

The `hawkdraw` package

A LaTeX package to generate vector graphic output based on l3draw and related LaTeX3 modules using a simple syntax

Jasper Habicht *

Version 0.11.2, released on 13 November 2026

---

1 Introduction

My sharp eyes spotted you, ground creature, already when you were still looking for this manual. Let me introduce myself: I am Victor, the drawing hawk. I especially like drawing with LaTeX and since I value precision, using `l3draw` in combination with `l3fp` is my favourite way to draw. I created the `hawkdraw` package to make it easier for you to use the powerful drawing capabilities of the LaTeX3 kernel.

The `hawkdraw` package allows the user to use a simple syntax similar to Ti*k*Z to generate vector graphic output based on `l3draw` and related LaTeX3 modules such as `l3fp` which provides very precise calculations. The syntax is meant to be simple and flexible but also coherent and logical with the aim of allowing the user to quickly draw graphics. The package name relates to the quickness and agility of hawks and the preciseness of their eyesight.

The package intends to define a set of user-level commands that can be used for drawing simple graphics. It does not aim to be a substitute for the Ti*k*Z package or to offer a similar scope of application.

The package is currently still in an experimental stage. The code may be subject to fundamental and frequent changes. The author is grateful for reporting any bugs via GitHub at https://github.com/jasperhabicht/hawkdraw/issues. A site for asking questions about the package and for suggestions for improvement is available at https://github.com/jasperhabicht/hawkdraw/discussions.

! Note that the underlying code of the `l3draw` package which is about to be included as module to the LaTeX3 kernel is experimental as well and hence might change possibly breaking the functionality of this package. It is highly recommended to use it with an up-to-date TeX distribution, especially with the most recent version of the `l3draw` package.

* E-mail: mail@jasperhabicht.de. I am grateful to Joseph Wright, Max Chernoff, Clea F. Rees, Mikael Persson Sundqvist and Jonathan P. Spratte who helped me improving the code and the math behind it. Victor, the drawing hawk: © 2026 Hannah Klöber.

2 Contents

1 Introduction 1

2 Contents 2

3 Loading the package 3

4 General remarks 3

5 Main user environment 4
- 5.1 hawkdraw environment . . . 4
- 5.2 Setting baseline, layers and bounding box . . . 4

6 Scopes and layers 5

7 Path syntax 6
- 7.1 Path command and options . . . 7
- 7.2 Points on the path . . . 7
- 7.3 Straight lines . . . 8
- 7.4 Bézier curves . . . 9
- 7.5 Closing paths . . . 9
- 7.6 Circles, ellipses, arcs, rectangles and grids . . . 9
- 7.7 Plot functions . . . 13
  - 7.7.1 sharp plot function . . . 13
  - 7.7.2 smooth plot function . . . 13
- 7.8 Points and nodes . . . 14
  - 7.8.1 Points . . . 14
  - 7.8.2 Nodes and frames . . . 21

8 Mapping operation 26

9 Path options 28
- 9.1 Stroke and color fill, opacity, clipping . . . 28
- 9.2 Line styling . . . 29
- 9.3 Rounded corners and fill rule . . . 31
- 9.4 Transformations . . . 31

10 Path manipulations 33
- 10.1 Shortening paths . . . 33
- 10.2 Offsetting paths . . . 33
- 10.3 Softpath-related settings . . . 34

11 Defining custom styles and colors 34

12 Arrow tips 35
- 12.1 Arrow tips at path ends . . . 35
- 12.2 Arrow tips on paths . . . 36

13 Patterns 37

14 Opacity and shadings 39
- 14.1 Opacity groups . . . 39
- 14.2 Soft masks . . . 40
- 14.3 Blend modes . . . 41
- 14.4 Linear, radial and conic shadings . . . 43

15 Tagging support 44

16 Bidirectionality support 45

17 Use with ConTeXt 45

18 Changes 46

3 Loading the package

To install the package, copy the package file `hawkdraw.sty` together with the module files ending with `.code.tex` into the working directory or into the `texmf` directory. After the package has been installed, the `hawkdraw` package is loaded by calling `\usepackage{hawkdraw}` in the preamble of the document.

The package can be used with PDFLaTeX, LuaLaTeX or XeLaTeX. The package does not load any dependencies, but it needs a LaTeX kernel of 1 June 2022 or newer. The package requires an up-to-date TeX distribution with the latest version of the LaTeX3 kernel.

! The `hawkdraw` package supports PDFLaTeX and LuaLaTeX in full. Most of the basic commands should work with other compilation engines as well. Blend modes and soft masks, however, are not supported with LaTeX (via DVI) or XeLaTeX.

4 General remarks

Similar to Ti*k*Z, the `hawkdraw` package makes use of key-value pairs to set options for commands. Some options are available for almost all commands, other have very specific uses. In the descriptions of the options, this manual states which data type the relevant option assumes. The following data types exist:

Data type	Explanation
`boolean`	A boolean only allows the values `true` or `false`. If not stated otherwise, setting the option without a value is equivalent with setting the value to `true`.
`string`	A string is a concatenation of letters or numbers of any length. Typically, strings do not contain symbols. For example, strings are used for names of points, frames of nodes, patterns or arrow tips.
`float`	A floating-point number is a number without or with unit that uses a dot as decimal separator.
`dimension`	A dimension or length is a typical TeX length consisting of a floating point number and a length unit.
`tuple`	A tuple consists of two floating-point numbers without or with unit that are separated by a comma. A tuple may be enclosed by parenthesis.
`point`	A point is essentially a tuple, but it can also be a reference to a named node or, using a special syntax, denote a point in polar coordinates (see below).
`clist`	A comma-separated list is a list of strings that are separated by commas. Such a list can consist of no or only one item as well. If the relevant list can only have a maximum number of items, this is stated in the description of the relevant option. Note that a comma-separated list is typically not surrounded by parenthesis.
`options`	A key-value list consists of comma separated entities of options that may be followed by an equals sign and a value. A value can be another key-value list.

`color`	This means a color definition as specified by the `l3color` module. Note that colors defined by the `xcolor` package have no effect when using the `hawkdraw` package.
`code`	TeX code, typically with certain specifications as stated in the description of the relevant option.
none	Some options do not accept any data type as they are only executing some function.

