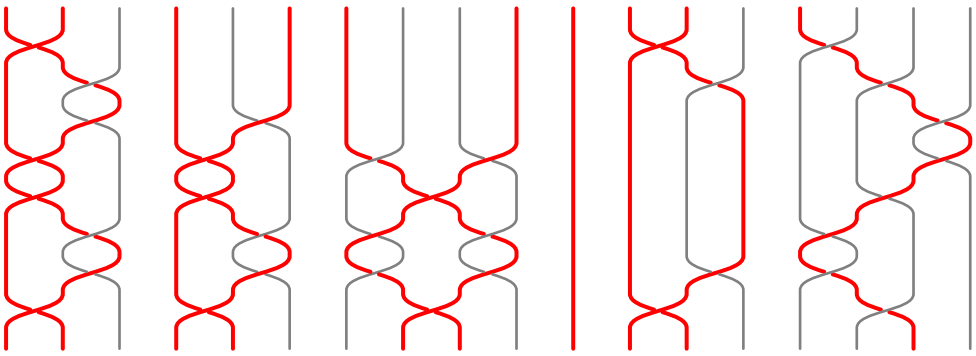
The braids Package: Documentation

Andrew Stacey
loopspace@mathforge.org

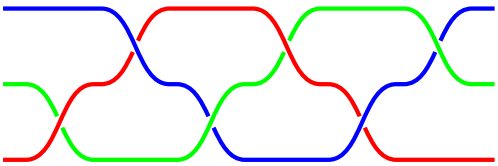
v2.2 from 2022/10/26



1 Introduction

This is a package for drawing braid diagrams using PGF/TikZ. An example follows.

```
\begin{center}
\begin{tikzpicture}
\pic[
  rotate=90,
  braid/.cd,
  every strand/.style={ultra thick},
  strand 1/.style={red},
  strand 2/.style={green},
  strand 3/.style={blue},
] {braid={s_1 s_2^{-1} s_1 s_2^{-1} s_1 s_2^{-1}}};
\end{tikzpicture}
\end{center}
```



2 TikZ Library Usage

Version 2.0 changed the implementation to use the TikZ `pic` syntax. It also converted it to a TikZ library, so to use it put the following in the preamble.

```
\usetikzlibrary{braids}
```

(Or add it to the copious list of TikZ libraries that you are already using.)

`braid` A braid is specified by the pic name `braid`. The usual syntax for this is as follows:

```
\pic[options] at (coordinate) {braid={braid-word}};
```

`braid-word` The `braid-word` is an expression in the braid group, such as `s_1 s_2^{-1} s_{3,5}`. The generator labels are not significant.

The subscript determines how the strands cross, as follows:

1. If the subscript is a single number, as in `s_2`, the crossing goes from that number over the next.
2. If the subscript is two or more numbers separated by a comma, as in `s_{2,4}` or `s_{1,3,5}`, imagine holding the numbered strands and moving each to the position of the next with the last strand moving under the others to where the first stands. Any intermediate non-specified strand is behind all those involved in the crossing.
3. If the subscript is hyphenated, as in `s_{1-5}`, this is equivalent to `s_{1,2,3,4,5}`. This can be used as part of a list, as in `s_{1-3,5}`.

All of these crossings take place in the same amount of vertical space. Be advised that crossings involving a lot of strands can get quite squashed.

The exponent can be `1`, `{-1}`, or missing (in which case it defaults to `1`, note also that the exponent is read as a TEX-token so `{1}` is also legal). If the exponent is -1 then the braid element represented by the crossing is inverted.

- `s_1` is strand 1 over strand 2.
- `s_1^{-1}` is strand 2 over strand 1.
- `s_{1,3}` is strand 1 over strand 3, and both are over strand 2.
- `s_{1-3}` is strand 3 under strand 2 and then under strand 1.

Certain other symbols are allowed in the `braid-word` which control the rendering of the braid. These extras are as follows.

