum height` to the height of this bounding box

The `span horizontal` key will also set `/pgf/minimum width` to the width of this bounding box

The last one combines `span vertical` and `span horizontal`.

/tikz/ext/north left=⟨*specification*⟩ (default 0pt)
/tikz/ext/south left=⟨*specification*⟩ (default 0pt)
/tikz/ext/north right=⟨*specification*⟩ (default 0pt)
/tikz/ext/south right=⟨*specification*⟩ (default 0pt)
/tikz/ext/west above=⟨*specification*⟩ (default 0pt)
/tikz/ext/west below=⟨*specification*⟩ (default 0pt)
/tikz/ext/east above=⟨*specification*⟩ (default 0pt)
/tikz/ext/east below=⟨*specification*⟩ (default 0pt)

These work similarly to `left`, `right`, `above` and `below` but they are north- or south-aligned.



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

[4] This is probably more useful when `/tikz/on grid` is used.

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

`/tikz/ext/corner above left=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner below left=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner above right=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner below right=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner north left=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner south left=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner north right=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner south right=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner west above=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner west below=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner east above=`⟨*specification*⟩ (default `0pt`)
`/tikz/ext/corner east below=`⟨*specification*⟩ (default `0pt`)

These work the same as `above left`, `below left`, … but they use the new generic corner anchors

16 Scaling Pictures to a Specific Size

Ti*k*Z Library `ext.scalepicture`

```
\usetikzlibrary{ext.scalepicture} % LaTeX and plain TeX
\usetikzlibrary[ext.scalepicture] % ConTeXt
```

This library scales Ti*k*Z pictures to a specific width or height by scaling the whole picture.

If one of the keys below are used on a Ti*k*Z picture, meaning as an option to `\tikzpicture` or `\begin{tikzpicture}`, the size of the picture[5] will be measured and written to the AUX file so that it will be available at the next compilation run and an appropriate scaling for the picture can be installed.

`\tikzextpicturewidth`

Returns the last measured width of the picture.

This will expand to `0pt` if the picture hasn't been measured before.

`\tikzextpictureheight`

Returns the last measured height of the picture.

This will expand to `0pt` if the picture hasn't been measured before.

`/tikz/ext/save picture size` (style, no value)

pre 0.6 `/tikz/save picture size`

This key is usually used by the keys provided by this library. Normally, this is not needed to be explicitly given.

16.1 Externalization

As this library usually needs multiple compilations to produce stable pictures it is incompatible with the `external` library.

However, the library provides support for the `memoize` [**memoize**] package. When it is used the arguments to the keys below will be saved as the context of the memo. This means that the arguments need to be a valid `\dimexpr` expression.

16.2 Keeping the aspect ratio

The following *unstarred* keys do not change the aspect ratio of the picture.

`/tikz/ext/picture width=`⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/picture width`

Scales the picture so that the width of the picture will be ⟨*dimension*⟩. This will keep the aspect ratio the same.

`/tikz/ext/minimum picture width=`⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/minimum picture width`

As above but will not change the size of the picture if its width is greater than ⟨*dimension*⟩.

`/tikz/ext/maximum picture width=`⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/maximum picture width`

As above but will not change the size of the picture if its width is less than ⟨*dimension*⟩.

`/tikz/ext/picture height=`⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/picture height`

Scales the picture so that the height of the picture will be ⟨*dimension*⟩. This will keep the aspect ratio the same.

`/tikz/ext/minimum picture height=`⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/minimum picture height`

As above but will not change the size of the picture if its height is greater than ⟨*dimension*⟩.

`/tikz/ext/maximum picture height=`⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/maximum picture height`

As above but will not change the size of the picture if its height is less than ⟨*dimension*⟩.

`/tikz/ext/minimum picture size={`⟨*width*⟩`}{`⟨*height*⟩`}` (no default)

[5] This is the size of the pseudo-node `current bounding box`.

pre 0.6 `/tikz/minimum picture size`

Scales the picture so that its height will be at least ⟨*width*⟩ and its height will be at least ⟨*height*⟩.

`/tikz/ext/maximum picture size`={⟨*width*⟩}{⟨*height*⟩} (no default)

pre 0.6 `/tikz/maximum picture size`

Scales the picture so that its height will be at most ⟨*width*⟩ and its height will be at most ⟨*height*⟩.

16.3 Changing the aspect ratio

The following *starred* keys do change the aspect ratio.

`/tikz/ext/picture width*`=⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/picture width*`

Scales the picture so that the width of the picture will be ⟨*dimension*⟩. This will only scale the x axis.

`/tikz/ext/minimum picture width*`=⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/minimum picture width*`

As above but will not change the size of the picture if its width is greater than ⟨*dimension*⟩.

`/tikz/ext/maximum picture width*`=⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/maximum picture width*`

As above but will not change the size of the picture if its width is less than ⟨*dimension*⟩.

`/tikz/ext/picture height*`=⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/picture height*`

Scales the picture so that the height of the picture will be ⟨*dimension*⟩. This will only scale the y axis.

`/tikz/ext/minimum picture height*`=⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/minimum picture height*`

As above but will not change the size of the picture if its height is greater than ⟨*dimension*⟩.

`/tikz/ext/maximum picture height*`=⟨*dimension*⟩ (no default)

pre 0.6 `/tikz/maximum picture height*`

As above but will not change the size of the picture if its height is less than ⟨*dimension*⟩.

`/tikz/ext/picture size*`={⟨*width*⟩}{⟨*height*⟩} (no default)

pre 0.6 `/tikz/picture size*`

Scales the picture so that its width will be ⟨*width*⟩ and its height will be ⟨*height*⟩.

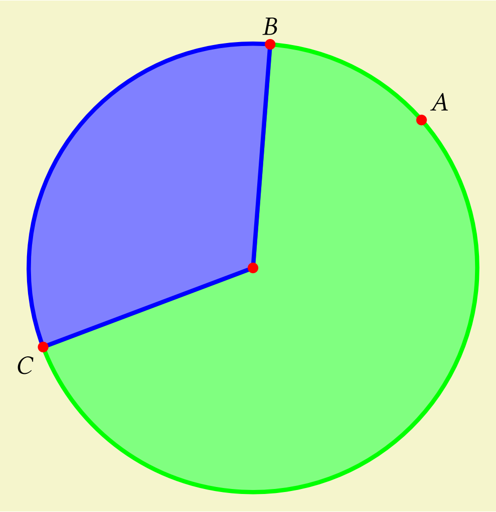
This will scale both axes but independent from each other.

17 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[ext/arc through={clockwise, (A)}] (C)
  -- (arc through center) -- cycle;
\draw[ultra thick, draw=blue, fill=blue!50]
  (B) to[ext/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 by used for circles in the `canvas` coordinate system.

`/tikz/ext/arc through/through`=⟨*coordinate*⟩ (no default, initially `(0,0)`)

pre 0.6 `/tikz/arc through/through`

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

`/tikz/ext/arc through/center suffix`=⟨*suffix*⟩ (no default, initially)

pre 0.6 `/tikz/arc trough/center suffix`

The `arc through` will define a coordinate named `arc through center`⟨*suffix*⟩ so that it can be referenced later.

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

pre 0.6 `/tikz/arc trough/clockwise`

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

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

pre 0.6 `/tikz/arc trough/counter clockwise`

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/ext/arc through`=⟨*key-value*⟩ (no default)

pre 0.6 `/tikz/arc trough/arc through`

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.

18 Autobending

Ti*k*Z Library `ext.topaths.autobend`

```
\usetikzlibrary{ext.topaths.autobend} % LaTeX and plain TeX
\usetikzlibrary[ext.topaths.autobend] % ConTeXt
```

This library provides various bended `to` paths that bend in the specified direction.

Q & A: [**AutoBend-Q**] & [**AutoBend-A**]

The keys `/tikz/bend left` and `/tikz/bend left` from Ti*k*Z bend the requested curve in relation of the connecting coordinates/nodes.

The keys provided by this library bend the curve in the direction relative to the paper (north, south, west and east) or relative to the current coordinate system (up, down, left and right).

`/tikz/ext/autobend north=`⟨*angle*⟩ (default last value)

Works like the `bend left` and `bend right` options but bends the curve to the top of the page (i. e. it ignores the current transformation).

`/tikz/ext/autobend south=`⟨*angle*⟩ (default last value)

Works like `autobend north` but bends the curve to the bottom of the page.

`/tikz/ext/autobend west=`⟨*angle*⟩ (default last value)

Works like `autobend north` but bends the curve to the left of the page.

`/tikz/ext/autobend east=`⟨*angle*⟩ (default last value)

Works like `autobend north` but bends the curve to the right of the page.

`/tikz/ext/autobend up=`⟨*angle*⟩ (default last value)

Works like the `bend left` and `bend right` options but bends the curve upwards (i. e. it observes the current transformation).

`/tikz/ext/autobend down=`⟨*angle*⟩ (default last value)

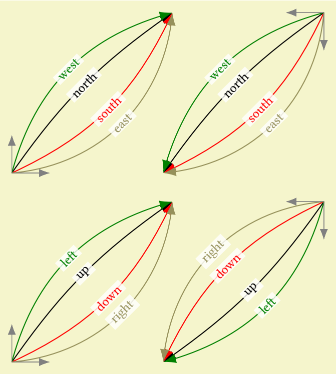
Works like `autobend up` but bends the curve downwards.

`/tikz/ext/autobend left=`⟨*angle*⟩ (default last value)

Works like `autobend up` but bends the curve leftwards.

`/tikz/ext/autobend right=`⟨*angle*⟩ (default last value)

Works like `autobend up` but bends the curve rightwards.



```
\usetikzlibrary {arrows.meta, ext.topaths.autobend}
\begin{tikzpicture}[
  every path/.append style=-Latex,
  pics/cs/.style={
    /tikz/transform shape,
    code={\draw[help lines, Latex-Latex] (up:1) |- (right:1);}
  },
  nodes={sloped, fill=white, inner ysep=+.1em, fill opacity=.8, text opacity=1, scale=.5}]
\foreach[count=\i] \c/\d in {black/north, red/south,
                             green!50!black/west, yellow!50!black/east}
  \draw[\c] (0,0) pic {cs} to[ext/autobend \d=\i0] node{\d} +(45:3);
\foreach[count=\i] \c/\d in {black/north, red/south,
                             green!50!black/west, yellow!50!black/east}
  \draw[shift=(right:2), rotate=180, \c]
          (45:-3) pic {cs} to[ext/autobend \d=\i0] node{\d} (0,0);

\tikzset{shift=(down:2.5)}
\foreach[count=\i] \c/\d in {black/up, red/down,
                             green!50!black/left, yellow!50!black/right}
  \draw[\c] (0,0) pic {cs} to[ext/autobend \d=\i0] node{\d} +(45:3);
\foreach[count=\i] \c/\d in {black/up, red/down,
                             green!50!black/left, yellow!50!black/right}
  \draw[shift=(right:2), rotate=180, \c]
          (45:-3) pic {cs} to[ext/autobend \d=\i0] node{\d} (0,0);
\end{tikzpicture}
```

19 Mirror, Mirror on the Wall

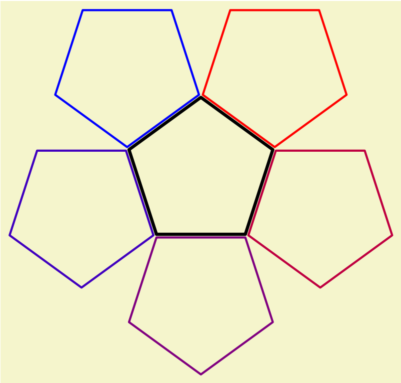
TikZ Library `ext.transformations.mirror`

```
\usetikzlibrary{ext.transformations.mirror} % LaTeX and plain TeX
\usetikzlibrary[ext.transformations.mirror] % ConTeXt
```

This library adds more transformations to TikZ.

