The string-diagrams package*
Draw string diagrams using TikZ

Paolo Brasolin
paolo.brasolin@gmail.com

v0.3.0 (2026/11/07)

Please note this is the major version zero, meant for initial development: *anything MAY change at any time.* The upside is that this is the best time to contribute! Of course you can also just keep the `sty` along with your code and not care at all.

1 Documentation

`/pgf/box`

New: 2023-05-31
Updated: 2026-11-07

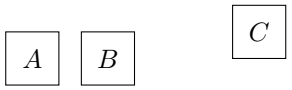
To draw boxes, you use this style on a node. Its label is already typeset in math mode, so write it without dollars: `{f}`, as `{$f$}` breaks the node.

```
\begin{tikzpicture}
  \node[box] {A};
\end{tikzpicture}
```



You can draw multiple boxes using any of your standard Ti*k*Z positioning techniques. Don't forget to label the nodes so you can easily reference them. The last box below is placed with the `calc` library of Ti*k*Z, which you load with `\usetikzlibrary{calc}`.

```
\begin{tikzpicture}
  \node[box] (A) at (0,0) {A};
  \node[box, right of=A] (B) {B};
  \node[box] (C) at ($(B)+(2cm,1em)$) {C};
\end{tikzpicture}
```



*Thanks!

/pgf/box␣ports␣north /pgf/box ports north=⟨*integer*⟩
/pgf/box␣ports␣east /pgf/box ports east=⟨*integer*⟩
/pgf/box␣ports␣south /pgf/box ports south=⟨*integer*⟩
/pgf/box␣ports␣west /pgf/box ports west=⟨*integer*⟩

New: 2023-06-12
Updated: 2026-11-07

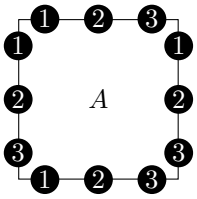
You can open up any number of ports on any side of a box using the appropriate key; boxes have none until you do. Like other Ti*k*Z keys, these can also be set on a picture or a scope, for all the boxes in it. Then, you can refer to the opened ports by their index: they are numbered from left to right on the north and south sides, and from top to bottom on the east and west sides. Asking a box for a port it doesn't have is an error.

```
\begin{tikzpicture}[
  marker/.style={circle, fill, inner sep=1pt, text=white},
]

\node[
  box,
  box ports north=3,
  box ports east=3,
  box ports south=3,
  box ports west=3,
  minimum width=6em,
  minimum height=6em,
] (A) {A};

\foreach \side in {north,east,south,west}
  \foreach \index in {1,...,3}
    \node[marker] at (A.\side.\index) {\index};

\end{tikzpicture}
```



/pgf/box␣ports /pgf/box ports=⟨*integer*⟩/⟨*integer*⟩/⟨*integer*⟩/⟨*integer*⟩

New: 2023-06-12
Updated: 2026-11-07

The `box ports` key is a shortcut to set the number of ports on all sides at once. It takes exactly four counts, for the north, east, south and west sides.

```
\begin{tikzpicture}[
  marker/.style={circle, fill, inner sep=1pt},
]

\node[box, box ports=1/2/3/4] (A) {A};

\foreach \side/\n in {north/1,east/2,south/3,west/4}
  \foreach \index in {1,...,\n}
    \node[marker] at (A.\side.\index) {};

\end{tikzpicture}
```



The same value can also be passed to the `box` key itself; without it, `box` keeps the counts in force.

`\wires` `\wires[⟨TikZ keys⟩]{⟨connectivity⟩}`

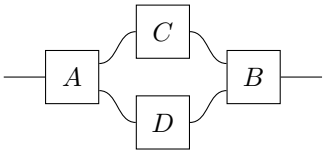
New: 2023-05-31
Updated: 2026-11-07

To connect boxes, you can use the `\wires` macro. Its optional argument is Ti*k*Z styling for the wires. Its mandatory argument lists them, grouped by the node they start from: ⟨*node*⟩ = { ⟨*anchor*⟩ = ⟨*node*⟩.⟨*anchor*⟩, ... }.

Wires start and end at the sides of nodes (`north`, `east`, `south` and `west`), at their corners (`north east` and so on), and at the ports of boxes (such as `east.1`). They leave sides and ports straight out, and corners diagonally. Other anchors, such as `center`, have no direction, so wires can't start or end there. Every entry is a wire: a node can be listed twice, and an anchor can carry several wires, both going out and coming in.