Options in the `hawkdraw` package are divided into sets. Depending on where the options are used in the document, only specific sets can be made use of. If a command or environment accepts specific sets of options, the relevant prefix should be omitted.

```
\hawkdrawset{‹options›}
```

The command `\hawkdrawset` is used to set options. It can be used inside or outside of `hawkdraw` environments. It uses the `path/` set of options. If options of other sets should be set, they need to be set from the root level. For example, options contained in the `picture/` set can be set using `\hawkdrawset{picture/<option>}`.

5 Main user environment

5.1 `hawkdraw` environment

The package has only very few main user commands that can be used almost everywhere in the document body. Among these, the `hawkdraw` environment is the most important as it is used to define a drawing and most other commands are only defined inside this environment.

```
\begin{hawkdraw}[‹options›]
\end{hawkdraw}
```

The `hawkdraw` environment is the main environment for drawing. It defines most other commands described in the following sections. It can take an optional argument for options to the environment, which are explained in the following, or to paths which then apply to all paths inside the environment. As for options, the `hawkdraw` environment accepts the `picture/` set, the `path/` set and the `tag/` set.

5.2 Setting baseline, layers and bounding box

```
picture/baseline={‹dimension›}
```

The option `baseline` sets the baseline for the environment as dimension. The default baseline is typically the bottom of the the bounding box of the drawing.

foo bar baz

```
foo
\begin{hawkdraw}[baseline={opt}]
  \node[stroke, anchor={H,l}]{bar}
\end{hawkdraw}
baz
```

```
picture/layers={‹clist›}
```

The option `layers` sets the layers for the environment as a comma-separated list. The layer `main` must always be present and is the default layer. Layers are drawn in the order they are defined (see below).

```
picture/bounding box={auto}
picture/bounding box={zero}
picture/bounding box={‹clist›}
picture/overlay
```

The option `bounding box` sets the environment to have a custom bounding box. It can take the value `auto` which sets the bounding box to the default bounding box of the drawing. It can take the value `zero` to set a zero-sized bounding box. It can also take as value a comma-separated list of four dimensions that define the lower left x- and y-coordinate as well as the upper right x- and y-coordinate of a rectangle that describes the bounding box of the drawing.

The option `overlay` is an alias for `bounding box={zero}`.




```
foo
\begin{hawkdraw}[
  bounding box={zero},
  baseline={opt}
]
  \stroke[color stroke={blue}]{
    (0cm,0cm) circle[radius={0.5cm}]
  }
\end{hawkdraw}
bar
```

Note that the `hawkdraw` environment sets all category codes to the default LaTeX category codes, but it collects all characters with category code 13 (active) and a character code between 32 and 127, resets these to active inside nodes and rescans the node's contents to ensure proper output.

```
picture/color space={gray}
picture/color space={rgb}
picture/color space={cmyk}
```

The option `color space` sets the environment to be in a fixed color space, either grayscale, RGB or CMYK.

6 Scopes and layers

```
\begin{scope}[‹options›]
\end{scope}
```

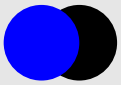
The `scope` environment is only available inside the `hawkdraw` environment. It can be used to create scopes for the localization of path states, style settings and clipping regions. As for options, the `scope` environment accepts the `scope/` set, the `path/` set and the `tag/` set.



```
\begin{hawkdraw}[color fill={blue}]
  \fill{ (0cm,0cm) circle[radius={5pt}] }
  \begin{scope}[color fill={black}]
    \fill{ (1cm,0cm) circle[radius={5pt}] }
  \end{scope}
  \fill{ (2cm,0cm) circle[radius={5pt}] }
\end{hawkdraw}
```

scope/**layer**={‹string›}

The option `layer` sets the scope to be placed on the relevant layer. The layer must be defined beforehand using the `layers` option (see .



```
\begin{hawkdraw}[layers={background,main}]
  \fill[color fill={blue}]{
    (0cm,0cm) circle[radius={0.5cm}]
  }
  \begin{scope}[layer={background}]
    \fill{
      (0.5cm,0cm) circle[radius={0.5cm}]
    }
  \end{scope}
\end{hawkdraw}
```

scope/**position absolute**

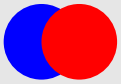
The option `position absolute` shifts the scope so that its origin is at the lower left corner of the current page. Positioning absolutely only works with the option `bounding box` option of the `hawkdraw` environment set to `zero` (or with using the shorthand option `overlay`).

```
\begin{hawkdraw}[overlay]
  \begin{scope}[position absolute]
    \fill[color fill={blue}]{
      (0cm,0cm) circle[radius={2cm}]
    }
  \end{scope}
\end{hawkdraw}
```

7 Path syntax

Paths are defined by a sequence of points and path construction operations. The following sections describe the syntax for defining paths and the available operations. Note that commands used in a path definition need to be expandable in order for the parser to work properly. This is especially true for parts of path definitions that are stored in user defined commands for later use.

The following example shows how to store a (partial) path definition in an expandable command using `\NewExpandableDocumentCommand`.



```
\NewExpandableDocumentCommand{\drawcircle}{}{
  circle[radius={0.5cm}]
}

\begin{hawkdraw}
  \fill[color fill={blue}]{
    (0cm,0cm) \drawcircle
  }
  \fill[color fill={red}]{
    (0.5cm,0cm) \drawcircle
  }
\end{hawkdraw}
```

7.1 Path command and options

```
\path[‹options›]{‹path operations›}
```

The command `\path` is used to to draw a path inside the `hawkdraw` environment. The path is defined by a sequence of points and path construction operations. The optional argument just prepends the relevant options to the path and has the same effect as if the options were specified in the mandatory argument. The following operations can be used to construct the path:

```
[‹options›]
```

Options can be for the path using square brackets. The command `\path` accepts the `path/` set including all subsets. Options can be set multiple times for the same path after any point on the path. Some options (such as color options or transformations) will apply to the entire path and the last specified value will be used.

7.2 Points on the path

Generally, the package makes use of the `l3fp` module of the LaTeX kernel. As such, it is possible to execute calculations on the fly to define the points. The basic form of a point is a tuple of two expressions resulting in a floating-point number where both expressions are separated by a comma.