As explained in section 21, 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.

19.1 Using the reflection matrix

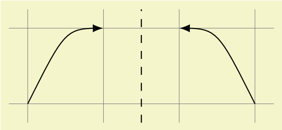


```
\usetikzlibrary {shapes.geometric,ext.transformations.mirror}
\begin{tikzpicture}[line join=round, thick, reg poly/.style={
  shape=regular polygon, regular polygon sides={#1}}]
\node[reg poly=5, minimum size=+2cm, draw, very thick] (a) {};
\foreach \i[evaluate={\col=(\i-1)/.04}] in {1,...,5}
  \node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

`/tikz/ext/xmirror=`⟨*value or coordinate*⟩ (default `0pt`)

pre 0.6 `/tikz/xmirror`

Sets up a transformation that mirrors along a horizontal line that goes through point (⟨*value*⟩, 0) or ⟨*coordinate*⟩.



```
\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[ext/xmirror=(m),-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

/tikz/ext/ymirror=⟨*value or coordinate*⟩ (default 0pt)

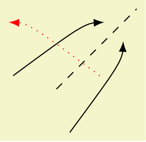
pre 0.6 /tikz/ymirror

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

/tikz/ext/mirror x=⟨*coordinate*⟩ (default (0,0))

pre 0.6 /tikz/mirror x

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

/tikz/ext/mirror y=⟨*coordinate*⟩ (default (0,0))

pre 0.6 /tikz/mirror y

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

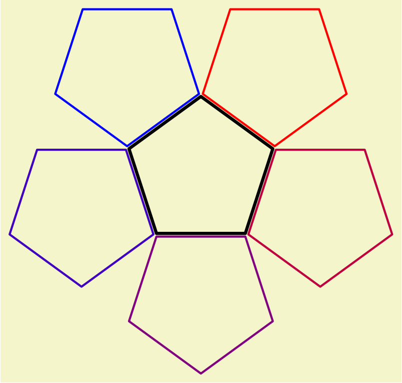
/tikz/ext/mirror=⟨*point A*⟩--⟨*point B*⟩ (no default)

pre 0.6 /tikz/mirror

Sets up a transformation that mirrors along a line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.

When only ⟨*point A*⟩ is given that line goes through ⟨*point A*⟩ and the origin.

19.2 Using built-in transformations

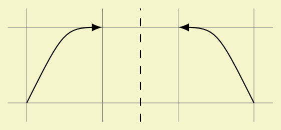


```
\usetikzlibrary {shapes.geometric,ext.transformations.mirror}
\begin{tikzpicture}[line join=round, thick, reg poly/.style={
  shape=regular polygon, regular polygon sides={#1}}]
\node[reg poly=5, minimum size=+2cm, draw, very thick] (a) {};
\foreach \i[evaluate={\col=(\i-1)/.04}] in {1,...,5}
  \node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

`/tikz/ext/xMirror=`⟨*value or coordinate*⟩ (default `0pt`)

pre 0.6 `/tikz/xMirror`

Sets up a transformation that mirrors along a horizontal line that goes through point (⟨*value*⟩, 0) or ⟨*coordinate*⟩.



```
\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[ext/xMirror=(m),-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`/tikz/ext/yMirror=`⟨*value or coordinate*⟩ (default `0pt`)

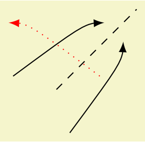
pre 0.6 `/tikz/yMirror`

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

`/tikz/ext/Mirror x=`⟨*coordinate*⟩ (default `(0,0)`)

pre 0.6 `/tikz/Mirror x`

Similar to `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[ ext/xMirror=(m), -latex, red, dotted] (0,0) .. controls (.5,1) .. (1,1);
\draw[ext/Mirror x=(m), -latex]              (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`/tikz/ext/Mirror y=`⟨*coordinate*⟩ (default `(0,0)`)

pre 0.6 `/tikz/Mirror y`

Similar to `yMirror`, this however uses the `xyz` coordinate system instead of the `canvas` system.

`/tikz/ext/Mirror=`⟨*point A*⟩`--`⟨*point B*⟩ (no default)

pre 0.6 `/tikz/Mirror`

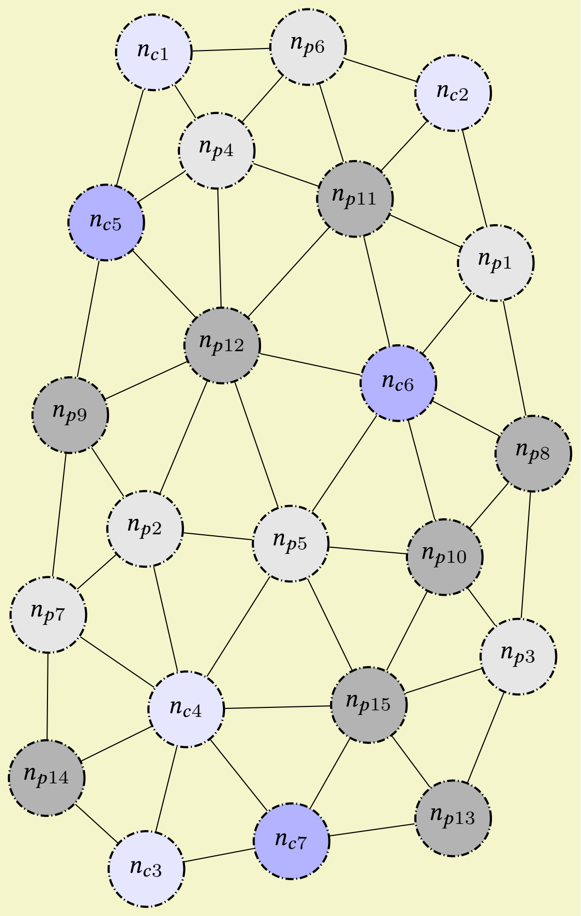
Sets up a transformation that mirrors along a line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.

When only ⟨*point A*⟩ is given that line goes through ⟨*point A*⟩ and the origin.

Part III

PGF Libraries

These libraries (should) work with both PGF and Ti*k*Z.



```
\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/ext/if=c#1
      {/utils/ext/ifnum={#2<5}{bluelight}{bluedark}}
      {/utils/ext/ifnum={#2<8}{light}{dark}}},
  light/.style={fill=gray!20},  bluelight/.style={fill=blue!10},
  dark/.style ={fill=gray!60},  bluedark/.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};
};
```

20 Arrow Tips

Ti*k*Z Library `ext.arrows`

```
\usepgflibrary{ext.arrows}  % LaTeX and plain TeX and pure pgf
\usepgflibrary[ext.arrows]  % ConTeXt and pure pgf
\usetikzlibrary{ext.arrows} % LaTeX and plain TeX when using TikZ
\usetikzlibrary[ext.arrows] % ConTeXt when using TikZ
```

This library adds arrows to PGF/Ti*k*Z.

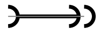
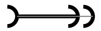
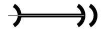
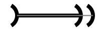
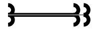
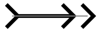
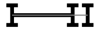
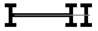
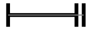
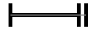
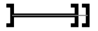
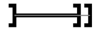
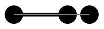
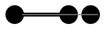
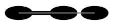
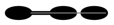
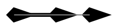
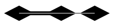
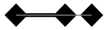
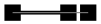
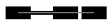
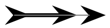
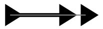
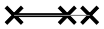
Q & A: [**ArrowLoop-Q**, **ArrowHug-Q**, **ArrowUntipped1-Q**, **ArrowUntipped2-Q**] & [**ArrowLoop-A**, **ArrowHug-A**, **ArrowUntipped1-A**, **ArrowUntipped2-A**]

The arrow tips of the `arrows.meta` library always just touch the end of original line – which is usually what you want.

But for some arrow tips (and when they lie along a path) it makes sense that these tips shoot a bit over the end of the line. This is why these arrow tips exist. They can be categorized into three groups:

1. Centered
2. Untipped
3. Overtipped[6]

Not all original arrow tips got all variants. For a summary, refer to table on the right side. As with the original tips of the `arrows.meta` library these can be organized in the following categories.

Group	Original	Centered	Untipped	Overtipped
Barbed	Arc Barb			–
	Parenthesis			–
	Hooks		–	–
	Straight Barb		–	–
	Tee Barb			–
	Bar			–
	Bracket			–
Geometric	Circle			–
	Ellipse			–
	Kite		–	–
	Diamond		–	–
	Turned Square		–	–
	LaTeX	–	–	–
	Square		–	–
	Rectangle		–	–
	Stealth		–	–
	Triangle		–	–
Rays	Rays		–	–

[6] The Overtipped arrow tips aren't yet implemented.

20.1 Centered

20.1.1 Barbed Arrow Tips

Arrow Tip Kind `ext_Centered Arc Barb`

pre 0.6 `Centered Arc Barb`

This is a variant of the `Arc Barb` tip. The center of the arc lies on the original end of the path.

Arrow Tip Kind `ext_Centered Bar`

pre 0.6 `Centered Bar`

A variant of the simple `Bar` tip. This is a simple instance of `ext_Centered Tee Barb` for length zero.

The middle of the line will lie on original end of the path.

Arrow Tip Kind `ext_Centered Bracket`

pre 0.6 `Centered Bracket`

This is a variant of the `Bracket` tip and therefore an instance of the `ext_Centered Tee Barb` arrow tip that results in something resembling a bracket.

The middle of the vertical part will lie on the original end of the path.

Arrow Tip Kind `ext_Centered Hooks`

pre 0.6 `Centered Hooks`

A variant of the `Hooks` tip. The starting point of the hooks will lie on the original end of the path.

Arrow Tip Kind `ext_Centered Parenthesis`

pre 0.6 `Centered Parenthesis`

This is a variant of the `Parenthesis` tip and thus an instance of the `ext_Centered Arc Barb` arrow tip.

Arrow Tip Kind `ext_Centered Straight Barb`

pre 0.6 `Centered Straight Barb`

A variant of the `Straight Barb` tip.

Arrow Tip Kind `ext_Centered Tee Barb`

pre 0.6 `Centered Tee Barb`

A variant of the `Tee Barb` tip.

The middle of the vertical part will lie on the original end of the path.

20.1.2 Geometric Arrow Tips

Arrow Tip Kind `ext_Centered Circle`

pre 0.6 `Centered Circle`

A variant of the `Circle` tip. The center of the circle will lie on the original end of the path.

Arrow Tip Kind `ext_Centered Diamond`

pre 0.6 `Centered Diamond`

This is a variant of the `Diamond` tip and thus an instance of `ext_Centered Kite` where the length is larger than the width.

Arrow Tip Kind `ext_Centered Ellipse`

pre 0.6 `Centered Ellipse`

This is a variant of the `Ellipse` tip and thus another name for the `ext_Centered Circle` tip that is twice as wide as high.

Arrow Tip Kind `ext_Centered Kite`

pre 0.6 `Centered Kite`

A variant of the `Kite` tip.

The widest part will lie on the original end of the path.

Arrow Tip Kind `ext_Centered Rectangle`

pre 0.6 `Centered Rectangle`

A variant of the `Rectangle` tip. By default, it is twice as long as high.

Arrow Tip Kind `ext_Centered Square`

pre 0.6 `Centered Square`

A variant of the `Square` tip.

Arrow Tip Kind `ext_Centered Stealth`

pre 0.6 `Centered Stealth`

This is a variant of the `Stealth` tip.

The weighted center will lie at the original end of the path.