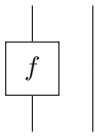
The boundary of the picture is a node too, called `@`: wires to its sides are the loose ends of the diagram. They go straight out to the side they are wired to, or bend towards it.

```
\begin{tikzpicture}[scale=0.6]
  \node[box=0/2/0/1] (A) at (-2, 0) {A};
  \node[box=0/1/0/2] (B) at (+2, 0) {B};
  \node[box=0/1/0/1] (C) at ( 0,+1) {C};
  \node[box=0/1/0/1] (D) at ( 0,-1) {D};
  \wires{
    A = { east.1 = C.west
        , east.2 = D.west
        , west = @.west },
    B = { east = @.east },
    C = { east = B.west.1 },
    D = { east = B.west.2 },
  }
\end{tikzpicture}
```



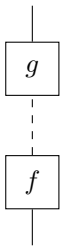
The boundary surrounds the picture as it is when `\wires` is first called, with a margin, and later calls in the same picture share it: so draw all nodes before the wires. A wire across the boundary runs through its middle. To place one elsewhere, as an identity beside a box, wire a coordinate to the boundary both ways. Place the coordinate with a path, as below, since `\coordinate` doesn't count towards the size of the picture, and the boundary might not reach it.

```
\begin{tikzpicture}
  \node[box=1/0/1/0] (f) at (0,0) {f};
  \path (0.8,0) coordinate (i);
  \wires{
    f = { north.1 = @.north
        , south.1 = @.south },
    i = { north = @.north
        , south = @.south },
  }
\end{tikzpicture}
```



The keys of a call apply to all its wires, and come after the angles that `\wires` computes, so `out` and `in` override them. To style some wires differently, call `\wires` more than once.

```
\begin{tikzpicture}
  \node[box=1/0/1/0] (f) at (0,0) {f};
  \node[box=1/0/1/0] (g) at (0,1.5) {g};
  \wires{ f = { south.1 = @.south } }
  \wires[dashed]{
    f = { north.1 = g.south.1 } }
  \wires{ g = { north.1 = @.north } }
\end{tikzpicture}
```



Wires take their directions from the names of anchors, in the frame of the picture. So pictures can be shifted and scaled, but not rotated or mirrored, and rotated nodes aren't supported.

Up to version 0.2.1, `\wires` took a third argument listing the loose ends. They are now wires to the boundary, and a third argument is no longer read.

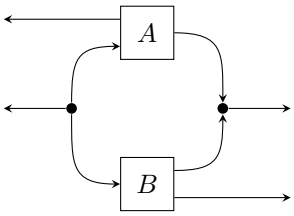
/pgf/dot

New: 2023-05-31

Updated: 2026-11-07

To split and join wires, you can use `dot`s. Wires attach to their sides and corners, and leave corners diagonally. Remember to have fun with styling wires.

```
\begin{tikzpicture}
  \node[box=0/1/0/2] (A) at ( 0,+1) {A};
  \node[box=0/2/0/1] (B) at ( 0,-1) {B};
  \node[dot] (x) at (+1, 0) {};
  \node[dot] (y) at (-1, 0) {};
  \wires[looseness=1.5,-stealth]{
    A = { east.1 = x.north
        , west.1 = @.west },
    B = { east.1 = x.south
        , east.2 = @.east },
    y = { north = A.west.2
        , south = B.west
        , west = @.west },
    x = { east = @.east },
  }
\end{tikzpicture}
```




```
\begin{tikzpicture}
  \node[dot] (c) at (0,0) {};
  \node[dot] (m) at (0,1.2) {};
  \wires{
    c = { south = @.south
        , north west = m.south west
        , north east = m.south east },
    m = { north = @.north },
  }
\end{tikzpicture}
```



That's it. This is the package, for now.

2 Implementation

Open the DocStrip guards and set the internal namespace prefix (as per LaTeX3 DocStrip convention).

```
⟨*package⟩
⟨@@=stridi⟩
```

Load the essential support (expl3, and Ti*k*Z with the `fit` library that the boundary of `\wires` needs) "up-front".

```
\RequirePackage{expl3}[2023/06/05]
\RequirePackage{tikz}[2023/01/15]
\usetikzlibrary{fit}
```

Identify the package and give the over all version information.

```
\ProvidesExplPackage
  {string-diagrams}
  {2026/11/07}
  {0.3.0}
  {Draw string diagrams using TikZ}
```

/pgf/box␣ports␣north
/pgf/box␣ports␣east
/pgf/box␣ports␣south
/pgf/box␣ports␣west
/pgf/box␣ports

Define high level keys to configure the number of ports on each side. Boxes start with no ports. The shortcut checks that it gets four counts, and sets the keys of the sides to their values; given nothing, it leaves them alone, so that the `box` style without counts keeps the ones in force.

```
\msg_new:nnn { string-diagrams } { port-counts }
  {
    Box~ports~take~four~counts,~for~the~north,~east,~south~and~west~sides,~
    as~in~1/2/1/2,~not~‘#1’.
  }
\seq_new:N \l__stridi_port_counts_seq
\pgfkeys{
  /pgf/box~ports~north/.initial=0,
  /pgf/box~ports~east/.initial=0,
  /pgf/box~ports~south/.initial=0,
  /pgf/box~ports~west/.initial=0,
  /pgf/box~ports/.code={
    \tl_if_blank:nF { #1 }
      {
        \seq_set_split:Nnn \l__stridi_port_counts_seq { / } { #1 }
        \int_compare:nNnTF { \seq_count:N \l__stridi_port_counts_seq } = { 4 }
          {
            \exp_args:Ne \pgfkeysalso
              {
                /pgf/box~ports~north = \seq_item:Nn \l__stridi_port_counts_seq { 1 } ,
                /pgf/box~ports~east  = \seq_item:Nn \l__stridi_port_counts_seq { 2 } ,
                /pgf/box~ports~south = \seq_item:Nn \l__stridi_port_counts_seq { 3 } ,
                /pgf/box~ports~west  = \seq_item:Nn \l__stridi_port_counts_seq { 4 } ,
              }
          }
          { \msg_error:nnn { string-diagrams } { port-counts } { #1 } }
```

```
      }
  },
}
```

(*End of definition for* `/pgf/box ports north` *and others. These functions are documented on page 2.*)

`\__stridi_intersect_hv_lines_through:NN`

Calculates the intersection of two lines parallel to axes passing through given points on the plane.