1. To get crossings to render at the same height, separate them with a hyphen (note: no check is made to ensure that the crossings can legally be put at the same height; *caveat emptor*). For example, `s_1-s_3`.
2. To draw a *floor* – which is a rectangle behind the braid occupying some number of levels (the default being one level) – precede the braid element by a vertical line, as in `s_1 | s_2`. The floor is itself a pic which, by default, consists of a rectangle and two horizontal lines. The rectangle picks up any `fill` options and the lines any `draw` options that are set in the

`braid/every floor` and `braid/floor <n>` styles. The `n` is the level number, starting at 1.

More general floors can be drawn using the key `braid/add floor`. This takes one argument which is a comma separated list of parameters that specifies the position and size floor:

```
braid/add floor={x,y,width,height,name}
```

The units used are the "natural" units of the braid: strand separation and level height. The `name` is optional and if given can be used to style the floor in that the style `braid/floor <name>` is applied to that floor.

Replacing the floor pic will change how it is drawn. The coordinate system is set up for the floor pic so that the floor is a unit square with lower left corner at the origin.

3. The identity element can occur in the braid-word. It is represented by `1`. This inserts the identity which corresponds to no crossing. However, it takes the same amount of space as if there were a crossing.

2.1 Style Options

There are various keys that change the behaviour or rendering of the braid. All of these are in the `/tikz/braid/` namespace.

`number of strands` The key `number of strands` sets the minimum number of strands for the braid. The number of strands will grow according to the terms in the braid word so this merely sets a lower bound. If not set, the number of strands will be determined by the terms in the braid word.

`height` The key `height` sets the height of the piece of the braid corresponding to an element in the group. This can be negative.

`width` The key `width` sets the separation of the strands in the braid. This can be negative.

`border height` The key `border height` adds a little extra length to the strands at the start and end of the braid.

`gap` The key `gap` is used to determine how much of a gap to leave in the under strand at a crossing. This should be a number strictly between 0 and .5. The curve is drawn using a cubic bezier and the gap is in terms of the time parameter, so the gap will not increase exactly proportionally to the value given by this key, though that is a reasonable approximation.

`control factor` As just said, the parts of the strands involved in a crossing are drawn using a cubic bezier curve. The control points are vertically above or below their respective end point. This key determines that vertical separation. It is multiplied by the `height` so that it scales properly. It can be set to 0 whereupon the strands in the crossing are straight lines. The default is 0.5.

`nudge factor` The crossings are not quite placed one after another. There is a small "nudge" between the end of one crossing and the start of another. Due to the way that the strands are lengthened, if there is no "nudge" then some PDF renderers produce slightly strange results at certain magnifications. This key controls how much that "nudge" is, as a factor of the `height`. For the aforementioned reason, it should not be set to 0 (the default is 0.05). Note that this does not change the height of a crossing. Rather, it nudges the height at which the strands start to cross over.

`every strand, strand <n>` The style of the strands are controlled by two types of option. Style options that are set on the `pic` are passed to every strand. It is also possible to add style options to individual strands using the keys `every strand` and `strand <n>`. The strands are numbered by their starting position.

`every floor, floor <n>` When a floor is requested behind a crossing, it is rendered as a pic. These keys control how the floor is styled.

2.2 Coordinates and Anchors

The braid is littered with coordinates. Each strand gets a coordinate at each end, and at every level between crossings. These are labelled and numbered by the initial strand position and the crossing level. They are also labelled and numbered by the final strand position with the prefix `rev`. With a `name prefix`, the coordinate names look like the following:

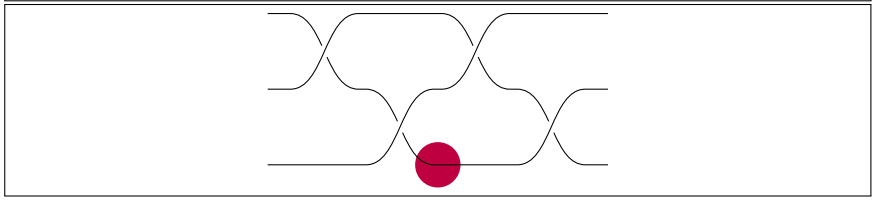
```
<name prefix>-<strand number>-<crossing number>
<name prefix>-rev-<strand number>-<crossing number>
```

The crossing number can also be either `s` or `e` for the start and end of the strand. Note that `-1-0` and `-1-s` are slightly different in that `s` includes the border height.