Arrow Tip Kind `ext_Centered Triangle`

pre 0.6 `Centered Triangle`

This is a variant of the `Triangle` tip and thus an instance of the `ext_Centered Kite` tip with zero inset.

Arrow Tip Kind `ext_Centered Turned Square`

pre 0.6 `Centered Turned Square`

This is a variant of the `Turned Square` tip and thus an instance of the `ext_Centered Kite` tip with identical width and height and mid-inset.

20.1.3 Special Arrow Tips

Arrow Tip Kind `ext_Centered Rays`

pre 0.6 `Centered Rays`

A variant of the `Rays` tip. The origin of the rays will lie on the original end of the path.

20.2 Untipped

20.2.1 Barbed Arrow Tips

Arrow Tip Kind `ext_Centered Arc Barb`

pre 0.6 `Centered Arc Barb`

This is a variant of the `Arc Barb` tip. The arrow tip will protrude half its line width over the original end of the path.

Arrow Tip Kind `ext_Untipped Bar`

pre 0.6 `Untipped Bar`

A variant of the simple `Bar` tip. This is a simple instance of `ext_Untipped Tee Barb` for length zero.

The middle of the line will lie on original end of the path.

Arrow Tip Kind `ext_Untipped Bracket`

pre 0.6 `Untipped Bracket`

This is a variant of the `Bracket` tip and therefore an instance of the `ext_Untipped Tee Barb` arrow tip that results in something resembling a bracket.

The arrow tip will protrude half its line width over the original end of the path.

Arrow Tip Kind `ext_Untipped Parenthesis`

pre 0.6 `Untipped Parenthesis`

This is a variant of the `Parenthesis` tip and thus an instance of the `ext_Untipped Arc Barb` arrow tip.

Arrow Tip Kind `ext_Untipped Tee Barb`

pre 0.6 `Untipped Tee Barb`

A variant of the `Tee Barb` tip.

The middle of the vertical part will lie on the original end of the path.

20.2.2 Geometric Arrow Tips

Arrow Tip Kind `ext_Untipped Circle`

pre 0.6 `Untipped Circle`

A variant of the `Circle` tip. This tip will protrude half its line width over the original end of the path.

Arrow Tip Kind `ext_Untipped Ellipse`

pre 0.6 `Untipped Ellipse`

This is a variant of the `Ellipse` tip and thus another name for the `ext_Untipped Circle` tip that is twice as wide as high.

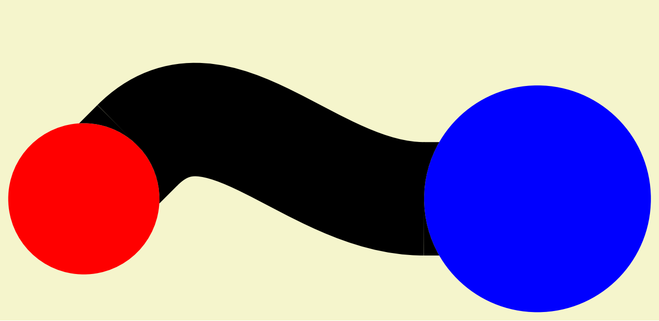
20.3 Original Arrow Tips

Arrow Tip Kind `ext_Hug Cap`

pre 0.6 `Hug Cap`

This arrow tips will hug a circle that would touch the end of the path.

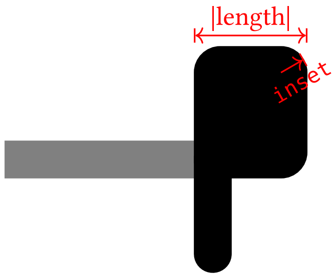
Use the `/pgf/arrow keys/length` key to set up the radius of that circle.



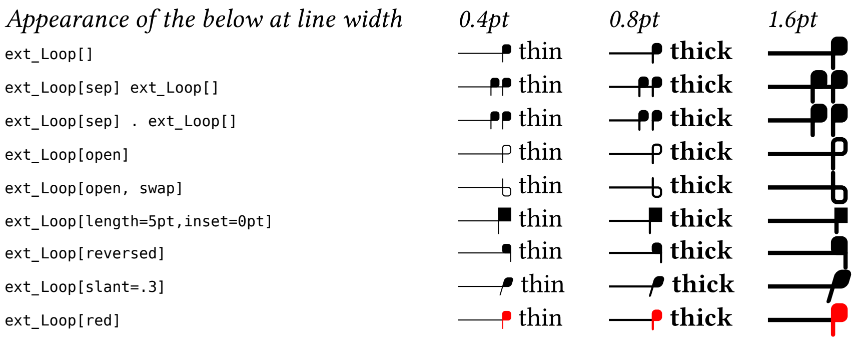
```
\usepgflibrary {ext.arrows}
\begin{tikzpicture}[
  dot/.style 2 args={
    shape=circle, outer sep=+0pt, fill={#1}, minimum size={#2}}]
\node[dot={red} {2cm}] (A)          {};
\node[dot={blue}{3cm}] (B) at (6,0) {};
\draw[
  line width=1.5cm,
  arrows={ext_Hug Cap[length=1cm]-ext_Hug Cap[length=1.5cm]}
] (A) to[out=45, in=180] (B);
\end{tikzpicture}
```

Arrow Tip Kind `ext_Loop`

pre 0.6 `Loop`



This arrow tip attaches a one-sided loop to the end of the line. The `length` refers to the length of the whole tip while the `inset` specifies the radius of the three rounded corners. The width of the tip is twice the `length` (but can't specified independently).



Appearance of the below at line width	*0.4pt*	*0.8pt*	*1.6pt*
`ext_Loop[]`	thin	**thick**	
`ext_Loop[sep] ext_Loop[]`	thin	**thick**	
`ext_Loop[sep] . ext_Loop[]`	thin	**thick**	
`ext_Loop[open]`	thin	**thick**	
`ext_Loop[open, swap]`	thin	**thick**	
`ext_Loop[length=5pt,inset=0pt]`	thin	**thick**	
`ext_Loop[reversed]`	thin	**thick**	
`ext_Loop[slant=.3]`	thin	**thick**	
`ext_Loop[red]`	thin	**thick**	

The following options have no effect: `harpoon`, `round`, `line width`.

On `double` lines, the arrow tip will not look correct.

21 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 `\pgfmathanglebetweenpoints` which involves the arctangent (`atan2`) function of PGFmath.

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

21.1 Using the reflection matrix

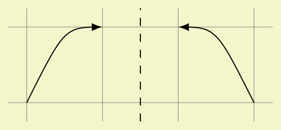
The following commands use the reflection matrix that sets the transformation matrix following

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

`\pgfexttransformxmirror`{⟨*value*⟩}

pre 0.6 `\pgftransformxmirror`

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



```
\usepgflibrary {ext.transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.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);
\pgfexttransformxmirror{1.5}

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

21.2 Using built-in transformations

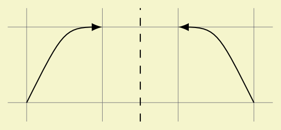
The following commands use a combination of shifting, rotating, −1 scaling, rotating back and shifting back to reach the mirror transformation.

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

`\pgfexttransformxMirror`{⟨*value*⟩}

pre 0.6 `\pgftransformxMirror`

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



```
\usepgflibrary {ext.transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.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);
\pgfexttransformxMirror{1.5}

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

\pgfexttransformymirror{⟨*value*⟩}

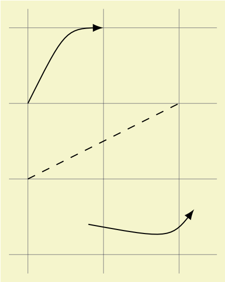
pre 0.6 \pgftransformymirror

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

\pgfexttransformmirror{⟨*point A*⟩}{⟨*point B*⟩}

pre 0.6 \pgftransformmirror

Sets up a transformation that mirrors along the line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.

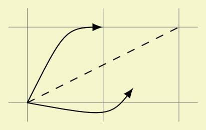


```
\usepgflibrary {ext.transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -2.25) grid (2.5, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

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

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

\pgfextqtransformmirror{⟨*point A*⟩}

pre 0.6 \pgfqtransformmirror

Sets up a transformation that mirrors along the line that goes through the origin and ⟨*point A*⟩.



```
\usepgflibrary {ext.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);
\pgfextqtransformmirror{\pgfpointxy{2}{1}}

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

\pgfexttransformyMirror{⟨*value*⟩}

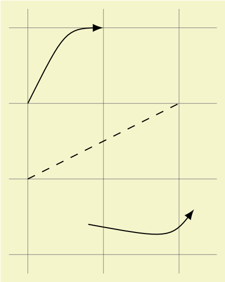
pre 0.6 \pgftransformyMirror

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

\pgfexttransformMirror{⟨*point A*⟩}{⟨*point B*⟩}

pre 0.6 \pgftransformMirror

Sets up a transformation that mirrors along the line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.

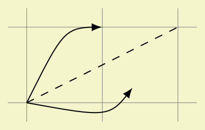


```
\usepgflibrary {ext.transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -2.25) grid (2.5, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

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

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

\pgfextqtransformMirror{⟨*point A*⟩}

pre 0.6 \pgfqtransformMirror

Sets up a transformation that mirrors along the line that goes through the origin and ⟨*point A*⟩.



```
\usepgflibrary {ext.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);
\pgfextqtransformMirror{\pgfpointxy{2}{1}}

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

22 Shape: Circle Arrow

Ti*k*Z Library ext.shapes.circlearrow

```
\usepgflibrary{ext.shapes.circlearrow}  % LaTeX and plain TeX and pure pgf
\usepgflibrary[ext.shapes.circlearrow]  % ConTeXt and pure pgf
\usetikzlibrary{ext.shapes.circlearrow} % LaTeX and plain TeX when using TikZ
\usetikzlibrary[ext.shapes.circlearrow] % ConTeXt when using TikZ
```

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

Q & A: [**ShapeCircleArrow-Q**] & [**ShapeCircleArrow-A**]

Shape ext_circle arrow

This shape is an arrow whose path is an arc – defined very similar to the `arc` path operation – that can possibly be customized with arrow tips.

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

pre 0.6 `/pgf/circle arrow start angle`

Sets the start angle.

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

pre 0.6 `/pgf/circle arrow end angle`

Sets the end angle.

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

pre 0.6 `/pgf/circle arrow delta angle`

Sets the delta angle.

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

pre 0.6 `/pgf/circle arrow arrows`

The specification will be forwarded to `\pgfsetarrows` .

A few handful styles are pre-defined.

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

pre 0.6 `/pgf/circle arrow turn left north`

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

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

pre 0.6 `/pgf/circle arrow turn left east`

As above but `circle arrow start angle = 10`.

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

pre 0.6 `/pgf/circle arrow turn left west`

As above but `circle arrow start angle = 280`.

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

pre 0.6 `/pgf/circle arrow turn left south`

As above but `circle arrow start angle = 190`.

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

pre 0.6 `/pgf/circle arrow turn right north`

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

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

pre 0.6 `/pgf/circle arrow turn right east`

As above but `circle arrow start angle = 10`.

`/pgf/ext/circle arrow turn right west` (no value)

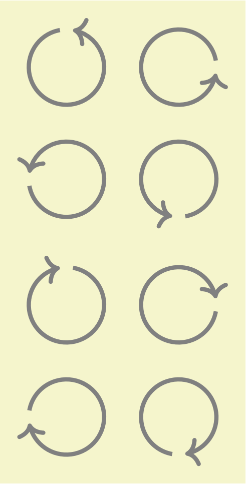
pre 0.6 `/pgf/circle arrow turn right west`

As above but `circle arrow start angle = 280`.