`#1` : Point through which the vertical line passes
`#2` : Point through which the horizontal line passes

```
\cs_new_protected:Nn \__stridi_intersect_hv_lines_through:NN {
  \pgfextractx { \pgf@xa } { #1 }
  \pgfextracty { \pgf@ya } { #2 }
  \pgfpoint { \pgf@xa } { \pgf@ya }
}
```

(*End of definition for* `\__stridi_intersect_hv_lines_through:NN`.)

`\__stridi_subdivide_segment:nNNNNN`

Defines the port anchors of a side of a box: equally spaced points on a segment. Anchors are defined globally, by the first box with enough ports on that side, so each one checks the port count of the box it is asked of. The box is named in the error when an anchor of a named node is asked for, but not while placing a node by one of its own anchors.

`#1` : Side of the box
`#2` : Points count
`#3` : Point containing the x coordinate of the starting point
`#4` : Point containing the y coordinate of the starting point
`#5` : Point containing the x coordinate of the ending point
`#6` : Point containing the y coordinate of the ending point

```
\msg_new:nnn { string-diagrams } { missing-port }
  { #1~has~no~port~#2.#3,~since~its~#2~side~has~#4. }
\cs_new_protected:Nn \__stridi_subdivide_segment:nNNNNN {
  \int_step_inline:nnnn { #2 } { -1 } { 1 } {
    \cs_if_exist:cTF
      { pgf@anchor@box@#1.##1 }
      { \prg_break: }
      { \prg_do_nothing: }
    \cs_new_nopar:cpn
      { pgf@anchor@box@#1.##1 }
      {
        \int_compare:nNnTF { ##1 } > { #2 }
          {
            \msg_error:nneeee { string-diagrams } { missing-port }
              {
                \cs_if_exist:NTF \pgfreferencednodename
                  { Box~`\pgfreferencednodename' }
                  { This~box }
              }
              { #1 } { ##1 } { #2 }
            \pgfpointlineattime
              { 0.5 }
```

```
              { \__stridi_intersect_hv_lines_through:NN { #3 } { #4 } }
              { \__stridi_intersect_hv_lines_through:NN { #5 } { #6 } }
          }
          {
            \pgfmathdivide
              { 2 * ##1 - 1 }
              { 2 * #2 }
            \pgfpointlineattime
              { \pgfmathresult }
              { \__stridi_intersect_hv_lines_through:NN { #3 } { #4 } }
              { \__stridi_intersect_hv_lines_through:NN { #5 } { #6 } }
          }
      }
  }
}
```

(*End of definition for* `\__stridi_subdivide_segment:nNNNNN`.)

`box` Define a rectangular shape with configurable ports.

```
\pgfdeclareshape{box}{

  % Inherit all the structure of rectangle
  \inheritsavedanchors[from=rectangle]
  \clist_map_inline:nn
    {
      north~west,  north, north~east,
            west, center,       east,
        mid~west,    mid,   mid~east,
       base~west,   base,  base~east,
      south~west,  south, south~east,
    }
    { \inheritanchor[from=rectangle]{#1} }
  \inheritanchorborder[from=rectangle]
  \inheritbackgroundpath[from=rectangle]

  % Dump port counts into saved macros
  \savedmacro\portsnorth
    {\pgfmathtruncatemacro\portsnorth{\pgfkeysvalueof{/pgf/box~ports~north}}}
  \savedmacro\portseast
    {\pgfmathtruncatemacro\portseast{\pgfkeysvalueof{/pgf/box~ports~east}}}
  \savedmacro\portssouth
    {\pgfmathtruncatemacro\portssouth{\pgfkeysvalueof{/pgf/box~ports~south}}}
  \savedmacro\portswest
    {\pgfmathtruncatemacro\portswest{\pgfkeysvalueof{/pgf/box~ports~west}}}

  % Add ports definitions to shape definition
  \expandafter\pgfutil@g@addto@macro\csname pgf@sh@s@box\endcsname{
    \__stridi_subdivide_segment:nNNNNN { north } { \portsnorth }
      { \southwest } { \northeast } { \northeast } { \northeast }
    \__stridi_subdivide_segment:nNNNNN { east } { \portseast }
      { \northeast } { \northeast } { \northeast } { \southwest }
    \__stridi_subdivide_segment:nNNNNN { south } { \portssouth }
      { \southwest } { \southwest } { \northeast } { \southwest }
```

```
    \__stridi_subdivide_segment:nNNNNN { west } { \portswest }
      { \southwest } { \northeast } { \southwest } { \southwest }
  }

}
```

(*End of definition for* box.)