In addition, if the braid is named then a rectangular node is defined that fits around the whole braid. This node is not actually rendered but its anchors can be used afterwards as if it had been.

`anchor` The key `anchor` (in the `braid` name space) can be used to shift the braid so that a different part of it is at the specified location. If the argument contains a hyphen then it is assumed to refer to a point on a strand with the same interpretation as the coordinates (except without the `<name prefix>-`, of course). If the argument does not contain a hyphen then it is taken to be an anchor of a surrounding rectangular node (the node might not itself exist – if the braid isn't named – but that doesn't affect the positioning). In the following example, the braid is shifted so that where the third strand starts the second level is at the position `(1,1)`

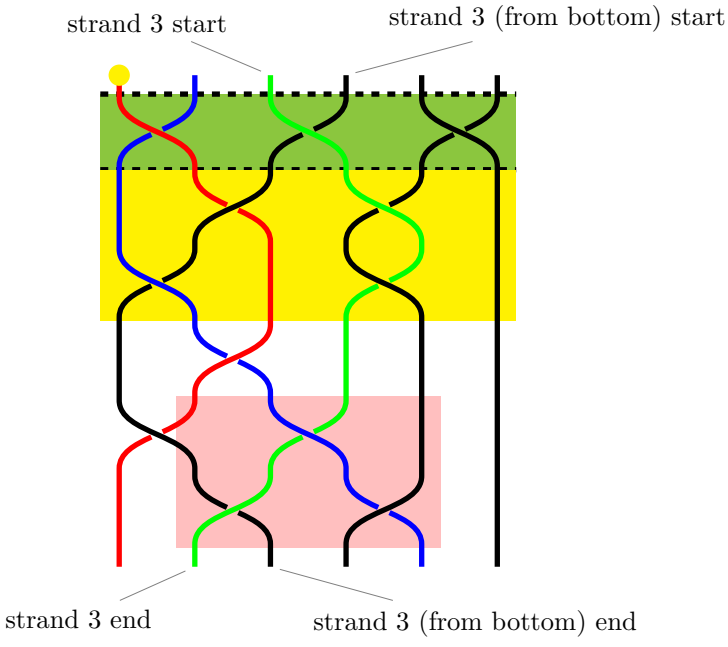
```
\begin{center}
\begin{tikzpicture}
\fill[purple] (1,1) circle[radius=3mm];
\pic[braid/anchor=3-2,rotate=90] at (1,1) {braid={s_2 s_1 s_2
    s_1}};
\end{tikzpicture}
\end{center}
```



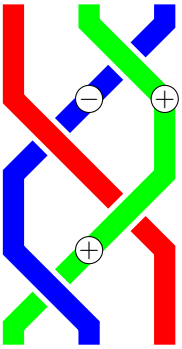
3 Examples

Here are more detailed examples.