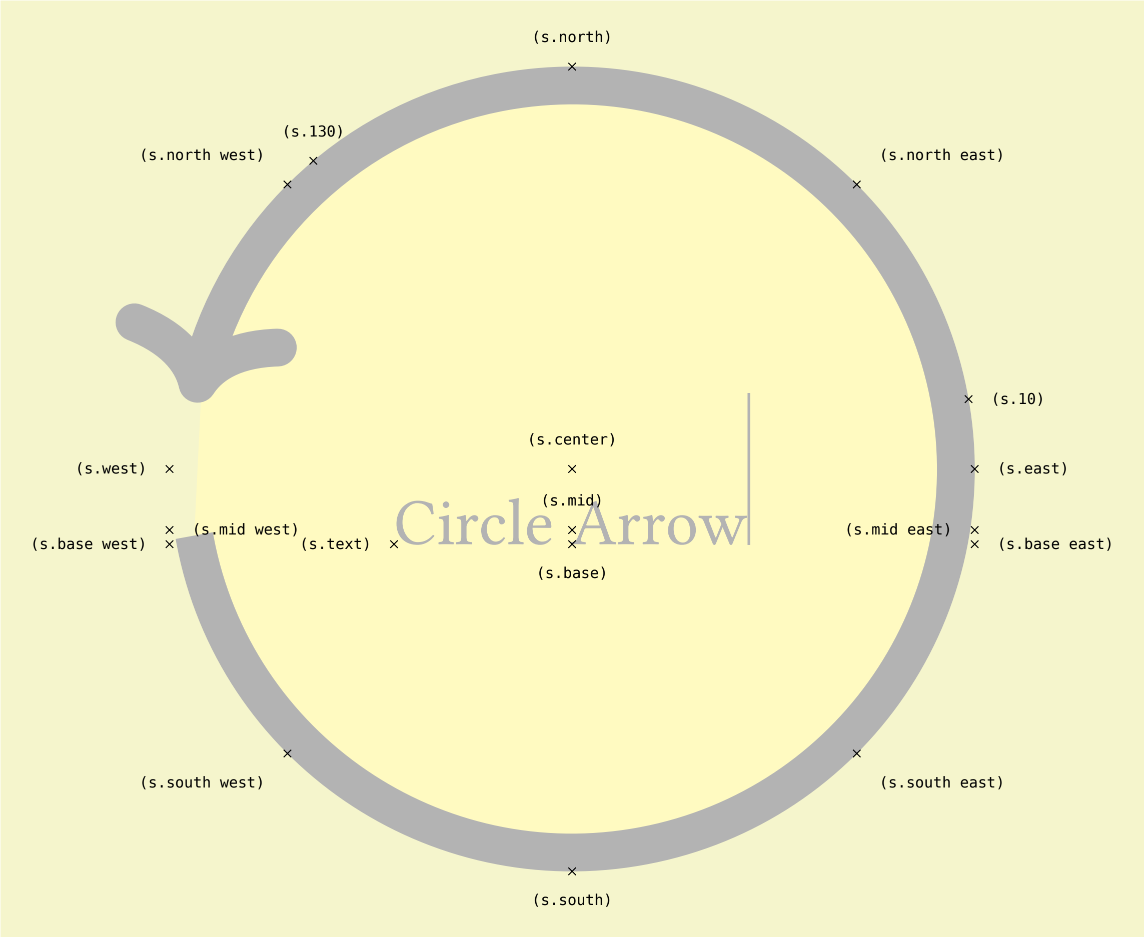
`/pgf/ext/circle arrow turn right south` (no value)

pre 0.6 `/pgf/circle arrow turn right south`

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=ext_circle arrow, ultra thick, inner sep=1em}
] (m) {
  |[ext/circle arrow turn left north]|  & |[ext/circle arrow turn left east]|   \\
  |[ext/circle arrow turn left west]|   & |[ext/circle arrow turn left south]|  \\
  |[ext/circle arrow turn right north]| & |[ext/circle arrow turn right east]|  \\
  |[ext/circle arrow turn right west]|  & |[ext/circle arrow turn right south]| \\
};
\end{tikzpicture}
```



```
\usetikzlibrary {ext.shapes.circlearrow}
\begin{tikzpicture}\Huge
\node[name=s, shape=ext_circle arrow,
  ext/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}
```

23 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.

Q & A: [**ShapeSplitCircle-Q**] & [**ShapeSplitCircle-A**]

Shape ext_circle cross split

This shape has four node parts that are placed near the center of a circle.

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

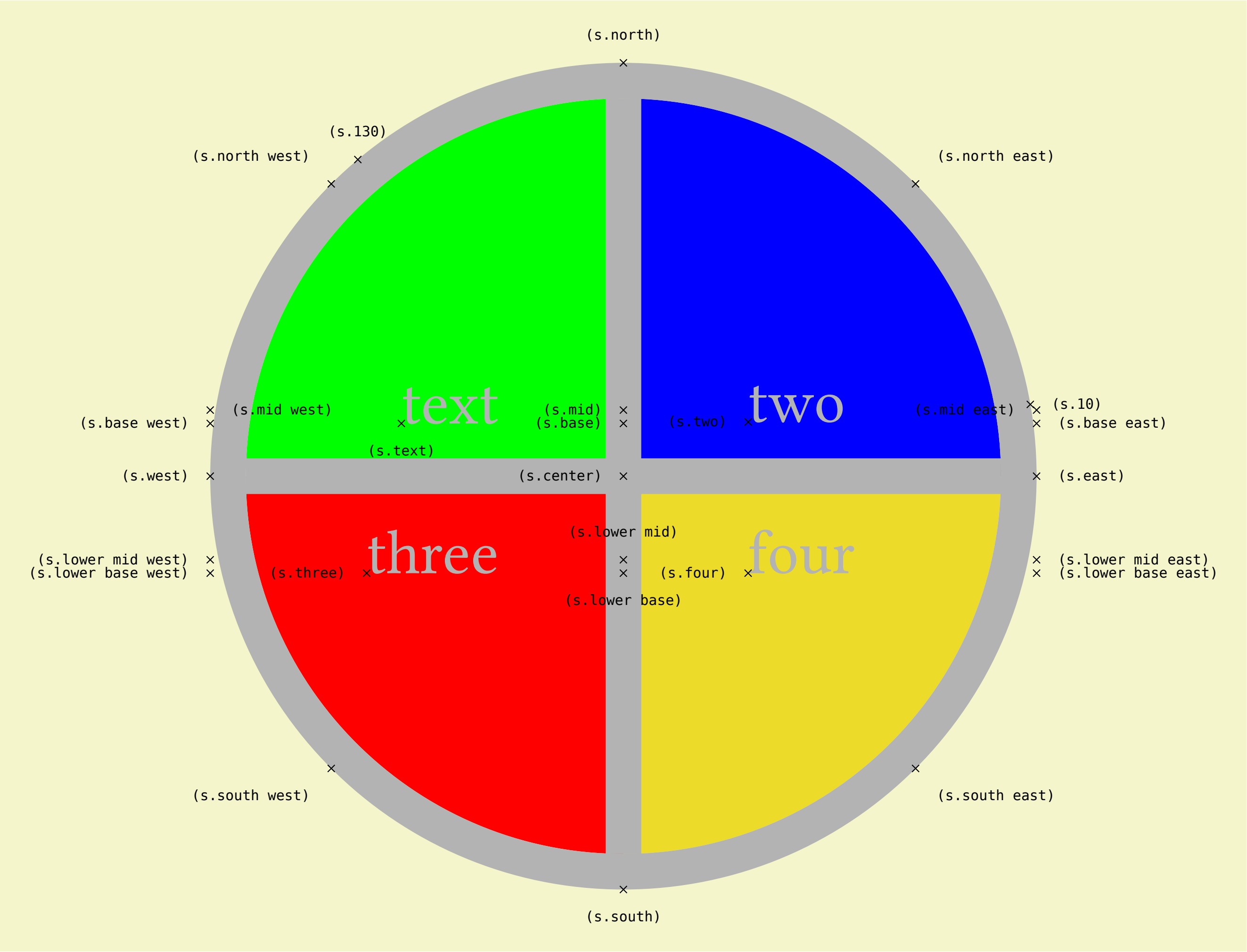
pre 0.6 `/pgf/circle cross split part fill`

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

pre 0.6 `/pgf/circle cross split uses custom fill`

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 TeX-if `\ifextpgfcirclecrosssplitcustomfill` appropriately.


(s.north)
(s.130)
(s.north west)
(s.north east)
text
two
(s.mid west)
(s.mid)
(s.base)
(s.two)
(s.mid east)
(s.10)
(s.base west)
(s.base east)
(s.text)
(s.west)
(s.center)
(s.east)
(s.lower mid)
three
four
(s.lower mid west)
(s.lower base west)
(s.three)
(s.four)
(s.lower mid east)
(s.lower base east)
(s.lower base)
(s.south west)
(s.south east)
(s.south)

```
\usepgflibrary {ext.shapes.circlecrosssplit}
\begin{tikzpicture}\Huge
\node[name=s, shape=ext_circle cross split, shape example, inner xsep=1.5cm, fill=none,
  ext/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}
```

24 Shape: Heatmark

Ti*k*Z Library ext.shapes.heatmark