/pgf/box Define style to draw boxes.

```
\ExplSyntaxOff
\tikzset{
  box/.default={},
  box/.style args={#1}{
    shape=box,
    draw,
    inner sep=.5em,
    minimum width=2em,
    minimum height=2em,
    execute at begin node=$,
    execute at end node=$,
    /pgf/box ports=#1,
  },
}
\ExplSyntaxOn
```

(*End of definition for* /pgf/box. *This function is documented on page* 1.)

/pgf/dot Define style to draw dots.

```
\ExplSyntaxOff
\tikzset{
  dot/.style={
    shape=circle,
    fill,
    inner sep=0,
    minimum width=0.4em,
  },
}
\ExplSyntaxOn
```

(*End of definition for* /pgf/dot. *This function is documented on page* 4.)

__stridi_clamp_to_node:n Moves the point in the internal pgf variables to the nearest point of the bounding rectangle of the given node, if it lies outside it.

```
\dim_new:N \l__stridi_x_dim
\dim_new:N \l__stridi_y_dim
\cs_new_protected:Nn \__stridi_clamp_to_node:n {
  \dim_set:Nn \l__stridi_x_dim { \pgf@x }
  \dim_set:Nn \l__stridi_y_dim { \pgf@y }
  \pgfpointanchor { #1 } { north~east }
  \dim_set:Nn \l__stridi_x_dim { \dim_min:nn { \l__stridi_x_dim } { \pgf@x } }
  \dim_set:Nn \l__stridi_y_dim { \dim_min:nn { \l__stridi_y_dim } { \pgf@y } }
```

```
  \pgfpointanchor { #1 } { south~west }
  \dim_set:Nn \l__stridi_x_dim { \dim_max:nn { \l__stridi_x_dim } { \pgf@x } }
  \dim_set:Nn \l__stridi_y_dim { \dim_max:nn { \l__stridi_y_dim } { \pgf@y } }
  \pgfpoint { \l__stridi_x_dim } { \l__stridi_y_dim }
}
```

(*End of definition for* `\__stridi_clamp_to_node:n`.)

`\__stridi_calctarget:nnnn` Performs the core calculation for

. The result is stored in the internal `pgf` variables. A wire from a corner to a side named

`\__stridi_anchors_to_proj:Nnnnn` in it, such as south west to south or west, ends offset along its diagonal. A bend that would end past a corner of the boundary ends at that corner.

`#1` : source node
`#2` : source anchor
`#3` : target node (can be boundary)
`#4` : target anchor

```
\cs_new_protected:Nn \__stridi_calctarget:nnnn {
  \pgfpointanchor{#1}{#2} \pgf@xa=\pgf@x \pgf@ya=\pgf@y
  \pgfpointanchor{#3}{#4} \pgf@xb=\pgf@x \pgf@yb=\pgf@y
  \regex_match_case:nn
  {
    { \A north ; north \  west \b } { \pgfpoint { \pgf@xa-.618*(\pgf@yb-\pgf@ya) } { \pgf@yb
    { \A north ; north \  east \b } { \pgfpoint { \pgf@xa+.618*(\pgf@yb-\pgf@ya) } { \pgf@yb
    { \A south ; south \  west \b } { \pgfpoint { \pgf@xa-.618*(\pgf@ya-\pgf@yb) } { \pgf@yb
    { \A south ; south \  east \b } { \pgfpoint { \pgf@xa+.618*(\pgf@ya-\pgf@yb) } { \pgf@yb
    { \A east  ; north \  east \b } { \pgfpoint { \pgf@xb } { \pgf@ya+.618*(\pgf@xb-\pgf@xa)
    { \A east  ; south \  east \b } { \pgfpoint { \pgf@xb } { \pgf@ya-.618*(\pgf@xb-\pgf@xa)
    { \A west  ; north \  west \b } { \pgfpoint { \pgf@xb } { \pgf@ya+.618*(\pgf@xa-\pgf@xb)
    { \A west  ; south \  west \b } { \pgfpoint { \pgf@xb } { \pgf@ya-.618*(\pgf@xa-\pgf@xb)
    { \A north ; north \b } { \pgfpoint { \pgf@xa } { \pgf@yb } }
    { \A north ; south \b } { \pgfpoint { \pgf@xa } { \pgf@yb } }
    { \A south ; north \b } { \pgfpoint { \pgf@xa } { \pgf@yb } }
    { \A south ; south \b } { \pgfpoint { \pgf@xa } { \pgf@yb } }
    { \A north ; east  \b } { \pgfpoint { \pgf@xa+.618*(\pgf@yb-\pgf@ya) } { \pgf@yb } }
    { \A south ; east  \b } { \pgfpoint { \pgf@xa-.618*(\pgf@yb-\pgf@ya) } { \pgf@yb } }
    { \A north ; west  \b } { \pgfpoint { \pgf@xa-.618*(\pgf@yb-\pgf@ya) } { \pgf@yb } }
    { \A south ; west  \b } { \pgfpoint { \pgf@xa+.618*(\pgf@yb-\pgf@ya) } { \pgf@yb } }
    { \A east  ; east  \b } { \pgfpoint { \pgf@xb } { \pgf@ya } }
    { \A east  ; west  \b } { \pgfpoint { \pgf@xb } { \pgf@ya } }
    { \A west  ; east  \b } { \pgfpoint { \pgf@xb } { \pgf@ya } }
    { \A west  ; west  \b } { \pgfpoint { \pgf@xb } { \pgf@ya } }
    { \A east  ; north \b } { \pgfpoint { \pgf@xb } { \pgf@ya+.618*(\pgf@xb-\pgf@xa) } }
    { \A west  ; north \b } { \pgfpoint { \pgf@xb } { \pgf@ya-.618*(\pgf@xb-\pgf@xa) } }
    { \A east  ; south \b } { \pgfpoint { \pgf@xb } { \pgf@ya-.618*(\pgf@xb-\pgf@xa) } }
    { \A west  ; south \b } { \pgfpoint { \pgf@xb } { \pgf@ya+.618*(\pgf@xb-\pgf@xa) } }
  } { #4 ; #2 }
  \__stridi_clamp_to_node:n { #3 }
}
```