It is advisable to enclose the two expressions of the tuple in curly braces in cases where complicated calculations should be executed in order to create a group to ensure proper evaluation and avoiding conflicts with parsing the coordinates, especially when using a syntax for polar points as described below.

```
(‹float›,‹float›)
```

Points on the path can be defined using x and y parts as floating-point numbers. The first item in the tuple is the x-coordinate, and the second item is the y-coordinate.

Points are per default defined as tuples representing the x- and y-coordinate. If floating-point numbers without unit are used, points are assumed per default. If other units are used, these are converted into points internally. The below alternative representations of points are as well calculated into the default tuple representation:

```
(‹float›>‹float)
(‹float,‹float›>‹float›)
```

Points on the path can also be defined using radius and angle parts as floating-point numbers. The first item in the tuple is the radius and the second item is the angle. The first item can contain a comma to define two radii.

```
(‹float›:‹float›)
(‹float›:‹float›,‹float›)
```

Points on the path can also be defined using radius and angle parts as floating-point numbers in reversed order. The first item in the tuple is the angle, and the second item is the radius. The second item can contain a comma to define two radii.

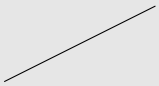
```
(‹string›)
```

Points on the path can finally be referenced using names as strings. Points on the path can be named by either using the `point` operation with the name given as argument (see below) or using the `node` operation with the given name set via the `name` option.

7.3 Straight lines

```
-- (‹point›)
```

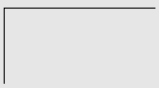
Using `--` a line to the next point is drawn.



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    -- (2cm,1cm)
  }
\end{hawkdraw}
```

```
|- (‹point›)
-| (‹point›)
```

Using `|-` a vertical, then horizontal line to the next point is drawn. Using `-|` a vertical, then horizontal line to the next point is drawn.



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    |- (2cm,1cm)
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    -| (2cm,1cm)
  }
\end{hawkdraw}
```

7.4 Bézier curves

```
.. (‹control point›) .. (‹point›)
.. (‹control point›) & (‹control point›) .. (‹point›)
```

The above syntax draws a Bézier curve to the next point with a single control point (quadratic Bézier) or with two control points (cubic Bézier).



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    .. (1cm,1cm)
    .. (2cm,1cm)
  }
\end{hawkdraw}
```

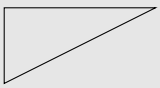


```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    .. (0cm,0.5cm) & (1cm,1cm)
    .. (2cm,1cm)
  }
\end{hawkdraw}
```

7.5 Closing paths

```
!
```

Using `!` the path is closed by drawing a line from the last point to the first point.



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    -- (2cm,1cm)
    -- (0cm,1cm)
    !
  }
\end{hawkdraw}
```

7.6 Circles, ellipses, arcs, rectangles and grids

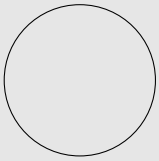
As for options, the following operators only accept the relevant subset to the `path/` set.

```
circle[‹options›]
```

Using the `circle` operation, a circle is drawn with the current point as center.

```
path/circle/radius={‹dimension›}
```

The radius is specified as a dimension with the `radius` option.



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    circle[radius={1cm}]
  }
\end{hawkdraw}
```

ellipse[‹options›]

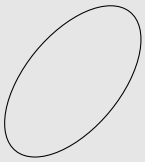
Using the `ellipse` operation, an ellipse is drawn with the current point as center.

```
path/ellipse/radius x={‹dimension›}
path/ellipse/radius y={‹dimension›}
path/ellipse/vector a={‹tuple›}
path/ellipse/vector b={‹tuple›}
```

The vectors of the ellipse are specified as tuples of dimensions and can be set with the `vector a` and `vector b` options. ALternatively, the options `radius x` and `radius y` can be used to define the radii in the x- and y-dimension.



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    ellipse[
      radius x={0.5cm},
      radius y={1cm}
    ]
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    ellipse[
      vector a={(0.5cm,1cm)},
      vector b={(0.75cm,0cm)}
    ]
  }
\end{hawkdraw}
```

arc[‹options›]

Using the `arc` operation, an arc is drawn with the current point as origin.

```
path/arc/radius={‹dimension›}
path/arc/radius x={‹dimension›}
path/arc/radius y={‹dimension›}
path/arc/vector a={‹tuple›}
path/arc/vector b={‹tuple›}
path/arc/angle start={‹float›}
path/arc/angle end={‹float›}
path/arc/angle delta={‹float›}
```

The vectors of the arc are specified as tuples of dimensions and can be set with the `vector a` and `vector b` options.Alternatively, the options `radius x` and `radius y` can be used to define the radii in the x- and y-dimension. It is also possible to use the `radius` option to define a uniform radius for both dimensions. The angles are specified as dimensions with the `angle start` and `angle end` options. Instead of the `angle end` option, the option `angle delta` can be used.



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    arc[
      radius={1cm},
      angle start={45},
      angle end={135}
    ]
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    arc[
      radius x={0.5cm},
      radius y={1cm},
      angle start={45},
      angle delta={90}
    ]
  }
\end{hawkdraw}
```



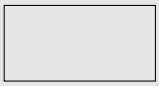
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    arc[
      vector a={(0.5cm,1cm)},
      vector b={(0.75cm,0cm)},
      angle start={45},
      angle end={135}
    ]
  }
\end{hawkdraw}
```

rectangle[‹options›]

Using the `rectangle` operation, an rectangle with the current point as origin corner is drawn.

path/rectangle/**corner**={‹tuple›}
path/rectangle/**relative**={‹boolean›}

The other corner of the rectangle is specified as a point (a tuple) with the `corner` option. If the boolean option `relative` is set, the corner point is interpreted as relative to the current point instead of absolute.



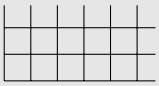
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

grid[‹options›]

Using the `grid` operation, a grid with the current point as origin corner is drawn.

path/grid/**step**={‹clist›}
path/grid/**corner**={‹tuple›}
path/grid/**relative**={‹boolean›}

The step is specified as a comma-separated list of at most two dimensions with the `step` option. The first item in the list is the step in the x-direction, and the second item is the step in the y-direction. If only one dimension is given, it is used for both directions. The other corner is specified as a point (a tuple) with the `corner` option. If the boolean option `relative` is set, the corner point is interpreted as relative to the current point instead of absolute.