```
\usepgflibrary{ext.shapes.heatmark}  % LaTeX and plain TeX and pure pgf
\usepgflibrary[ext.shapes.heatmark]  % ConTeXt and pure pgf
\usetikzlibrary{ext.shapes.heatmark} % LaTeX and plain TeX when using TikZ
\usetikzlibrary[ext.shapes.heatmark] % ConTeXt when using TikZ
```

A circular shape that has customizable rings around it.

Q & A: [**ShapeHeat-Q**] & [**ShapeHeat-A**]

Shape ext_heatmark

/pgf/ext/heatmark arcs=⟨*arcs num*⟩ (no default, initially 3)

pre 0.6 /pgf/heatmark arcs

Sets the number of arc around the circle to ⟨*arcs num*⟩.

/pgf/ext/heatmark arc width=⟨*arc width*⟩ (no default, initially 4pt)

pre 0.6 /pgf/heatmark arc width

Sets the width of the rings around the circle to ⟨*arc width*⟩.

/pgf/ext/heatmark arc sep=⟨*sep length*⟩ (no default, initially 1pt)

pre 0.6 /pgf/heatmark arc sep

Sets the whitespace between the rings to ⟨*sep length*⟩.

/pgf/ext/heatmark arc rings=⟨*rings num*⟩ (no default, initially 3)

pre 0.6 /pgf/heatmark arc rings

Sets the number of rings around the circle to ⟨*rings num*⟩

/pgf/ext/heatmark arc sep angle=⟨*sep angle*⟩ (no default, initially 20)

pre 0.6 /pgf/heatmark arc sep angle

Sets the whitespace angle between the arcs in one ring to ⟨*sep angle*⟩.

/pgf/ext/heatmark inner opacity=⟨*inner opacity*⟩ (no default, initially 0.8)

pre 0.6 /pgf/heatmark inner opacity

Sets the opacity of the inner ring to ⟨*inner opacity*⟩.

/pgf/ext/heatmark outer opacity=⟨*low opacity*⟩ (no default, initially 0.2)

pre 0.6 /pgf/heatmark outer opacity

Sets the opacity of the outer ring to ⟨*outer opacity*⟩.

The opacity of the rings between the outer and the inner ring will be interpolated by these two opacities.

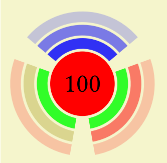
This shape takes the value of `/pgf/shape border rotate` into consideration.
For every ring and for every arc the following styke keys are tried.

`/pgf/ext/heatmark ring` ⟨*ring number*⟩ (style, no value)

`/pgf/ext/heatmark arc` ⟨*arc number*⟩ (style, no value)

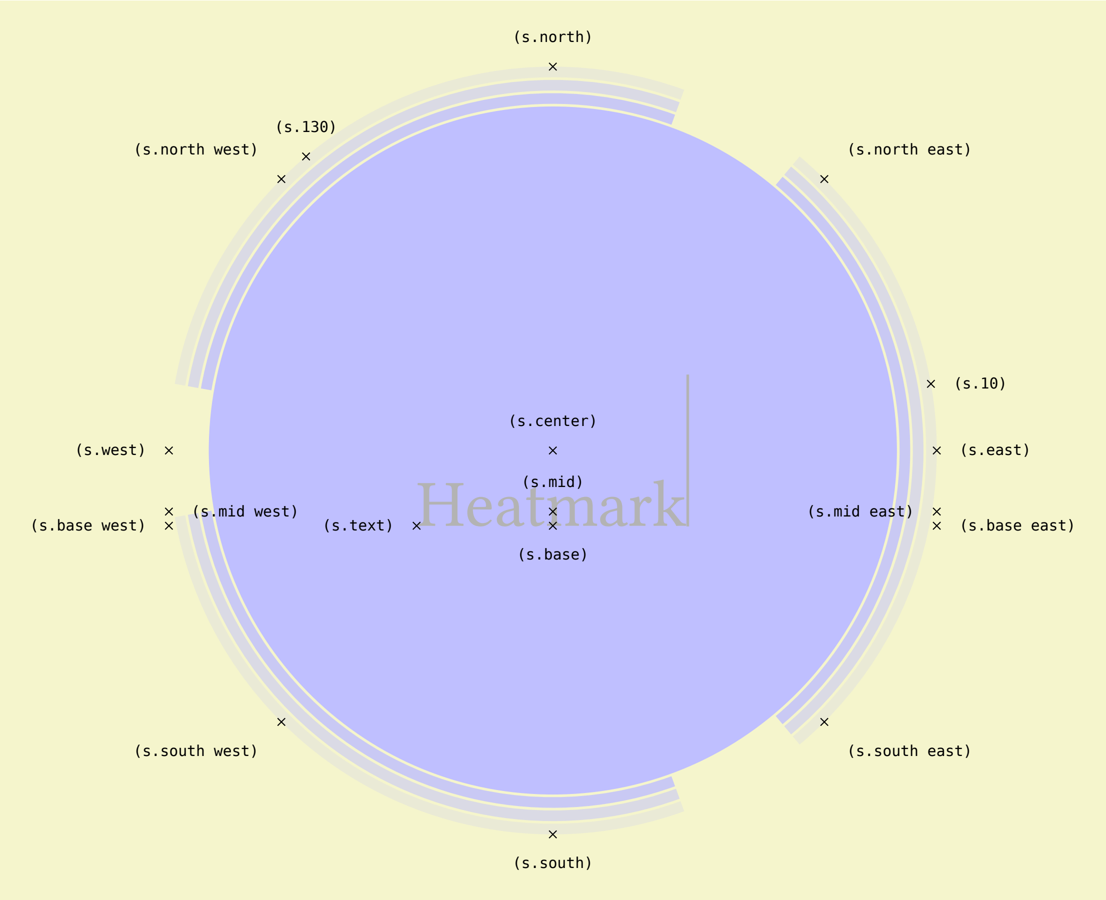
`/pgf/ext/heatmark ring` ⟨*ring number*⟩ `arc` ⟨*arc number*⟩ (style, no value)

The PGFshape is setup in a way that even Ti*k*Z styles can be used with a little bit work:



```
\usetikzlibrary {ext.shapes.heatmark}
\tikz[
  shape border rotate=90,
  /pgf/ext/heatmark ring 1/.append style={/tikz/fill=green},
  /pgf/ext/heatmark arc 1/.append style={/tikz/fill=blue},
  /pgf/ext/heatmark ring 2 arc 2/.append style={/tikz/fill=yellow!70!black}
] \node[ext_heatmark, fill=red] (n) {100};
```

It is best to use this shape with no actual border (`draw = none`) and the `outer sep` set to zero.



```
\usetikzlibrary {ext.shapes.heatmark}
\begin{tikzpicture}\Huge
\node[name=s, shape=ext_heatmark, shape example,
  fill=blue!25, draw=none, outer sep=0pt]
  {Heatmark\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}
```

25 Shape: Rectangle with Rounded Corners

Ti*k*Z 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.

Shape `ext_rectangle with rounded corners`

This library provides a `rectangle with rounded corners` where every corner can have a different radius.

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

pre 0.6 `/pgf/rectangle with rounded corners north west radius`

Sets the north west radius to ⟨*dimen*⟩.

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

pre 0.6 `/pgf/rectangle with rounded corners north east radius`

Sets the north east radius to ⟨*dimen*⟩.

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

pre 0.6 `/pgf/rectangle with rounded corners south west radius`

Sets the south west radius to ⟨*dimen*⟩.

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

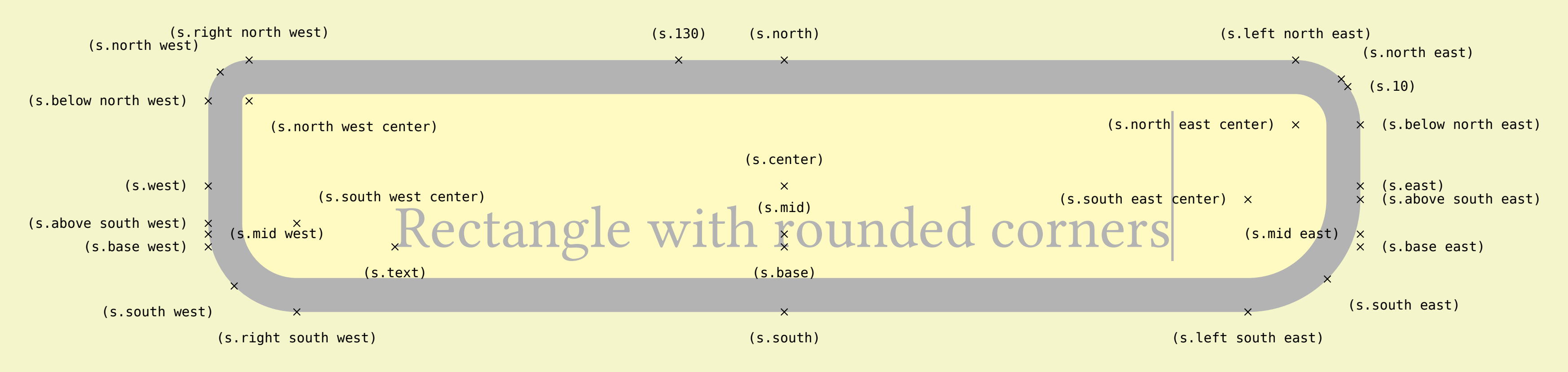
pre 0.6 `/pgf/rectangle with rounded corners south east radius`

Sets the south east radius to ⟨*dimen*⟩.

`/pgf/ext/rectangle with rounded corners radius=`⟨*dimen*⟩ (no default)

pre 0.6 `/pgf/rectangle with rounded corners radius`

Sets all radii to ⟨*dimen*⟩.



```
\usepgflibrary {ext.shapes.rectangleroundedcorners}
\begin{tikzpicture}\Huge
\node[name=s, shape=ext_rectangle with rounded corners, shape example,
  ext/rectangle with rounded corners north west radius=10pt,
  ext/rectangle with rounded corners north east radius=20pt,
  ext/rectangle with rounded corners south west radius=30pt,
  ext/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}
```

26 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".

Q & A: [**ShapeSuperEllipse-Q**] & [**ShapeSuperEllipse-A**]

Shape ext_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

$$\begin{aligned} x(t) &= |\cos t|^{\frac{2}{m}} \cdot r_x \operatorname{sgn}(\cos t) \\ y(t) &= |\sin t|^{\frac{2}{n}} \cdot r_y \operatorname{sgn}(\sin t) \end{aligned}$$

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

`/pgf/ext/superellipse x exponent=`⟨*x exponent*⟩ (no default, initially `2.5`)

pre 0.6 `/pgf/superellipse x exponent`

This sets m.

`/pgf/ext/superellipse y exponent=`⟨*y exponent*⟩ (no default, initially `2.5`)

pre 0.6 `/pgf/superellipse y exponent`

This sets n.

`/pgf/ext/superellipse step=`⟨*step*⟩ (no default, initially `5`)

This specifies the step of the underlying plot handler. The smaller ⟨*step*⟩ is, the slower computation will be.

Sensible values for ⟨*step*⟩ are integer dividers of 90, i. e. 2, 3, 5, 6, 9, 10, 15, 18, 30 and 45.

`/pgf/ext/superellipse exponent=`⟨*exponent*⟩ (no default)

pre 0.6 `/pgf/superellipse exponent`

Sets both `superellipse x exponent` and `superellipse y exponent` to ⟨*exponent*⟩.

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

`ext_superellipsex(`t`, `$2/m$`, `r_x`)`
`\pgfmathextsuperellipsex{`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 $0 \leq t \leq 90$.

`ext_superellipsey(`t`, `$2/n$`, `r_y`)`
`\pgfmathextsuperellipsey{`t`}{`$2/m$`}{`r_x`}`

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.

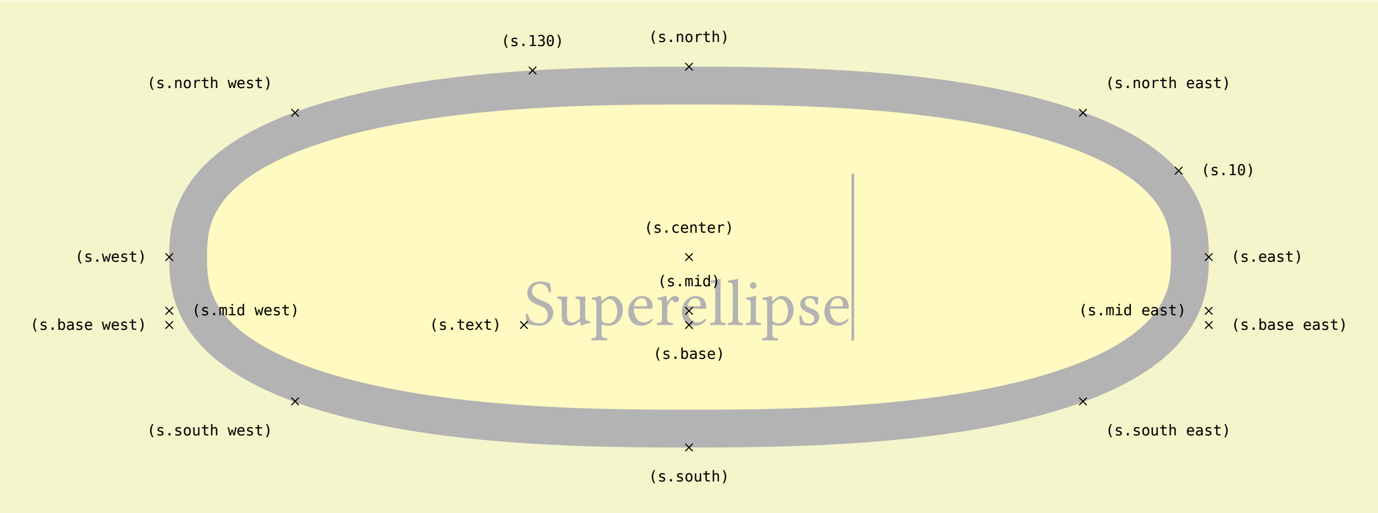
`\pgfextmathsuperellipseXY{`$\langle t \rangle$`}{`$\langle 2/m \rangle$`}{`$\langle 2/n \rangle$`}{`$\langle a \rangle$`}{`$\langle b \rangle$`}`

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

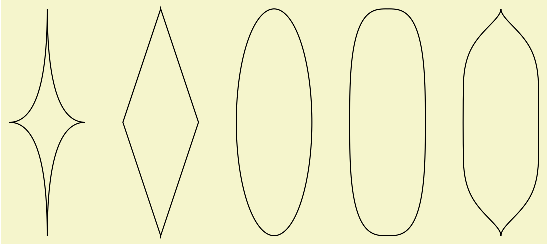
$$x = a \cos^{2/m} t$$
$$y = b \cos^{2/n} t$$

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

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