`\l__stridi_angle_int` Scratch variable for `\__stridi_anchor_to_angle:Nnn`.

```
\int_new:N \l__stridi_angle_int
```

(*End of definition for* `\__stridi_calctarget:nnnn`, `\__stridi_anchors_to_proj:Nnnnn`, *and* `\l__stridi_angle_int`.)

`\__stridi_anchor_to_angle:Nnn` Calculates in/out angle given a node/anchor pair. Wires leave sides and their ports straight out, and corners diagonally. Accounts for when the node is the boundary. The result is stored in the given variable.

`#1` : token list variable
`#2` : node
`#3` : anchor

```
\cs_new_protected:Nn \__stridi_anchor_to_angle:Nnn {
  \regex_match_case:nnTF
    {
      { \A north \  east \b } { \int_set:Nn \l__stridi_angle_int {   45 } }
      { \A north \  west \b } { \int_set:Nn \l__stridi_angle_int {  135 } }
      { \A south \  east \b } { \int_set:Nn \l__stridi_angle_int {  -45 } }
      { \A south \  west \b } { \int_set:Nn \l__stridi_angle_int { -135 } }
      { \A north \b } { \int_set:Nn \l__stridi_angle_int {  90 } }
      { \A east  \b } { \int_set:Nn \l__stridi_angle_int {   0 } }
      { \A south \b } { \int_set:Nn \l__stridi_angle_int { -90 } }
      { \A west  \b } { \int_set:Nn \l__stridi_angle_int { 180 } }
    }
    { #3 }
    {
      \str_if_eq:nnTF { @ } { #2 }
        { \int_sub:Nn \l__stridi_angle_int { 180 } } { }
      \tl_set:Nf #1 { \int_use:N \l__stridi_angle_int }
    }
    { \tl_clear:N #1 }
}
```

(*End of definition for* `\__stridi_anchor_to_angle:Nnn`.)

`\__stridi_anchors_to_proj:Nnnnn` Calculates a target point given two node/anchor pairs. The first pair can be on the boundary. The result is stored in the given token list variable.

`#1` : token list variable
`#2` : source node
`#3` : source anchor
`#4` : target node
`#5` : target anchor

```
\cs_new_protected:Nn \__stridi_anchors_to_proj:Nnnnn {
  \regex_match_case:nn
    {
      { \A @ \. } {
        \__stridi_calctarget:nnnn { #4 } { #5 } { #2 } { #3 }
        \tl_set:Ne #1 { \dim_use:N \pgf@x, \dim_use:N \pgf@y }
      }
      { } { \tl_set:Nn #1 { #2.#3 } }
    }
```

```
    { #2.#3 }
}
```

(*End of definition for* `\__stridi_anchors_to_proj:Nnnnn`.)

`\l__stridi_oangle_tl`
`\l__stridi_iangle_tl`
`\l__stridi_source_tl`
`\l__stridi_target_tl`

Initialize scratch variables required by `\__stridi_draw_wire:nnnnn`.

```
\tl_new:N \l__stridi_oangle_tl
\tl_new:N \l__stridi_iangle_tl
\tl_new:N \l__stridi_source_tl
\tl_new:N \l__stridi_target_tl
```

(*End of definition for* `\l__stridi_oangle_tl` *and others.*)

`\__stridi_check_end:nnN`

Raises an error if a wire end has no angle to leave at, saying whether it has no anchor or an anchor off the sides.

`#1` : node
`#2` : anchor
`#3` : token list variable holding the angle

```
\msg_new:nnn { string-diagrams } { end-without-anchor }
  { Wire~end~‘#1’~has~no~anchor,~so~the~wire~is~not~drawn. }
\msg_new:nnn { string-diagrams } { end-off-sides }
  {
    Wire~end~‘#1.#2’~is~not~on~a~side,~so~the~wire~is~not~drawn:~
    wires~end~at~north,~east,~south~or~west,~or~at~ports.
  }
\cs_new_protected:Nn \__stridi_check_end:nnN {
  \tl_if_empty:NT #3
    {
      \tl_if_blank:nTF { #2 }
        { \msg_error:nnn { string-diagrams } { end-without-anchor } { #1 } }
        { \msg_error:nnnn { string-diagrams } { end-off-sides } { #1 } { #2 } }
    }
}
```

(*End of definition for* `\__stridi_check_end:nnN`.)