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    grid[step={10pt}, corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    grid[step={5pt,10pt}, corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

7.7 Plot functions

plot[‹options›]{(‹point›) ...}

The `plot` operator starts a path segment that uses a function as defined by the user. The operator initiates storing the last point and all points given in the braced argument in a sequence. After the operator, the function is executed using the points stored in the sequence. As for options, the `plot` operator only accepts the `path/plot/` subset.

path/plot/**function**={‹string›}

The plot function can be selected using the `plot/function` option that expects the name of a plot function as string. Currently, two plot functions are predefined, `sharp` and `smooth`, the latter being selected per default.

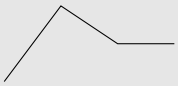
It is possible to use the `map` operator inside the argument to the `plot` operator.



```
\begin{hawkdraw}
  \path[stroke, plot/function={smooth}]{
    (0cm,0cm)
    plot {
      map[end=7] {
        ({#1 * 0.25cm},{sin(#1) * 0.25cm})
      }
    }
    -- (1cm,1.5cm)
  }
\end{hawkdraw}
```

7.7.1 `sharp` plot function

The `sharp` plot function results in an output equivalent to the `--` operator. It moves to the first point in the stored sequence and executes simple line-to operations to the following points.



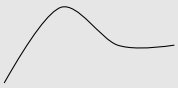
```
\begin{hawkdraw}
  \stroke[plot/function={sharp}]{
    (0cm,0cm)
    plot { (0.75cm,1cm) (1.5cm,.5cm) (2.25cm,.5cm) }
  }
\end{hawkdraw}
```

7.7.2 `smooth` plot function

The `smooth` plot function uses a simple algorithm to smoothen the curve. For paths that have more than three points, it takes the slope of a vector from the point before the relevant point to the point after the relevant point, scales this by a certain amount and uses the resulting points as control points for a Bézier curve. The first and the last path segments are quadratic Bézier curves, all other segments are cubic Bézier curves. If the path only contains two points, a straight line is drawn.

path/plot/**smooth/tension**={‹float›}

The factor by which the slope vector is scaled is set to 0.15 by default and can be adjusted using the option `plot/smooth/tension`.



```
\begin{hawkdraw}
  \stroke[plot/function={smooth}]{
    (0cm,0cm)
    plot { (0.75cm,1cm) (1.5cm,.5cm) (2.25cm,.5cm) }
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \stroke[
    plot/function={smooth},
    plot/smooth/tension={0.25}
  ]{
    (0cm,0cm)
    plot { (0.75cm,1cm) (1.5cm,.5cm) (2.25cm,.5cm) }
  }
\end{hawkdraw}
```

7.8 Points and nodes

Points are defined as tuple of two dimensions where the first represents the dimension on the x-axis and the second the dimension on the y-axis. Apart from defining points with these two dimensions directly, it is also possible to calculate these two dimensions or to define a point as being at a defined part of a path segment. Moreover, it is possible name points and later refer to them.

Nodes are boxes (coffins to be more exact) that can contain arbitrary contents. Nodes also have frames that act as boxes surrounding these contents. Since such contents typically have some width and height, it is possible to refer to the corners and other points on the border of nodes via anchors.

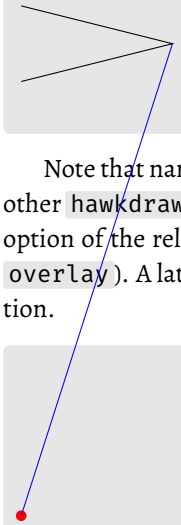
Note that nodes typically cannot directly contain verbatim contents. They can, however, contain boxes that contain verbatim contents.

7.8.1 Points

```
point[‹options›]{‹string›}
```

Using the `point` operation, a point at the current position is defined and named using the value given in the mandatory argument. As for options, the `point` operator only accepts the `path/point/` subset. The point can be referenced later by using the name in parentheses (see above).

The command `\point{#1}` is a shorthand for `\path{ (0pt,0pt) point {#1} }` where `#1` is the name of the point.



```
\begin{hawkdraw}
  \path{ (2cm,0.5cm) point {A} }
  \stroke{ (0cm,0cm) -- (A) -- (0cm,1cm) }
\end{hawkdraw}
```

Note that named points are defined globally. It is therefore possible to refer to points defined in other `hawkdraw` environments on the same page. This only works with the option `bounding box` option of the relevant `hawkdraw` environment set to `zero` (or with using the shorthand option `overlay`). A later definition of the a point with the same name will overwrite the previous definition.


```
\begin{hawkdraw}[overlay]
  \fill[color fill=red]{
    (0cm,0cm)
    circle[radius={2pt}]
  }
  \path[stroke, color stroke={blue}]{
    (0cm,0cm) -- (A)
  }
\end{hawkdraw}
```

```
path/point/at={‹tuple›}
path/point/at part={‹float›}
path/point/at distance={‹dimension›}
```

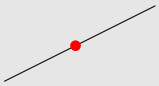
The position of the point can be adjusted with the `at` option which accepts a point (a tuple).

The position of a point can also be adjusted using the `at part` option which accepts a floating-point number that defines the position on the last path segment with 0 being the previous and 1 being the current point. If the value given is negative or larger than 1, the position of the point is interpolated according to the slope at the start or end of the path.