```
\usetikzlibrary {ext.shapes.superellipse}
\begin{tikzpicture}[ext/superellipse step=1]\Huge
\node[name=s,shape=ext_superellipse,shape example] {Superellipse\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}
```



```
\usetikzlibrary {ext.shapes.superellipse}
\begin{tikzpicture}[minimum width=1cm, minimum height=3cm]
\foreach \xe/\ye[count=\i] in {.5/.5, 1/1, 2/2, 3/3, .5/5}
  \node[draw, ext_superellipse, ext/superellipse x exponent=\xe, ext/superellipse y exponent=\ye] at (1.5*\i,0) {};
\end{tikzpicture}
```

27 Shape: Uncentered Rectangle

PGF Library ext.shapes.uncenteredrectangle

```
\usepgflibrary{ext.shapes.uncenteredrectangle}  % LaTeX and plain TeX
\usepgflibrary[ext.shapes.uncenteredrectangle]  % ConTeXt
```

A rectangle that has a variable horizontal center with three node parts.

Q & A: [**UncRectCD-Q**, **UncRectForest-Q**] & [**UncRectCD-A**, **UncRectForest-A**]

Shape ext_uncentered rectangle

For some alignment problems, this shape could be useful.

It has three node parts: the standard `text` part, the `left` part that is to the left of `text` and the `right` part that is to the right of `text`.

When edges are to be connected with this shape, the following key changes to which inner center this shape will calculate the appropriate point on the border.

`/pgf/ext/uncentered rectangle center`=⟨*left*⟩ or ⟨*text*⟩ or ⟨*right*⟩ or ⟨*real*⟩ (no default, initially `text`)

pre 0.6 `/pgf/uncentered rectangle center`

Sets the center that is to be used for connecting edges.

This will also move the anchors `north`, `mid`, `base` and `south` along. In the picture below, this are marked red.

`/pgf/ext/uncentered rectangle use saved center`=⟨*true*⟩ or ⟨*false*⟩ (default `true`)

pre 0.6 `/pgf/uncentered rectangle use saved center`

When this is set to true, the border anchors will use the horizontal center that was used when the node was created.

For support of the `cd` library of the `tikz-cd` package, this shape also supports a dynamic y value for its anchors `center`, `west` and `east`.

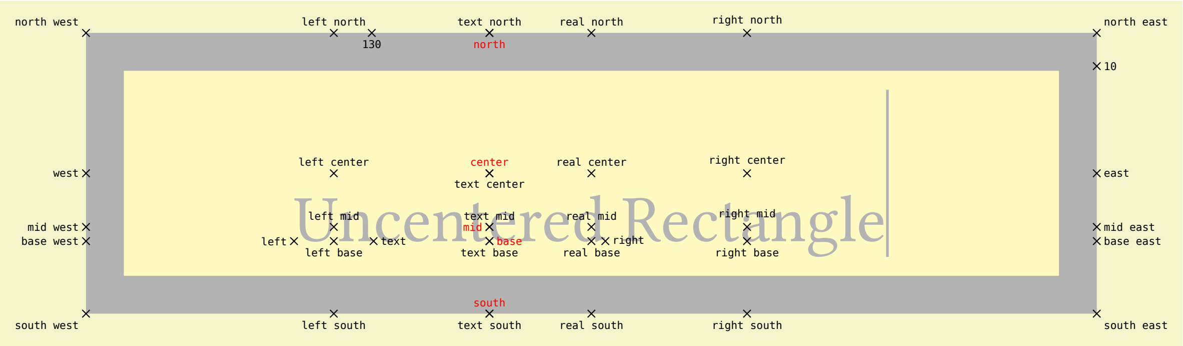
`/pgf/ext/uncentered rectangle center yshift`=⟨*dimension*⟩ (no default, initially `{}`)

pre 0.6 `/pgf/uncentered rectangle center yshift`

This determines the distance between the baseline and the `center` anchors.

If ⟨*dimension*⟩ is empty, the real vertical center will be used.

For use with `cd`, set this to `axis_height`.



```
\usepgflibrary {ext.shapes.uncenteredrectangle}
\begin{tikzpicture}[style north/.style=red, style south/.style=red, style center/.style=red, style base/.style=red, style mid/.style=red]
\Huge
\node[shape example, name=n, ext_uncentered rectangle]
  {centered \nodepart{left} Un \nodepart{right} \space Rectangle\vrule width 1pt height 2cm}
  foreach \anchor/\pos in {
   north west/above left, north/below, north east/above right, real north/above,  left north/above, right north/above, text north/above,
         west/left,       center/above,       east/right,       real center/above, left center/above,right center/above,text center/below,
     mid west/left,          mid/left,    mid east/right,       real mid/above,    left mid/above,   right mid/above,   text mid/above,
    base west/left,         base/right,  base east/right,       real base/below,   left base/below,  right base/below,  text base/below,
   south west/below left, south/above, south east/below right, real south/below,  left south/below, right south/below, text south/below,
                           10/right,         130/below,                           left/left,        right/right,       text/right}{
    plot[mark=x, only marks] coordinates {(n.\anchor)}
    node[inner sep=.1em, style \anchor/.try, style/.expand once=\pos] {\tiny\ttfamily\anchor}};
\end{tikzpicture}
```

Ti*k*Z Library ext.shapes.uncenteredrectangle

```
\usetikzlibrary{ext.shapes.uncenteredrectangle} % LaTeX and plain TeX
\usetikzlibrary[ext.shapes.uncenteredrectangle] % ConTeXt
```

This library extends the cd library (from the tikz-cd package) so that it can be used with the uncentered rectangle shape.

Q: [UncRectCD2-Q]

This library provides only one key.

/tikz/ext/tikz-cd fix (style, no value)

pre 0.6 /tikz-ext/tikz-cd fix

This key installs various “fixes” to the /tikz/commutative diagrams/every diagram style:

- Firstly, is defines a `/tikz/matrix of math nodes` key (only for the `tikzcd` environment) which allows to toggle the `/tikz/commutative diagrams/math mode` for each node.[7]
- The helpful macro `\uncrec` will be installed.

  `\uncrec{`⟨*left*⟩`}{`⟨*center*⟩`}{`⟨*right*⟩`}`

  When used as the content of a `ext_uncentered rectangle` shape, the node parts will be setup so that ⟨*left*⟩ is in the left part of the node part etc.
- Since math mode will be disabled with the `ext_uncentered rectangle`, it is automatically enabled for each node part with `\uncrec` but it can be disabled with the following key.

  `/tikz/uncrec math mode=`⟨*true*⟩ or ⟨*false*⟩ (default `true`)

  When enabled the contents of `\uncrec` will be set in math mode.
- For easy access to the `uncentered rectangle` shape, the following keys are available inside a Commutative Diagram.

  `/tikz/uncrec=`⟨*left*⟩ or ⟨*text*⟩ or ⟨*right*⟩ or ⟨*real*⟩ (style, no default, initially `text`)

  This key sets the shape to `ext_uncentered rectangle` and `/pgf/ext/uncentered rectangle center` to its argument.

  `/tikz/commutative diagrams/install uncentered rectangle in columns=`⟨*column*⟩ (style, no default)

  All nodes in column ⟨*column*⟩ will be set to the `ext_uncentered rectangle` shape.

$$\begin{tikzcd}
C_{\%_1} \arrow[dr, dash] & & m_{r_1} = C_{\%_2} - C_{\%} \arrow[dl, dash] \\
& C_{\%} & \\
C_{\%_2} \arrow[ur, dash] & & m_{r_2} = C_{\%_1} - C_{\%} \arrow[ul, dash]
\end{tikzcd}$$

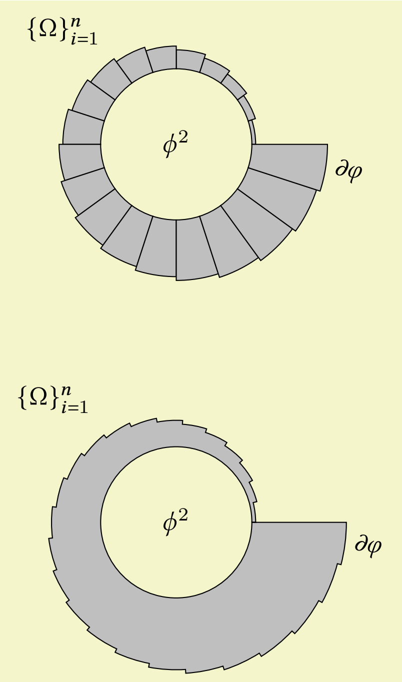
```
\usetikzlibrary {cd, ext.shapes.uncenteredrectangle}
\tikzextset{tikz-cd fix}
\newcommand*\C[1]{C_{\%_{#1}}}
\begin{tikzcd}[
  sep=tiny,
  arrows={-, gray},
  cells={font=\strut, inner xsep=.2ex, inner ysep=.1ex},
  install uncentered rectangle in column=3
]
\C{1} \drar &           & \uncrec{}{m_{r_1}}{{} = \C{2}-C_\%} \dlar\\
            & C_\% \\
\C{2} \urar &           & \uncrec{}{m_{r_2}}{{} = \C{1}-C_\%} \ular
\end{tikzcd}
```

$$\begin{tikzcd}
S \supset U_\tau \arrow[r, "\varphi_0"] \arrow[d, "\tau", "\sim"'] & U_\pi \subset T \arrow[d, "\pi", "\sim"'] \\
\operatorname{Bl}_{(0,0)}(\mathbb{A}^2) \supset V_\tau \arrow[r, "\epsilon"] & V_\pi \subset \mathbb{A}^2
\end{tikzcd}$$

```
\usetikzlibrary {cd, ext.shapes.uncenteredrectangle}
\tikzextset{tikz-cd fix}
\begin{tikzcd}[install uncentered rectangle in column/.list={1,2}]
  \uncrec{S \supset {}}{U_\tau}{}                                      \arrow[r, "\varphi_0"]
                                                                       \arrow[d, "\tau", "\sim"']
& \uncrec{}{U_\pi}{{} \subset T}                                       \arrow[d, "\pi",  "\sim"']
\\
  \uncrec{\operatorname{Bl}_{(0,0)}(\mathbb{A}^2) \supset{}}{V_\tau}{} \arrow[r, "\epsilon"]
& \uncrec{}{V_\pi}{{} \subset \mathbb{A}^2}
\end{tikzcd}
```

[7] Due to a bug with `/tikz/execute at end node` , the “automatic” math mode in matrices can’t be used with multipart nodes.

Part IV
Utilities



```
\usetikzlibrary {ext.misc}
\begin{tikzpicture}[
  declare function={bigR(\n)=smallR+.05*\n;},
  ext/declare constant={smallR=1; segments=20;},
  ext/full arc=segments]
\foreach \iN[evaluate={\endRadius=bigR(\iN+1);}, ext/use int=0 to segments-1]
  \filldraw[fill=gray!50] (\iN R:\endRadius)
    arc [radius=\endRadius, start angle=\iN R, delta angle=+1R] -- (\iN R+1R:smallR)
    arc [radius=smallR,       end angle=\iN R, delta angle=-1R] -- 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, ext/declare constant={segments=25;}, ext/full arc=segments}
\filldraw[fill=gray!50] (right:smallR)
  \foreach \iN[evaluate={\endRadius=bigR(\iN+1);}, ext/use int=0 to segments-1] {
    -- (\iN R:\endRadius) arc[radius=\endRadius, start angle=\iN R, delta angle=1R]}
    -- (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}
```

28 Calendar: Weeknumbers and more conditionals