`\__stridi_draw_wire:nnnnn`

Draws a wire, unless an end has no angle to leave at: then it raises an error for each such end.

`#1` : Ti*k*Z keys
`#2` : source node
`#3` : source anchor
`#4` : target node
`#5` : target anchor

```
\cs_new_protected:Nn \__stridi_draw_wire:nnnnn {
  \__stridi_anchor_to_angle:Nnn   \l__stridi_oangle_tl { #2 } { #3 }
  \__stridi_anchor_to_angle:Nnn   \l__stridi_iangle_tl { #4 } { #5 }
  \__stridi_check_end:nnN { #2 } { #3 } \l__stridi_oangle_tl
  \__stridi_check_end:nnN { #4 } { #5 } \l__stridi_iangle_tl
  \bool_lazy_or:nnF
```

```
    { \tl_if_empty_p:N \l__stridi_oangle_tl }
    { \tl_if_empty_p:N \l__stridi_iangle_tl }
    {
      \__stridi_anchors_to_proj:Nnnnn \l__stridi_source_tl { #2 } { #3 } { #4 } { #5 }
      \__stridi_anchors_to_proj:Nnnnn \l__stridi_target_tl { #4 } { #5 } { #2 } { #3 }

      \path
        [
          out={\tl_use:N \l__stridi_oangle_tl},
          in= {\tl_use:N \l__stridi_iangle_tl},
          draw, #1,
        ]
          (\tl_use:N \l__stridi_source_tl) to
          (\tl_use:N \l__stridi_target_tl)
        ;
    }
}
```

(*End of definition for* `\__stridi_draw_wire:nnnnn`.)

`\__stridi_draw_boundary:`
`\__stridi_draw_boundary_around:nn`

Creates the boundary node around the picture so far, or around the origin if nothing has been drawn yet (as pgf does, it tells an empty picture by its untouched bounds). A picture gets a single boundary: later calls of `\wires` in it reuse the first one, found by the picture that pgf records for each node.

`#1` : north east corner to surround
`#2` : south west corner to surround

```
\cs_new_protected:Nn \__stridi_draw_boundary: {
  % NOTE: the space keeps docstrip from turning the __stridi of the name of node @ into the m
  \bool_lazy_and:nnF
    { \cs_if_exist_p:c { pgf@sh@pi@ @ } }
    { \str_if_eq_p:ee { \use:c { pgf@sh@pi@ @ } } { \pgfpictureid } }
    {
      \dim_compare:nNnTF { \pgf@picmaxx } = { -16000pt }
        { \__stridi_draw_boundary_around:nn { 0pt, 0pt } { 0pt, 0pt } }
        {
          \__stridi_draw_boundary_around:nn
            { current~bounding~box.north~east }
            { current~bounding~box.south~west }
        }
    }
}
\cs_new_protected:Nn \__stridi_draw_boundary_around:nn {
  \node [%draw, box=2/2/0/1, dashed, red,
    name=@, node~contents={},
    fit={
      ([shift={(+2em,+1em)}]#1)
      ([shift={(-2em,-1em)}]#2)
    },
  ];
}
```

(*End of definition for* `\__stridi_draw_boundary:` *and* `\__stridi_draw_boundary_around:nn`.)

\l__stridi_split_seq Scratch variable for __stridi_split_node_anchor:NNn.

```
\seq_new:N \l__stridi_split_seq
```

(*End of definition for* \l__stridi_split_seq.)

__stridi_split_node_anchor:NNn Parses a token list into a node/anchor pair.

#1 : token list variable to store node
#2 : token list variable to store anchor
#3 : token list to be parsed

```
\cs_new_protected:Nn \__stridi_split_node_anchor:NNn {
  \regex_extract_once:nnN { ([^\.]*) \.? (.*) } { #3 } \l__stridi_split_seq
  \seq_pop:NN \l__stridi_split_seq #1  % discard match
  \seq_pop:NN \l__stridi_split_seq #1  % keep 1st matching group (node)
  \seq_pop:NN \l__stridi_split_seq #2  % keep 2nd matching group (anchor)
}
```

(*End of definition for* __stridi_split_node_anchor:NNn.)

\exp_args:Nnnnoo This helper is required by __stridi_adj_map_function:nnN.

```
\exp_args_generate:n { nnnoo }
```

(*End of definition for* \exp_args:Nnnnoo.)

\l__stridi_target_node_tl
\l__stridi_target_anchor_tl Scratch variables for __stridi_adj_map_wire:nNnnn: the two parts of a target.

```
\tl_new:N \l__stridi_target_node_tl
\tl_new:N \l__stridi_target_anchor_tl
```

(*End of definition for* \l__stridi_target_node_tl *and* \l__stridi_target_anchor_tl.)