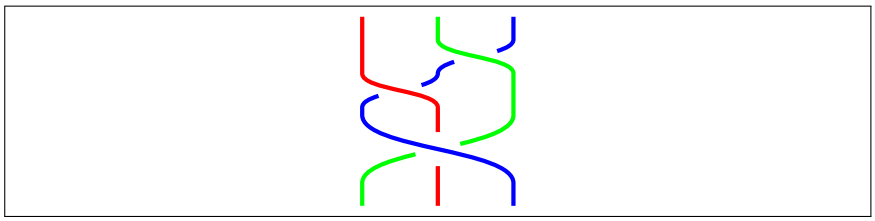
```
\begin{center}
\begin{tikzpicture}
\pic[
  braid/every floor/.style={fill=yellow},
  braid/floor 1/.style={draw=black,dashed,fill=yellow!50!green},
  line width=2pt,
  braid/strand 1/.style={red},
  braid/strand 2/.style={blue},
  braid/strand 3/.style={green},
  braid/add floor={2,4,3,2,a},
  braid/floor a/.style={fill=pink},
  name prefix=braid,
] at (2,0) {braid={| s_1-s_3-s_5 | s_2^{-1}-s_4| s_1-s_4 s_2^{-1}
    s_1-s_3 s_2^{-1}-s_4^{-1}}};
\fill[yellow] (2,0) circle (4pt);
\node[at=(braid-3-s),pin=north west:strand 3 start] {};
\node[at=(braid-3-e),pin=south west:strand 3 end] {};
\node[at=(braid-rev-3-s),pin=north east:strand 3 (from bottom)
    start] {};
\node[at=(braid-rev-3-e),pin=south east:strand 3 (from bottom)
    end] {};
\end{tikzpicture}
\end{center}
```



```
\begin{center}
\begin{tikzpicture}
\pic[
  braid/.cd,
  number of strands=3,
  line width=8pt,
  strand 1/.style={red},
  strand 2/.style={green},
  strand 3/.style={blue},
  gap=0.1,
  control factor=0,
  nudge factor=0,
  name prefix=braid,
] {braid={a_2 a_1 a_2^{-1} a_1}};
\node[circle,draw,fill=white,inner sep=0pt] at (braid-2-1) {\(+\)};
\node[circle,draw,fill=white,inner sep=0pt] at (braid-3-1) {\(-\)};
\node[circle,draw,fill=white,inner sep=0pt] at (braid-2-3) {\(+\)};
\end{tikzpicture}
\end{center}
```



```
\begin{center}
\begin{tikzpicture}
\pic[
  braid/.cd,
  number of strands=3,
  ultra thick,
  strand 1/.style={red},
  strand 2/.style={green},
  strand 3/.style={blue},
  gap=0.1,
%  control factor=0,
%  nudge factor=0,
  name prefix=braid,
] {braid={a_{1-3} a_{1,3}}};
\end{tikzpicture}
\end{center}
```



4 Original Package Usage (Depreciated)

The original version was as a separate package. This is still included for backwards compatibility. To use this package, you need to put the following in the preamble.

```
\usepackage{braids}
```

`\braid` A braid is specified by the command `\braid`. The syntax for this command is as follows:

```
\braid[style options] (name) at (coordinate) braid-word;
```

`braid-word` The `braid-word` is an expression in the braid group, such as `s_1 s_2^{-1}`. The generator labels are not significant. The exponent can be `1`, `{-1}`, or missing (in which case it defaults to `1`, note also that the exponent is read as a TEX-token so `{1}` is also legal). Certain other symbols are allowed in the `braid-word` which control the rendering of the braid. These extras are as follows.

1. To get crossings to render at the same height, separate them with a hyphen (note: no check is made to ensure that the crossings can legally be put at the same height; *caveat emptor*).

2. To draw a *floor*, precede the braid element by a vertical line. What happens then is that when the braid is rendered, the coordinates of the rectangle behind that crossing (wide enough to encompass all the strands) is passed to a command. The intention is that this command draw something behind the braid. The command is configurable by a key (see 4.1).

3. The identity element can occur in the braid-word. It is represented by `1`. This inserts the identity which corresponds to no crossing. However, it takes the same amount of space as if there were a crossing.

4. Strands can be labelled between crossings. To do this, the commands `\label`, `\olabel`, and `\clabel` are provided. These take three arguments, the first is optional. The result of this command is to place a node on top of a particular strand between the crossings where the command is given. The first (optional) argument can be used to pass style options to this node. The second argument is the strand number. The third argument is the label text.