```
\usepackage{pgfcalendar-ext} % LaTeX
\input pgfcalendar-ext.tex   % plain TeX
```

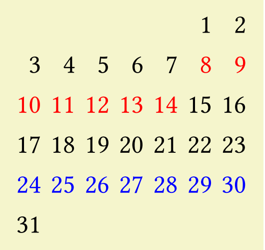
This package adds week numbers and more conditionals to the PGF package `pgfcalendar`.

Q & A: [WeekNum-Q, CalCond-Q, CalFullWeek-Q] & **[WeekNum-A, CalCond-A, CalFullWeek-A]**

28.1 Extensions

The following tests are added. In version pre 0.6, they're missing the prefix `ext/`.

- `ext/Jan` This test is passed by all dates that are in the month of January.
- `ext/Feb` as above.
- `ext/Mar` as above.
- `ext/Apr` as above.
- `ext/May` as above.
- `ext/Jun` as above.
- `ext/Jul` as above.
- `ext/Aug` as above.
- `ext/Sep` as above.
- `ext/Oct` as above.
- `ext/Nov` as above.
- `ext/Dec` as above.
- `ext/leap year=`⟨*year*⟩ This test checks whether the given year is a leap year. If ⟨*year*⟩ is omitted, it checks the year of the current date.
- `ext/and={`⟨*tests*⟩`}` This test passes when all ⟨*tests*⟩ pass.
- `ext/not={`⟨*tests*⟩`}` This test passes when ⟨*tests*⟩ do not pass.
- `ext/week of month=`⟨*num*⟩ This test passes when the date is in ⟨*num*⟩th week of the month. The first week of the month start at day 1 and ends with day 7.
- `ext/week of month'=`⟨*num*⟩ As above but counts from the last day of the month. For a month with 31 days, this means the "1st" week starts at day 25 and ends with day 31.
- `ext/calendar week of month=`⟨*num*⟩ This test passes when the date is in ⟨*num*⟩th calendar week of the month. The first week starts at the first day of the month and ends at the next Sunday.
- `ext/calendar week of month'=`⟨*num*⟩ As above but counts from the last day of the month.



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

```
\usetikzlibrary {ext.calendar-plus}
\tikz
  \calendar[
    dates=2022-10-01 to 2022-10-31,
    week list]
    if (ext/week of month=2)           [red]
    if (ext/calendar week of month'=2) [blue];
```

- `ext/yesterday={`⟨*tests*⟩`}` This test passes when the previous day passes ⟨*tests*⟩.
- `ext/week=`⟨*num*⟩ This test passes when the current week of the year equals ⟨*num*⟩.

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 28.2) are added. These are

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

28.2 Week numbering (ISO 8601)

`\pgfextcalendarjulianyeartoweek`{⟨*Julian day*⟩}{⟨*year*⟩}{⟨*week count*⟩}

pre 0.6 `\pgfcalendarjulianyeartoweek`

This command calculates the week for the ⟨*Julian day*⟩ of ⟨*year*⟩. The ⟨*week counter*⟩ must be a TeX count.

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 `\pgfextcalendarcurrentweek` will be available.

`\pgfextcalendarcurrentweek`

pre 0.6 `\pgfcalendarcurrentweek`

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

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

`\pgfextcalendarifdateweek`

pre 0.6 `\pgfcalendarifdateweek`

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

29 Repeating Things and Other Things

```
\usepackage{pgffor-ext} % LaTeX
\input pgffor-ext.tex   % plain TeX
```

This package adds small niceties to the `pgffor` package. Most of these additions are also available with the `ext.misc` library.

Warning: Consider this package experimental. At the very least, it will break the `...` notation and possibly gobbles spaces after the body.

Q & A: [**ForeachUse-Q**, **ForeachNoSep-Q**, **ForeachXparse-Q**] & [**ForeachUse-A**, **ForeachNoSep-A**, **ForeachXparse-A**]

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

`/pgf/foreach/ext/use int=`⟨*start*⟩`to`⟨*end*⟩`step`⟨*delta*⟩ (no default)

pre 0.6 `/pgf/foreach/use int`

The values ⟨*start*⟩, ⟨*end*⟩ and ⟨*delta*⟩ are evaluates by PGFmath at initialization. The part `step` ⟨*delta*⟩ is optional (⟨*delta*⟩ = 1).

`/pgf/foreach/ext/use float=`⟨*start*⟩`to`⟨*end*⟩`step`⟨*delta*⟩ (no default)

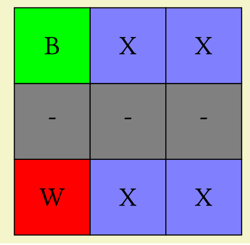
pre 0.6 `/pgf/foreach/use float`

Same as above, however the results are not truncated.

`/pgf/foreach/ext/no separator` (no value)

pre 0.6 `/pgf/foreach/no separator`

This key disables any separator between elements of the list. Every token is its own element. This also means that Unicode characters need to be grouped between `{` and `}` if LuaTeX isn't used. Spaces will be ignored.



```
\usetikzlibrary {ext.misc}
\newcommand*{\board}[3][]{%
  \begin{tikzpicture}[#1]
    \foreach[
      count=\i from 0,
      ext/no separator,
      evaluate=\i as \colX using {mod(\i,#2)},
      evaluate=\i as \rowY using {int(\i/#2)}
    ] \elem in {#3} {
        \draw[black, board/\elem/.try, ext/rectangle timer/.try=line]
          (\colX,\rowY) rectangle node {\elem} ++(1, 1);}
  \end{tikzpicture}}
\board[
  board/W/.style={fill=red},
  board/X/.style={fill=blue!50},
  board/B/.style={fill=green},
  board/-/.style={fill=gray},
]{3}{WXX---BXX}
```

`/pgf/foreach/ext/normal list` (no value)

pre 0.6 `/pgf/foreach/normal list`

This key simply disables all other special parsers and returns to the original list parser.

The following keys only work with LaTeX and cannot be used when only the `ext.misc` library or the plainTeX `pgffor-ext.tex` are loaded. For this, you will need to use `\usepackage{pgffor-ext}`.

`/pgf/foreach/ext/xparser={`⟨*argument specification*⟩`}{`⟨*foreach value*⟩`}` (no default)

pre 0.6 `/pgf/foreach/xparser`

This key can be used to specify a `xparse` specification for each element in the list.

For this to work somewhat seamless, the following needs to observed:

- Every `{`⟨*argument specification*⟩`}` get appended `u,`. This means there's always one additional mandatory argument at the end of every element.
- The `{`⟨*foreach value*⟩`}` needs to correspond to the `/pgf/foreach/var` value.

`/pgf/foreach/ext/xparser Om=`default (default `{}`)

pre 0.6 `/pgf/foreach/xparser Om`

Sets up a list whose elements may contain an optional argument inside `[]` which correspond to two `\foreach` variables, say `\Options/\Text`. The default value is the default value if the optional argument is missing.

Key handler ⟨*key*⟩`/.ext_list xparse={`⟨*argument specification*⟩`}{`⟨*comma-separated list of values*⟩`}`

pre 0.6 `.list xparse`

This handler causes the key to be used repeatedly, namely once for every element of the list of values. The ⟨*comma-separated list of values*⟩ is processed using `\foreach` and the given `xparse` ⟨*argument specification*⟩ with the aforementioned `xparser` key.

30 And a little bit more

Ti*k*Z Library `ext.misc`

```
\usetikzlibrary{ext.misc} % LaTeX and plain TeX
\usetikzlibrary[ext.misc] % ConTeXt
```

This library adds miscellaneous utilities to PGFmath, PGF or Ti*k*Z.

Q & A: [**FullArc-Q**] & [**FullArc-A**]

30.1 PGFmath

30.1.1 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.

`/pgf/ext/full arc=`⟨*num*⟩ (default `{}`)

pre 0.6 `/pgf/full arc`

The number ⟨*num*⟩ of segments will be set up. Using `full arc` with an empty value disables the segmentation and `1R` equals 1°.

The given value ⟨*num*⟩ is evaluated when the key is used and doesn't change when ⟨*num*⟩ contains variables that change.

The `R` operator can then be used.

x`R` (postfix operator; uses the `extfullarc` function)

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

30.1.2 Functions

`extstrrepeat("`*Text*`", `*x*`)`

pre 0.6 `strrepeat`

`\pgfmathextstrrepeat{"`*Text*`"}{`*x*`}`

Returns a string with *Text* repeated x times.

foofoofoofoofoo

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

`extisInString("`*String*`", "`*Text*`")`

pre 0.6 `isInString`

`\pgfmathextisInString{"`*String*`"}{"`*Text*`"}`

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

0 and 1

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

`extstrcat("`*Text A*`", "`*Text B*`", …)`

pre 0.6 `strcat`

`\pgfmathextstrcat{"`*Text A*`"}{"`*Text B*`"}{…}`

Returns the concatenation of all given parameters.

blue!21!green

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

`extisEmpty("`*Text*`")`

pre 0.6 `isEmpty`

`\pgfmathextisEmpty{"`*Text*`"}`

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

0 and 1 and 1

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

`extatanXY(`*x*`,`*y*`)`

pre 0.6 `atanXY`

`\pgfmathextatanXY{`*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{extatanXY(3,4)} \pgfmathresult
```

`extatanYX(`*y*`,`*x*`)`

pre 0.6 `atanYX`

`\pgfmathextatanYX{`*y*`}{`*x*`}`

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

53.13011

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

30.1.3 Functions: using coordinates

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

`extanglebetween("`*p1*`", "`*p2*`")`

pre 0.6 `anglebetween`

`\pgfmathextanglebetween{"`*p1*`"}{"`*p2*`"}`

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

`extqanglebetween("`*p*`")`

pre 0.6 `qanglebetween`

`\pgfmathextqanglebetween{"`*p*`"}`

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

`extdistancebetween("`*p1*`", "`*p2*`")`

pre 0.6 `distancebetween`

`\pgfmathextdistancebetween{"`*p1*`"}{"`*p2*`"}`

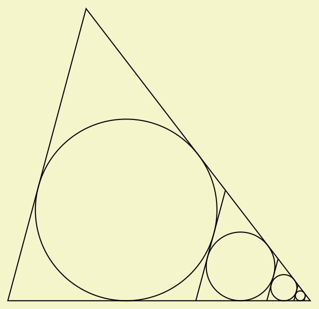
Return the distance (in pt) between the centers of the nodes *p1* and *p2*.

`extqdistancebetween("`*p*`")`

pre 0.6 `qdistancebetween`

`\pgfmathextqdistancebetween{"`*p*`"}`

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



```
\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{extdistancebetween("B","C")}
  \pgfmathsetmacro\triB{extdistancebetween("C","A")}
  \pgfmathsetmacro\triC{extdistancebetween("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}
```

30.2 PGFfor

This library loads also most of the functions of the `pgffor-ext` of section 29 on page 72.

30.3 PGFkeys

pgfkeys Library `ext.pgfkeys-plus`

```
\usepgfkeyslibrary{ext.pgfkeys-plus} % LaTeX and plain TeX
\usepgfkeyslibrary[ext.pgfkeys-plus] % ConTeXt
```

This extends PGFkeys and adds helpful `/utils` keys as well as handlers. This library gets loaded by the `ext.misc` library.

30.3.1 Conditionals

`/utils/ext/pgfmath if={`⟨*cond*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}` (no default)

pre 0.6 `/utils/if`

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 actually optional.