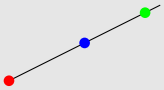
```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (2cm,1cm)
    point[at part={0.25}] {A}
  }
  \fill[color fill=red]{
    (A) circle[radius={2pt}]
  }
\end{hawkdraw}
```

Finally, the position of a point can be adjusted using the `at distance` option which accepts a dimension that defines the position on the last path segment starting from the previous point. If the distance is negative or larger than the length of the path, the position of the point is interpolated according to the slope at the start or end of the path.



```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (2cm,1cm)
    point[at distance={30pt}] {A}
  }
  \fill[color fill=red]{
    (A) circle[radius={2pt}]
  }
\end{hawkdraw}
```

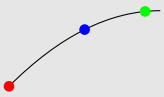
The position on a path using these options depend on the type of the path. For straight and curved lines as well as for circles and circular arcs, the distances are exact relative to the length of the path. For ellipses and elliptical arcs, the distances are distorted relative to the angle. For orthogonal lines, the distances are calculated such that half of the distance lies at the corner. For rectangles, the distances are calculated such that each side takes a quarter of the distance. The following examples show the output of different types of paths:



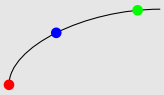
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    -- (2cm,1cm)
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    -| (2cm,1cm)
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    .. (1cm,1cm)
    .. (2cm,1cm)
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



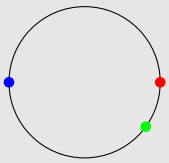
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    .. (0cm,0.5cm) & (1cm,1cm)
    .. (2cm,1cm)
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



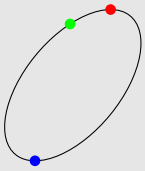
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    arc[
      radius={1cm},
      angle start={45},
      angle end={135}
    ]
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



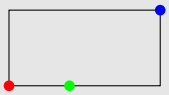
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    arc[
      vector a={(0.5cm,1cm)},
      vector b={(0.75cm,0cm)},
      angle start={45},
      angle end={135}
    ]
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    circle[
      radius={1cm}
    ]
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



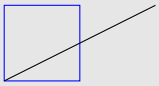
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    ellipse[
      vector a={(0.5cm,1cm)},
      vector b={(0.75cm,0cm)}
    ]
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```



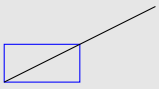
```
\begin{hawkdraw}
  \path[stroke]{
    (0cm,0cm)
    rectangle[
      corner={(2cm,1cm)}
    ]
    point[at part={0}] {A}
    point[at part={0.5}] {B}
    point[at part={0.9}] {C}
  }
  \fill[color fill=red]{ (A) circle[radius={2pt}] }
  \fill[color fill=blue]{ (B) circle[radius={2pt}] }
  \fill[color fill=green]{ (C) circle[radius={2pt}] }
\end{hawkdraw}
```

\getpoint{‹point›} ★

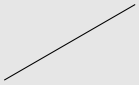
Inside a `hawkdraw` environment, the command `\getpoint` can be used to access the coordinates of a point on the current path. The point is given as an argument. The command returns the coordinates of the point as a tuple of dimensions and can be used in subsequent calculations. Note that you might need to wrap the `\getpoint` command (including the argument) in parenthesis in order to have the parser read its expansion as a tuple.



```
\begin{hawkdraw}
  \point[at={(1cm,1cm)}]{A}
  \stroke[color stroke={blue}]{
    (0cm,0cm) rectangle[corner={(A)}]
  }
  \stroke{
    (0cm,0cm)
    -- ({\getpoint{A} + (1cm,0cm)})
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \point[at={(1cm,0.5cm)}]{A}
  \stroke[color stroke={blue}]{
    (0cm,0cm) rectangle[corner={(A)}]
  }
  \stroke{
    (0cm,0cm)
    -- ({\getpoint{A} * 2})
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm)
    -- ({\getpoint{2cm>30}})
  }
\end{hawkdraw}
```

\getfirstpoint ★
\getlastpoint

The commands `\getfirstpoint` and `\getlastpoint` can be used to access the first and last point of the current path, respectively. The commands return the coordinates of the relevant point as a tuple of dimensions and can be used in subsequent calculations.

\getfirstslope ★
\getlastslope

The commands `\getfirstslope` and `\getlastslope` can be used to access the slope at the first and last point of the current path, respectively. The slope is represented as the angle with respect to the positive x-axis.

\getpointx{‹point›} ★
\getpointy{‹point›}

Inside a `hawkdraw` environment, the commands `\getpointx` and `\getpointy` can be used to access the x- and y-coordinate of a point, respectively. The point is given as an argument, and the commands return the relevant coordinate as dimension.

\getpathlength{‹token list›}

Inside a `hawkdraw` environment, the command `\getpathlength` retrieves the length of the current path (up to this point) and rstores this length as dimension in the token list given as argument.

\getintersectionlines{‹point›}{‹point›}{‹point›}{‹point›} ★

Inside a `hawkdraw` environment, the command `\getintersectionlines` can be used to access the intersection point of two lines. The lines are given as points in four arguments, the first two defining the first line and the last two defining the second line. The command returns the coordinates of the intersection point as a tuple of dimensions and can be used in subsequent calculations.

\getintersectioncircles[‹integer›]{‹point›}{‹dimension›}{‹point›}{‹dimension›} ★

Inside a `hawkdraw` environment, the command `\getintersectioncircles` can be used to access the intersection points of two circles. The circles are given as a point and radius in the first and next two mandatory arguments. The optional argument specifies the number of the intersection to return (root) and defaults to 1. The command returns the coordinates of the intersection points as tuples of dimensions and can be used in subsequent calculations.

\getintersectionlinecircle[‹integer›]{‹point›}{‹point›}{‹point›}{‹dimension›} ★

Inside a `hawkdraw` environment, the command `\getintersectionlinecircle` can be used to access the intersection points of a line and a circle. The line is given as two points in the first two mandatory arguments, and the circle is given as a point and radius in the next two mandatory arguments. The optional argument specifies the number of the intersection to return (root) and defaults to 1. The command returns the coordinates of the intersection points as tuples of dimensions and can be used in subsequent calculations.

\getpagewidth ★
\getpageheight

The commands `\getpagewidth` and `\getpageheight` can be used to access the width and height of the current page, respectively. The commands return dimensions and can be used in subsequent calculations.