__stridi_adj_map_function:nnN Unrolls the nested dictionary of adjacencies into pairs and maps a function over them. The given list of Ti*k*Z keys is passed through to the function. Entries are read one by one, in order, so a node listed twice keeps the wires of both entries, and an anchor listed twice fans out.

#1 : Ti*k*Z keys to pass through
#2 : nested dictionary of adjacencies
#3 : function to be mapped over pairs

```
\msg_new:nnn { string-diagrams } { node-without-wires }
  { Node~‘#1’~has~no~wires:~list~them~as~in~#1~=~\{~north~=~b.south~\}. }
\cs_new_protected:Nn \__stridi_adj_map_function:nnN {
  \keyval_parse:nnn
    { \msg_error:nnn { string-diagrams } { node-without-wires } }
    { \__stridi_adj_map_node:nNnn { #1 } #3 }
    { #2 }
}
```

(*End of definition for* __stridi_adj_map_function:nnN.)

`\__stridi_adj_map_node:nNnn` Maps a function over the wires of one node.

`#1` : Ti*k*Z keys to pass through
`#2` : function to be mapped over pairs
`#3` : node
`#4` : dictionary of its wires

```
\msg_new:nnn { string-diagrams } { end-without-target }
  { Wire~end~‘#1.#2’~has~no~target:~give~it~one~as~in~#2~=~b.south. }
\cs_new_protected:Nn \__stridi_adj_map_node:nNnn {
  \keyval_parse:nnn
    { \msg_error:nnnn { string-diagrams } { end-without-target } { #3 } }
    { \__stridi_adj_map_wire:nNnnn { #1 } #2 { #3 } }
    { #4 }
}
```

(*End of definition for* `\__stridi_adj_map_node:nNnn`.)

`\__stridi_adj_map_wire:nNnnn` Applies a function to one wire, splitting its target into node and anchor.

`#1` : Ti*k*Z keys to pass through
`#2` : function to be applied
`#3` : source node
`#4` : source anchor
`#5` : target

```
\cs_new_protected:Nn \__stridi_adj_map_wire:nNnnn {
  \__stridi_split_node_anchor:NNn \l__stridi_target_node_tl \l__stridi_target_anchor_tl { #5
  \exp_args:Nnnnoo #2
    { #1 } { #3 } { #4 } { \l__stridi_target_node_tl } { \l__stridi_target_anchor_tl }
}
```

(*End of definition for* `\__stridi_adj_map_wire:nNnnn`.)

`\wires` Define our superstar.

```
\NewDocumentCommand{\wires}{ O{} m }
{
  \__stridi_draw_boundary:
  \__stridi_adj_map_function:nnN
    { #1 } { #2 } \__stridi_draw_wire:nnnnn
}
```

(*End of definition for* `\wires`. *This function is documented on page* 3.)

Close the DocStrip guards and call it a day.

```
⟨/package⟩
```

Change History

0.1.0
General: initial version 1
0.2.0
General: make box ports configurable . 1
/pgf/box: acts as a shortcut for setting port counts 8
box: make ports configurable through Ti*k*Z keys 7
0.2.1
\wires: now correctly handles optional style parameter 14
0.3.0
General: replace loose ends with wires to the boundary node @ 1
/pgf/box: without counts, keeps the ones in force 8
/pgf/box␣ports: counts start at zero, and the shortcut checks that it gets four 5
box: ports a box doesn't have are an error 7
\wires: every listed wire is drawn, so nodes can be listed twice and anchors can fan out 14
loose ends are wires to the boundary node @, replacing the third argument 14
wire ends off the sides or without an anchor are errors 14
wires leave corners diagonally 14

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

\. 215, 291
/pgf/box *1*, 121
/pgf/box␣ports *2*, 11
/pgf/box␣ports␣east *2*, 11
/pgf/box␣ports␣north *2*, 11
/pgf/box␣ports␣south *2*, 11
/pgf/box␣ports␣west *2*, 11
/pgf/dot *4*, 136
\{ 300
\} 300
\␣ 164, 165, 166, 167, 168, 169, 170, 171, 195, 196, 197, 198

A

\A 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 195, 196, 197, 198, 199, 200, 201, 202, 215

B

\b 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 195, 196, 197, 198, 199, 200, 201, 202
bool commands:
\bool_lazy_and:nnTF 267
\bool_lazy_or:nnTF 247
box 82

C

clist commands:
\clist_map_inline:nn 86
\coordinate *3*
cs commands:
\cs_if_exist:NTF 49, 60
\cs_if_exist_p:N 268
\cs_new_nopar:Npn 53
\cs_new_protected:Nn 40, 47, 148, 159, 192, 212, 234, 242, 265, 280, 290, 301, 309, 315
\csname 109

D

dim commands:
\dim_compare:nNnTF 271
\dim_max:nn 155, 156
\dim_min:nn 152, 153

\dim_new:N 146, 147
\dim_set:Nn 149, 150, 152, 153, 155, 156
\dim_use:N 217

E

\endcsname 109
exp commands:
\exp_args:Ne 28
\exp_args:Nnnnoo 296, 317
\exp_args_generate:n 296
\expandafter 109
\ExplSyntaxOff 121, 136
\ExplSyntaxOn 135, 145