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

`/utils/ext/if=`⟨*token A*⟩⟨*token B*⟩`{`⟨*true*⟩`}{`⟨*false*⟩`}` (no default)

pre 0.6 `/utils/TeX/if`

This key checks via `\if` if ⟨*token A*⟩ matches ⟨*token B*⟩ and applies the styles ⟨*true*⟩ if it does, otherwise ⟨*false*⟩.

As a side effect on how PGFkeys parses argument, the ⟨*false*⟩ argument is actually optional.

`/utils/ext/ifx=`⟨*token A*⟩⟨*token B*⟩`{`⟨*true*⟩`}{`⟨*false*⟩`}` (no default)

pre 0.6 `/utils/TeX/ifx`

As above but via `\ifx`.

`/utils/ext/ifnum={`⟨*num cond*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}` (no default)

pre 0.6 `/utils/TeX/ifnum`

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

As a side effect on how PGFkeys parses arguments, the ⟨*false*⟩ argument is actually optional.

`/utils/ext/ifdim=`⟨*dim cond*⟩⟨*true*⟩⟨*false*⟩ (no default)

pre 0.6 `/utils/TeX/ifdim`

As above but with `\ifdim`.

`/utils/ext/ifempty=`⟨*Text*⟩⟨*true*⟩⟨*false*⟩ (no default)

pre 0.6 `/utils/TeX/ifempty`

This checks whether ⟨*Text*⟩ is empty and applies styles ⟨*true*⟩ if true, otherwise ⟨*false*⟩.

`/utils/ext/ifxempty=`⟨*Text*⟩⟨*true*⟩⟨*false*⟩ (no default)

pre 0.6 `/utils/TeX/ifxempty`

This checks whether fully expanded ⟨*Text*⟩ is empty and applies styles ⟨*true*⟩ if true, otherwise ⟨*false*⟩.

30.3.2 Handlers

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

Key handler ⟨*key*⟩`/.ext_pgfmath=`⟨*eval*⟩

pre 0.6 `.pgfmath`

This handler evaluates ⟨*eval*⟩ before it is handed to the key.

This handler works almost the same as the `.evaluated` handler but it does its evaluation in a group so that the result will not overwrite any other results.

Key handler ⟨*key*⟩`/.ext_pgfmath int=`⟨*eval*⟩

pre 0.6 .pgfmath int

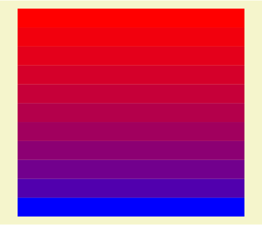
As above but truncates the result.

Key handler ⟨*key*⟩/.ext_pgfmath wrap={⟨*wrapper*⟩}{⟨*eval*⟩}

pre 0.6 .pgfmath wrap

This feeds the result of ⟨*eval*⟩ as #1 to ⟨*wrapper*⟩.

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



```
\usepgfkeyslibrary {ext.pgfkeys-plus}
\tikz\foreach \i in {0,10,...,100}
\draw[
  line width=+.25cm,
  % needs ## because its inside the \foreach body
  color/.ext_pgfmath wrap={red!##1!blue}{sqrt(\i)*10}
]
  (0,\i/40) -- +(right:3);
```

Key handler ⟨*key*⟩/.ext_pgfmath if={⟨*cond*⟩}{⟨*true*⟩}{⟨*false*⟩}

pre 0.6 .pgfmath if

Evaluates ⟨*cond*⟩ with PGFMath and returns ⟨*true*⟩ or ⟨*false*⟩ to the used key respectively.

Key handler ⟨*key*⟩/.ext_if=⟨*token A*⟩⟨*token B*⟩{⟨*true*⟩}{⟨*false*⟩}

pre 0.6 .if

Checks via \if if ⟨*token A*⟩ matches ⟨*token B*⟩ and applies the value ⟨*true*⟩ if it does, otherwise ⟨*false*⟩.

Key handler ⟨*key*⟩/.ext_ifx=⟨*token A*⟩⟨*token B*⟩{⟨*true*⟩}{⟨*false*⟩}

pre 0.6 .ifx

As above but via \ifx.

Key handler ⟨*key*⟩/.ext_ifnum={⟨*ifnum cond*⟩}{⟨*true*⟩}{⟨*false*⟩}

pre 0.6 .ifnum

Checks via \ifnum if ⟨*ifnum cond*⟩ and applies the value ⟨*true*⟩ if it does, otherwise ⟨*false*⟩.

Key handler ⟨*key*⟩/.ext_ifdim={⟨*ifdim cond*⟩}{⟨*true*⟩}{⟨*false*⟩}

pre 0.6 .ifdim

As above but via \ifdim.

Key handler ⟨*key*⟩/.ext_ifxempty={⟨*Text*⟩}{⟨*true*⟩}{⟨*false*⟩}

pre 0.6 .ifxempty

Checks whether a fully expanded ⟨*Text*⟩ is empty and applies the value ⟨*true*⟩ if it does, otherwise ⟨*false*⟩.

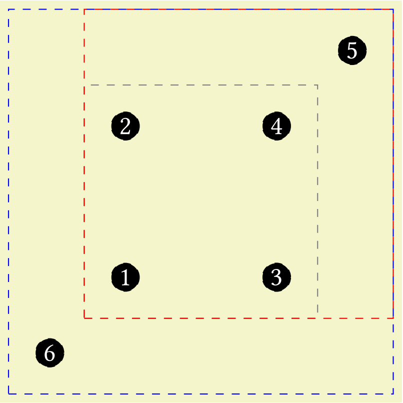
Key handler ⟨*key*⟩/.ext_ifempty={⟨*Text*⟩}{⟨*true*⟩}{⟨*false*⟩}

Checks whether ⟨*Text*⟩ is empty and applies the value ⟨*true*⟩ if it does, otherwise ⟨*false*⟩.

Key handler ⟨*key*⟩/.ext_List={⟨⟨*e1*⟩, ⟨*e2*⟩, …, ⟨*en*⟩⟩}

pre 0.6 .List

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)}
    \node[circle, fill, inner sep=1pt, text=white] (point-\cnt) at \point {\cnt};
  \node[gray, fit/.ext_List={(point-1),(point-...),(point-4)}] {};
  \node[red,  fit/.ext_List={(point-1),(point-...),(point-5)}] {};
  \node[blue, fit/.ext_List={(point-1),(point-...),(point-6)}] {};
\end{tikzpicture}
```

30.4 Ti*k*Z

`/tikz/ext/reverse clip=`⟨*direction*⟩ (default `counter clockwise`)

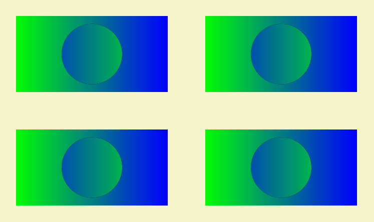
pre 0.6 `/tikz/reverse clip`

This key installs a very big rectangle which is either constructed `counter clockwise` (like the `circle` path operation) or `clockwise`.

`/tikz/ext/clip rule=`⟨*direction*⟩ (default `even odd`)

pre 0.6 `/tikz/clip rule`

This key switches directly[8] to the specified rule which is either `even odd` or `nonzero`. This corresponds to the `/tikz/even odd rule` and `/tikz/nonzero rule` keys.



```
\usetikzlibrary {ext.misc}
\newcommand*\myDiagram[1]{
  \fill[left color=blue, right color=green] (0, 0) rectangle (2, 1);
  \clip (1, .5) #1 [ext/reverse clip];
  \fill[left color=green, right color=blue] (0, 0) rectangle (2, 1);
}
\begin{tikzpicture}[radius=.4, row sep=5mm, column sep=5mm]
\matrix[
  row 2/.append style={ext/clip rule=even odd},
  column 1/.append style={ext/reverse clip/.default=clockwise}
]{
  \myDiagram{circle[]} &
  \myDiagram{+(0:.4) arc[start angle=0, delta angle=-360] -- cycle}
\\
  \myDiagram{circle[]} &
  \myDiagram{+(0:.4) arc[start angle=0, delta angle=-360] -- cycle}
\\};
\end{tikzpicture}
```

[8] Meaning, it directly executes `\pgfseteorule` /`\pgfsetnonzerorule` and doesn't accumulates where Ti*k*Z throws an error.

Part V

Changelog, Index & References

Changelog

Version 0.6 (2024-10-30)

- Added `\tikzextset`, `\tikzextversion` and `\tikzextversionnumber`
- Added six new `auto` placement mechanisms: `ext/above`, `ext/below`, `ext/west`, `ext/east`, `ext/north` and `ext/south`.
- Added `ext/auto offset` for `auto` placement.
- Added `ext/precise auto angle`.
- Added Ti*k*Z library `ext.arrows-plus`.
- Added Ti*k*Z library `ext.topaths.autobend`.
- Made `ext.node-families` and `ext.scalepicture` memoizable.

Version 0.5.1 (2023-04-02)

- Added PGF library `ext.arrows`.
- Bugfix to `ext.pgfkeys-plus`. [**GH6**]

Version 0.5 (2023-03-17)

- Added package `pgffor-ext`.
- Added Ti*k*Z library `ext.nodes`.
- Added Ti*k*Z library `ext.layers`.
- Bugfixes to `ext.calendar-plus`.
- Allow the original `rectangle` timer with `ext.paths.timer`.

Version 0.4.2 (2022-10-30)

- Added Ti*k*Z library `ext.scalepicture`.
- Bugfixes to `shapes.uncenteredrectangle`, `paths.ortho`, `positioning-plus` and `pgfcalender-ext`.

Version 0.4.1 (2022-10-23)

- Cleaned up directory structure of documentary.
- Added PGFkeys library `ext.pgfkeys-plus`.
- Added shape `uncentered rectangle` (PGF library `ext.shapes.uncenteredrectangle`).
- Fixed `ext.paths.arcto` – again [**GH2**].

Version 0.4 (2022-10-10)

- CTAN version of 0.3.1

Version 0.3.1 (2022-10-09)

- Fixed `ext.paths.ortho` keys `only vertical first` and `only horizontal first`.
- Moved all (except the `to paths`) to namespace `/tikz/ortho`. `/tikz/hvvh` and `/tikz/udlr` are considered deprecated.
- Fixed `\pgfcalendarjulianyeartoweek`.
- Added more calendar tests.
- Added directory structure.

Version 0.3 (2022-09-24)

- Added shape `circle arrow` (PGF library `ext.shapes.circlearrow`).

- Added shape `circle cross split` (PGF library `ext.shapes.circlecrosssplit`).
- Added shape `heatmark` (PGF library `ext.shapes.heatmark`).
- Added shape `rectangle with rounded corners` (PGF library `ext.shapes.rectangleroundedcorners`).
- Added shape `superellipse` (PGF library `ext.shapes.superellipse`).
- Added Ti*k*Z library `ext.node-families.shapes.geometric`.
- Fixed `ext.node-families`' key `size`.
- Renamed internal macros to use custom namespace starting with `\tikzext@`.
- Added some references.

Version 0.2 (2022-08-21)

- Added Ti*k*Z library `ext.positioning-plus`.
- Added Ti*k*Z library `ext.node-families`.

Version 0.1 (2022-08-16)

- Added Ti*k*Z library `ext.calendar-plus`.
- Added Ti*k*Z library `ext.misc`.
- Added Ti*k*Z library `ext.paths.arcto`.
- Added Ti*k*Z library `ext.paths.ortho`.
- Added Ti*k*Z library `ext.paths.timer`.
- Added Ti*k*Z library `ext.patterns.images`.
- Added Ti*k*Z library `ext.topaths.arcthrough`.
- Added Ti*k*Z 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 and references to functionalities outside of PGF/TikZ.