7.8.2 Nodes and frames

node[‹options›]{‹content›}

Using the `node` operation, a node at the current position is defined and assigned the relevant content by the relevant argument. As for options, the `node` operator accepts the `path/node/` subset as well as the `path/` set.

path/node/**rescan**={‹boolean›}

Using the `rescan` option, the node's contents are rescanned using the current category codes. The default value is `false`, but rescanning is activated automatically if active characters with character code between 32 and 127 are found before the relevant `hawkdraw` environment.

path/node/**name**={‹string›}
path/node/**at**={‹tuple›}
path/node/**at part**={‹float›}
path/node/**at distance**={‹dimension›}

The node can be named by setting the `name` option which can be referenced later by using the name in parentheses (see above). The position of the node can be adjusted with the `at` option which accepts a point (a tuple) or with the options `at part` or `at distance` (see above).

path/node/**anchor**={‹clist›}
path/node/**anchor**={‹float›}

The anchor of a node is specified with the `anchor` option which accepts a comma-separated list of at most two poles that define the anchor point as their intersection (see the explanations on coffins). Available poles are `t`, `b`, `H` (which refers to the baseline of the node's contents), `l`, `r`, `vc` and `hc`. Nodes where the width is set additionally provide the poles `T` and `B` that refer to the top and bottom baseline of the node's contents respectively.

It is possible to reference a point where two poles of the node intersect by appending the node name with the relevant poles separated by a comma, enclosed by square brackets. For example, if a node is named `foo`, the point where the top and left pole of the node intersect can be referenced by `(foo[t,l])`.

The `anchor` option also accepts a unit-less floating-point number that defines as anchor the point where a line starting from the center of the node and having an angle as specified by this number intersects with the node's border.

It is also possible to reference such an anchor point. For example, if a node is named `foo`, the point where a line with an angle of 30 degrees intersects with the node's border can be referenced by `(foo[30])`.



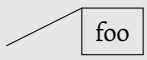

```
\begin{hawkdraw}
  \node{foo}
\end{hawkdraw}
```



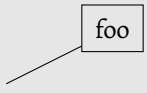
```
\begin{hawkdraw}
  \node[stroke, name={A}]{foo}
  \fill[color fill={red}]{
    (A[b,l]) circle[radius={2pt}]
  }
  \fill[color fill={red}]{
    (A[t,r]) circle[radius={2pt}]
  }
\end{hawkdraw}
```




```
\begin{hawkdraw}
  \node[stroke, name={B}]{foo}
  \fill[color fill={red}]{
    (B[30]) circle[radius={2pt}] }
  \fill[color fill={red}]{
    (B[270]) circle[radius={2pt}]
  }
\end{hawkdraw}
```



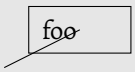

```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (1cm,0.5cm)
    node[anchor={t,l}, stroke] {foo}
  }
\end{hawkdraw}
```




```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (1cm,0.5cm)
    node[anchor={210}, stroke] {foo}
  }
\end{hawkdraw}
```

`path/node/`**`width`**`={‹dimension›}`

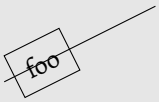
The width of the node can be set with the `width` option.




```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (1cm,0.5cm)
    node[stroke, width={1cm}] {foo}
  }
\end{hawkdraw}
```

path/node/**sloped**={‹boolean›}

Using the `sloped` option, the node can be rotated to follow the direction of the path. The default value is `false`.



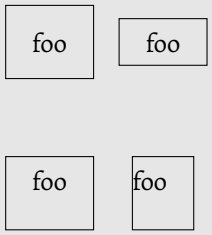

```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (2cm,1cm)
    node[
      stroke,
      at part={0.25},
      sloped
    ] {foo}
  }
\end{hawkdraw}
```

path/node/**flipped**={‹boolean›}

Using the `flipped` option, the node can be rotated by additional 180 degrees if it is sloped. The default value is `false`.

path/node/**padding**={‹clist›}

Using the `padding` option, the padding between the border of the node and the contents is set. The value is a comma-separated list of one to four dimensions. If four dimensions are given, they apply to the top, left, bottom and right side of the node respectively. If three dimensions are given, the first apples to the top, the second to the left and right side, the third to the bottom. If two dimensions are given, the first applies to the top and bottom and the second to the left and right sides. If only one dimension is given, it is applied to all four sides.




```
\begin{hawkdraw}
  \node[
    stroke,
    at={(0cm,2cm)},
    padding={10pt}
  ]{foo}
  \node[
    stroke,
    at={(1.5cm,2cm)},
    padding={5pt,10pt}
  ]{foo}
  \node[
    stroke,
    at={(0cm,0cm)},
    padding={5pt,10pt,15pt}
  ]{foo}
  \node[
    stroke,
    at={(1.5cm,0cm)},
    padding={5pt,10pt,15pt,0pt}
  ]{foo}
\end{hawkdraw}
```

```
path/node/color text={‹color›}
path/node/opacity text={‹float›}
```

Using the `color text` option, the color of the node contents is set (fill and stroke if applicable). The default text color is the current (text) color. The option `text opacity` sets the text opacity. The floating-point number should be in the range of 0 and 1. The default text opacity is 1 (fully opaque).

Note that `\DocumentMetadata{}` needs to be set in order to activate support for PDF transparency.

```
path/node/color text fill={‹color›}
path/node/color text stroke={‹color›}
path/node/opacity text fill={‹float›}
path/node/opacity text stroke={‹float›}
```

Using the options `color text fill` and `color text stroke`, the fill and stroke colors of the node contents are set specifically.

Using the options `opacity text fill` and `opacity text stroke`, the fill and stroke opacity of the node contents are set specifically.

Note that `\DocumentMetadata{}` needs to be set in order to activate support for PDF transparency.

```
path/node/background={‹code›}
```

The option `background` can be used to set a background to the node. The code can, for example, include an image using the `\includegraphics` command (for which to be available, the `graphicx` package needs to be loaded).




```
\begin{hawkdraw}
  \node[
      background={
        \includegraphics{
          example-image-duck
        }
      }
    ]{\Huge foo}
\end{hawkdraw}
```

path/node/**text rendering**={‹clist›}

The option `text rendering` sets the text rendering mode of the node contents. The value is a comma-separated list of one to three modes `fill`, `stroke` and `clip`. The option also accepts the value `none`. Using `clip`, the text is used as a clipping path on the node's background.