I

\inheritanchor 94
\inheritanchorborder 95
\inheritbackgroundpath 96
\inheritsavedanchors 85
int commands:
\int_compare:nNnTF 26, 56
\int_new:N 191
\int_set:Nn 195, 196, 197, 198, 199, 200, 201, 202
\int_step_inline:nnnn 48
\int_sub:Nn 207
\int_use:N 208

K

keyval commands:
\keyval_parse:nnn 302, 310

M

msg commands:
\msg_error:nnn 36, 238, 303
\msg_error:nnnn 239, 311
\msg_error:nnnnnn 58
\msg_new:nnn 11, 45, 227, 229, 299, 307

N

\NewDocumentCommand 320
\node 281
\northeast 111, 113, 115, 117

P

\path 254
\pgfdeclareshape 82
\pgfextractx 41
\pgfextracty 42
\pgfkeys 17
\pgfkeysalso 28
\pgfkeysvalueof 100, 102, 104, 106
\pgfmathdivide 71
\pgfmathresult 75
\pgfmathtruncatemacro . 100, 102, 104, 106
\pgfpictureid 269
\pgfpoint 43, 157, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187
\pgfpointanchor 151, 154, 160, 161
\pgfpointlineattime 65, 74
\pgfreferencednodename 60, 61
\portseast 101, 102, 112
\portsnorth 99, 100, 110
\portssouth 103, 104, 114
\portswest 105, 106, 116
prg commands:
\prg_break: 51
\prg_do_nothing: 52
\ProvidesExplPackage 6

R

regex commands:
\regex_extract_once:nnN 291
\regex_match_case:nn 162, 213
\regex_match_case:nnTF 193
\RequirePackage 3, 4

S

\savedmacro 99, 101, 103, 105
seq commands:
\seq_count:N 26
\seq_item:Nn 30, 31, 32, 33
\seq_new:N 16, 289
\seq_pop:NN 292, 293, 294
\seq_set_split:Nnn 25
\southwest 111, 113, 115, 117
str commands:
\str_if_eq:nnTF 206
\str_if_eq_p:nn 269
stridi internal commands:
__stridi_adj_map_function:nnN *13*, 299, 301, 323
__stridi_adj_map_node:nNnn 304, 307, 309
__stridi_adj_map_wire:nNnnn *13*, 312, 315, 315
__stridi_anchor_to_angle:Nnn *9*, 192, 192, 243, 244
__stridi_anchors_to_proj:Nnnnn 159, 212, 212, 251, 252
\l__stridi_angle_int 191, 195, 196, 197, 198, 199, 200, 201, 202, 207, 208
__stridi_calctarget:nnnn 159, 159, 216
__stridi_check_end:nnN 227, 234, 245, 246

__stridi_clamp_to_node:n 146, 148, 189
__stridi_draw_boundary: 265, 265, 322
__stridi_draw_boundary_around:nn 265, 272, 274, 280
__stridi_draw_wire:nnnnn *11*, 242, 242, 324
\l__stridi_iangle_tl 223, 244, 246, 249, 257
__stridi_intersect_hv_lines_through:NN . . 40, 40, 67, 68, 76, 77
\l__stridi_oangle_tl 223, 243, 245, 248, 256
\l__stridi_port_counts_seq 16, 25, 26, 30, 31, 32, 33
\l__stridi_source_tl . . 223, 251, 260
__stridi_split_node_anchor:NNn *13*, 290, 290, 316
\l__stridi_split_seq 289, 291, 292, 293, 294
__stridi_subdivide_segment:nNNNNN 45, 47, 110, 112, 114, 116
\l__stridi_target_anchor_tl 297, 316, 318
\l__stridi_target_node_tl 297, 316, 318
\l__stridi_target_tl . . 223, 252, 261
\l__stridi_x_dim 146, 149, 152, 155, 157
\l__stridi_y_dim 147, 150, 153, 156, 157

T

TEX and LATEX 2ε commands:
\pgf@picmaxx 271
\pgf@x 149, 152, 155, 160, 161, 217
\pgf@xa 41, 43, 160, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 184, 185, 186, 187
\pgf@xb . . . 161, 168, 169, 170, 171, 180, 181, 182, 183, 184, 185, 186, 187
\pgf@y 150, 153, 156, 160, 161, 217
\pgf@ya 42, 43, 160, 164, 165, 166, 167, 168, 169, 170, 171, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187
\pgf@yb . . . 161, 164, 165, 166, 167, 172, 173, 174, 175, 176, 177, 178, 179
\pgfutil@g@addto@macro 109
\tikzset 122, 137
tl commands:
\tl_clear:N 210
\tl_if_blank:nTF 23, 237
\tl_if_empty:NTF 235
\tl_if_empty_p:N 248, 249
\tl_new:N . . 223, 224, 225, 226, 297, 298
\tl_set:Nn 208, 217, 219
\tl_use:N 256, 257, 260, 261

U

use commands:
\use:N 269
\usetikzlibrary *1*, 5

W

\wires *3–5*, *12*, 320