   The three commands differ as to how they interpret the strand number. For `\olabel`, the strand number is taken to mean the strand that starts at that position. For `\clabel`, the strand number is taken to mean the strand that is currently at that position. The behaviour of `\label` is to choose one or other of these depending on whether the key `strand label by origin` is true or false. This key only has an effect at the start of the braid-word; it cannot be reset in the middle.

5. Style options can be given in the middle of a braid-word by enclosing them in square brackets. There are not many style options that it makes sense to change in the middle of the braid-word, since the strands are rendered all in one go at the end.

6. Scoping is handled by using braces. Thus to change a style only briefly, enclose the desired scope in braces.

`name` The (optional) `name` acts a little like the `name` of a TikZ node. When it is specified, the routine that renders the braid also saves certain coordinates as if they were node anchors. Specifically, `coordinate` nodes are placed at the centre of the braid diagram and at the ends of each strand. The centre has the label `name`, the strands are labelled `name-number-end` and `name-rev-number-end`, where `name` is the name given to the braid, `number` is the number of the strand counting from the left, and `end` is either `s` for the start or `e` for the end. If the version with `rev` is used then the numbers correspond to the *final* positions of the braids. The name can also be specified with the `name` key.

`at` The (optional) `at (coordinate)` syntax positions the braid at the `coordinate` in the current picture. Due to the implementation, the coordinate has to be known at the start, but the width and height of the braid are only known at the end. Therefore, the braid is positioned so that the start of the first strand is at `(coordinate)`. This can also be specified using the `at` key.

`style options` The `style options` set the style for the braid strands. They can be grouped into three types: options that set up the main parameters for the braid, options that set the default style for the strands, and options that set up styles for individual strands. The options are as follows.

4.1 Style Options

`number of strands` The key `number of strands` sets the minimum number of strands for the braid. The number of strands will grow according to the terms in the braid word so this merely sets a lower bound. If not set, the number of strands will be determined by the terms in the braid word.

`height` The key `height` sets the height of the piece of the braid corresponding to an element in the group.

`width` The key `width` sets the separation of the strands in the braid.

`border height` The key `border height` adds a little extra length to the strands at the start and end of the braid.

`gap` The key `gap` is used to determine how much of a gap to leave in the under strand at a crossing. This should be a number strictly between 0 and .5. The curve is drawn using a cubic bezier and the gap is in terms of the time parameter, so the gap will not increase exactly proportionally to the value given by this key, though that is a reasonable approximation.

`control factor` As just said, the parts of the strands involved in a crossing are drawn using a cubic bezier curve. The control points are vertically above or below their respective end point. This key determines that vertical separation. It is multiplied by the `height` so that it scales properly. It can be set to 0 whereupon the strands in the crossing are straight lines. The default is 0.5.

`nudge factor` The crossings are not quite placed one after another. There is a small "nudge" between the end of one crossing and the start of another. Due to the way that the strands are lengthened, if there is no "nudge" then some PDF renderers produce slightly strange results at certain magnifications. This key controls how much that "nudge" is, as a factor of the `height`. For the aforementioned reason, it should not be set to 0 (the default is 0.05). Note that this does not change the height of a crossing. Rather, it nudges the height at which the strands start to cross over.

`style strands` The style of the strands are controlled by two types of option. Style options that are set on the `\braid` command are passed to every strand. It is also possible to add style options to individual strands using the key `style strands`. This

takes two options, a comma-delimited list of strand numbers (which could be just a single number) and a list of options to be applied to that strand. Thus, the syntax is `style strands={n,m,...}{options}`. The strands are numbered by their starting position. Not all of the standard TikZ style options are possible due to the way that the strands are constructed. Basically, the options that are allowed are those that do not require changing the path or drawing it more than once.

`floor command` When a floor is requested behind a crossing, the actual way to render it is determined by a command. This key allows the user to define that command. The argument to this key should be the code that should be executed for each floor. To avoid the hassle of getting the number of hashes right, the command should take no arguments. Rather, the coordinates of the rectangle are saved in to macros `\floorsx`, `\floorsy`, `\floorex`, `\floorey` (these macros will expand to something like `10pt`) and the command should use these to position the drawing. The default is to draw a line at the top and at the bottom of the rectangle.