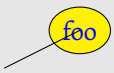



```
\begin{hawkdraw}
  \node[
      padding={0pt},
      pattern={dots},
      color text fill={yellow},
      color text stroke={red},
      text rendering={fill, stroke, clip}
    ]{\Huge foo}
\end{hawkdraw}
```

Note that if the text rendering mode is set to `clip`, the node's frame and background are clipped by the node's contents and therefore not visible. On the other hand, the node's contents are not visible if the text rendering mode is set to `none`.

path/node/**frame**={‹string›}

Using the `frame` command, the frame surrounding the node text can be selected. The `rectangle` frame is set per default. All options that are available for paths are also available for node frames.




```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (1cm,0.5cm)
    node[
      fill,
      color fill={yellow},
      color text={blue},
      frame={ellipse}
    ] {foo}
  }
\end{hawkdraw}
```

The command `\node{#1}` is a shorthand for `\path{ (0pt,0pt) node {#1} }` where `#1` is the contents of the node.

The following node frames are available, the frame `none` is special as it removes the bounding box from the node altogether:

Node frame	Name
foo	`none`
foo	`rectangle`
foo	`ellipse`
foo	`circle`
foo	`diamond`

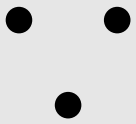
8 Mapping operation

```
map[‹options›]{‹code›}
```

The `map` operation is used to map over a comma-separated list of items or a range of numbers and execute code for each item that is given by the relevant argument. The code can reference the current item with `#1` and the current index with `#2`. The index starts at 1. The `map` operation can be nested. As for options, the `map` operator only accepts the `path/map/` subset.

```
path/map/list={‹clist›}
path/map/start={‹float›}
path/map/end={‹float›}
path/map/step={‹float›}
```

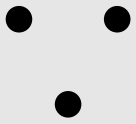
The option `list` sets the command to map over a comma-separated list of items. Each item in the list is passed as an argument to the code.



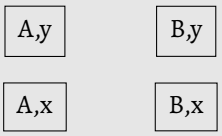
```
\begin{hawkdraw}
  \fill{
    map[list={30,150,270}] {
      (0.75cm>#1) circle[radius={5pt}]
    }
  }
\end{hawkdraw}
```

The option `start` sets the start of the range for mapping over a range of numbers as a floating-point number. The default start is 1. The option `end` sets the end of the range for mapping over a range of numbers as a floating-point number. The default end is 1. The option `step` sets the step for the range for mapping over a range of numbers as a floating-point number. The default step is 1.

The command `\map{#1}` is a shorthand for `\path{ (0pt,0pt) map {#1} }` where `#1` is the code to be mapped over.



```
\begin{hawkdraw}
  \map[start={30}, step={120}, end={270}] {
    [fill] (0.75cm>#1) circle[radius={5pt}]
  }
\end{hawkdraw}
```




```
\begin{hawkdraw}
  \stroke{
    map[list={A,B}] {
      map[list={x,y}] {
        (#2 cm,##2 cm) node {#1,##1}
      }
    }
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm)
    .. (7mm,-10mm) & (14mm,8mm) .. (20mm,-5mm)
    map[end={10}] {
        tip[
          at part={0.095 * #1},
          color fill=blue! #1 0!red
        ] {stealth}
    }
  }
\end{hawkdraw}
```

\hawkdrawmap[‹option›]{‹code›}

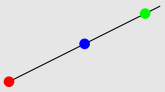
The command `\hawkdrawmap` can be used to repeat whole paths. The syntax is the same as for the `map` option.

\hawkdrawarrayset{‹string›}{‹clist›}

The command `\hawkdrawarrayset` can be used to define arrays of values. The first argument is the name of the array, and the second argument is a comma-separated list of items that are stored in the array. The items can be referenced with `\hawkdrawarrayitem`.

\hawkdrawarrayitem{‹string›}{‹integer›} ★

The command `\hawkdrawarrayitem` can be used to reference items in an array. The first argument is the name of the array, and the second argument is the index of the item to reference. The index is an integer where the first item has index 1.



```
\hawkdrawarrayset{myparts}{0,0.5,0.9}
\hawkdrawarrayset{mycolors}{red,blue,green}

\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (2cm,1cm)
    map[list={A,B,C}] {
      point[
        at part={\hawkdrawarrayitem{myparts}{#2}}
      ] {#1}
    }
  }
  \hawkdrawmap[list={A,B,C}]{
    \fill[
      color fill={\hawkdrawarrayitem{mycolors}{#2}}
    ]{ (#1) circle[radius={2pt}] }
  }
\end{hawkdraw}
```

9 Path options

The following options can be set for the path:

9.1 Stroke and color fill, opacity, clipping

```
path/stroke
path/stroke={false}
path/fill
path/fill={false}
```

These options add a stroke to the path or fill the path respectively. They do not require a value. Setting the value to `false` resets a prior setting and removes the path stroke or fill.

The command `\stroke` is a shorthand for `\path[stroke]`.

The command `\fill` is a shorthand for `\path[fill]`.

```
path/clip={‹boolean›}
```

The option `clip` sets the current path as a clipping region for subsequent paths. The option does not require a value but can be set explicitly to `true` or `false`. The option `clip` can be used in combination with the options `stroke` and `fill`. As the clipping region affects all following paths, it is often useful to use it in combination with scopes (see above).

The command `\clip` is a shorthand for `\path[clip]`.



```
\begin{hawkdraw}
  \path[stroke, clip]{
    (0cm,0cm)
    rectangle[corner={(2cm,1cm)}]
  }
  \fill{
    (1cm,0.5cm)
    circle[radius={0.6cm}]
  }
\end{hawkdraw}
```

```
path/color stroke={‹color›}
path/color fill={‹color›}
path/opacity stroke={‹float›}
path/opacity fill={‹float›}
```

The options `color stroke` and `color fill` set the color stroke and the color fill for the path. The default color stroke and color fill is the current (text) color.



```
\begin{hawkdraw}[line width={5pt}]
  \path[
    fill, color fill={yellow},
    stroke, color stroke={blue}
  ]{
    (0cm,0cm)
    rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}[line width={5pt}]
  \path[
    fill, opacity fill={0.25},
    stroke, opacity stroke={0.5}
  ]{
    (0cm,0cm)
    rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

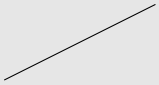
The options `opacity stroke` and `opacity fill` set the opacity stroke and the opacity fill for the path. The floating-point number should be in the range of 0 and 1. The default opacity stroke and opacity fill is 1 (fully opaque).

Note that `\DocumentMetadata{}` needs to be set in order to activate support for PDF transparency.

9.2 Line styling

```
path/line width={‹dimension›}
```

Sets the line width for the path. The default line width is 0.4 pt.