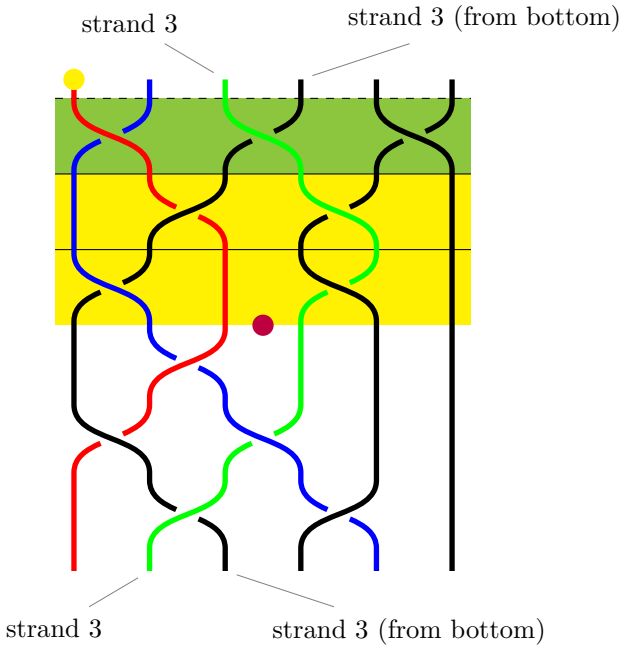
`style floors` `style all floors` In the spirit of separating *style* and *content*, the style options for the floors can be specified separately to the command (of course, they could be built in to the command). One advantage of this over building them in to the command is to allow them to be overridden for individual floors. The `style all floors` sets up options to be used for *all* floors, whilst the `style floors={n,m,...}{options}` sets up options to be used only for the listed floor. Anything specified in the `floor command` will take precedence over both of these.

Any other style options are passed to the underlying TikZ/PGF system and so may influence how the braid is drawn (but note that not all keys make sense due to the implementation).

5 Examples

Here are more detailed examples.

```
\begin{center}
\begin{tikzpicture}
\braid[
  style all floors={fill=yellow},
  style floors={1}{dashed,fill=yellow!50!green},
  floor command={%
   \fill (\floorsx,\floorsy) rectangle (\floorex,\floorey);
   \draw (\floorsx,\floorsy) -- (\floorex,\floorsy);
  },
  line width=2pt,
  style strands={1}{red},
  style strands={2}{blue},
  style strands={3}{green}
] (braid) at (2,0) | s_1-s_3-s_5 | s_2^{-1}-s_4| s_1-s_4 s_2^{-1}
    s_1-s_3 s_2^{-1}-s_4^{-1};
\fill[yellow] (2,0) circle (4pt);
\fill[purple] (braid) circle (4pt);
\node[at=(braid-3-s),pin=north west:strand 3] {};
\node[at=(braid-3-e),pin=south west:strand 3] {};
\node[at=(braid-rev-3-s),pin=north east:strand 3 (from bottom)] {};
\node[at=(braid-rev-3-e),pin=south east:strand 3 (from bottom)] {};
\end{tikzpicture}
\end{center}
```



```
\begin{center}
\begin{tikzpicture}
\braid[
  number of strands=3,
  line width=8pt,
  style strands={1}{red},
  style strands={2}{green},
  style strands={3}{blue},
  gap=0.1,
  control factor=0,
  nudge factor=0,
  strand label by origin=true,
  strand label/.style={circle,draw,fill=white,inner sep=0pt},
  yscale=1] (braid_1) a_2 \label{2}{\(+\)} \clabel{2}{\(-\)} a_1
    a_2^{-1} \olabel{2}{\(+\)} a_1;
\end{tikzpicture}
\end{center}
```