```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (2cm,1cm)
  }
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \stroke[line width={5pt}]{
    (0cm,0cm) -- (2cm,1cm)
  }
\end{hawkdraw}
```

path/**line cap**={‹string›}
path/**line join**={‹string›}
path/**miter limit**={‹float›}

The option `line cap` sets the line cap for the path. The possible values are `butt`, `round`, and `rectangle`. The default line cap is `butt`.



```
\begin{hawkdraw}[line width={5pt}]
  \stroke[line cap={round}]{
    (0cm,0cm) -- (2cm,1cm)
  }
\end{hawkdraw}
```

The option `line join` sets the line join for the path. The possible values are `miter`, `round`, and `bevel`. The default line join is `miter`.

The option `miter limit` sets the miter limit for the path. The default miter limit is 10.

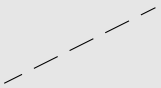


```
\begin{hawkdraw}[line width={5pt}]
  \stroke[line join={bevel}]{
    (0cm,0cm) -- (2cm,0.5cm)
    -- (0cm,1cm)
  }
\end{hawkdraw}
```

path/**dash pattern**={‹clist›}
path/**dash phase**={‹dimension›}

The option `dash pattern` sets the dash pattern for the path as a comma-separated list of dimensions. Each item in the list is a dimension representing the length of a dash or a gap. The default dash pattern is empty (a solid line).

The option `dash phase` sets the dash phase for the path as a dimension. The default dash phase is 0 pt.

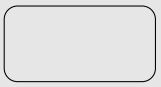


```
\begin{hawkdraw}
  \stroke[
    dash pattern={10pt,5pt},
    dash phase={2.5pt}
  ]{
    (0cm,0cm) -- (2cm,1cm)
  }
\end{hawkdraw}
```

9.3 Rounded corners and fill rule

path/**corner arc**={‹clist›}

The option `corner arc` the corner arc for the path as a comma-separated list of at most two dimensions. The first applies to the corner formed by the path leading into the corner, and the second applies to the corner formed by the path leading out of the corner. If only one dimension is given, it applies to both corners. The default corner arc is 0 pt (sharp corners).



```
\begin{hawkdraw}
  \stroke[corner arc={5pt}]{
    (0cm,0cm)
    rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

path/**fill rule**={‹string›}

This option sets the fill rule for the path. The possible values are `non-zero` and `even-odd`. The default fill rule is `non-zero`. Note that this option is not influenced by scoping.

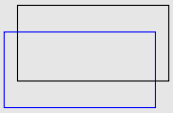


```
\begin{hawkdraw}[fill rule={even-odd}]
  \fill{
    (0cm,0cm)
    rectangle[corner={(2cm,1cm)}]
    (1cm,0.5cm)
    circle[radius={0.6cm}]
  }
\end{hawkdraw}
```

9.4 Transformations

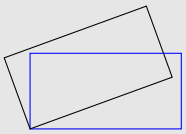
path/**shift**={‹tuple›}
path/**rotate**={‹float›}
path/**scale**={‹tuple›}
path/**slant**={‹tuple›}

The option `shift` sets the shift for the path as a tuple of dimensions. The first item in the tuple is the shift in the x-direction, and the second item is the shift in the y-direction. The default shift is (0 pt, 0 pt).



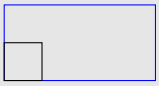
```
\begin{hawkdraw}
  \stroke[color stroke={blue}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
  \stroke[shift={(5pt,10pt)}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

The option `rotate` sets the rotation for the path as a floating-point number representing the angle of rotation. The default rotation is 0°.



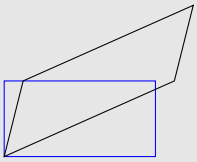
```
\begin{hawkdraw}
  \stroke[color stroke={blue}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
  \stroke[rotate={20}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

The option `scale` sets the scale for the path as a comma-separated list of at most two floating-point numbers. The first item in the tuple is the scale in the x-direction, and the second item is the scale in the y-direction. If only one number is given, it is used for both directions. The default scale is (1, 1).



```
\begin{hawkdraw}
  \stroke[color stroke={blue}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
  \stroke[scale={0.25,0.5}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

The option `slant` sets the slant for the path as a comma-separated list of at most two floating-point numbers. The first item in the tuple is the slant in the x-direction, and the second item is the slant in the y-direction. If only one number is given, it is used for both directions. The default slant is (0, 0).



```
\begin{hawkdraw}
  \stroke[color stroke={blue}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
  \stroke[slant={0.25,0.5}]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

10 Path manipulations

The `hawkdraw` packages makes use of a softpath mechanism to allow for manipulations of points of a paths. The mechanism for rounding path corners via the `corner arc` option (see above) relies on a lower-level softpath implementation of `l3draw`. Currently, mechanisms for shortening paths and for offsetting paths are provided by the `hawkdraw` which are described below.

Note that the order of manipulations is executed in the order the relevant options are given to the path. Thus, if a path should be shortened and also offset using a list (creating essentially two partial paths), the options for shortening should be given before the option for offsetting the path.

```
path/keep original path info={‹boolean›}
```

The option `keep original path info` can be used to restore the information about the position and slope of the first and last point of the path to that of the original path before it has been manipulated. Path manipulations per default change the information about the position and slope of the first and last point of the path, which for example affects where arrow tips are positioned.

10.1 Shortening paths

```
path/shorten start={‹dimension›}
path/shorten end={‹dimension›}
```

The options `shorten start` and `shorten end` each require as argument a dimension representing the length by which the current path should be shortened at its start or end. Only the first or last segment of the path is affected by this manipulation. Arrow tips are positioned at the updated start or end point of the path.

Shortening is not executed on circles, ellipses rectangles and grids.



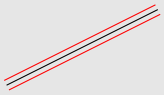
```
\begin{hawkdraw}
  \stroke{ (0cm,0cm) -- (2cm,1cm) }
  \stroke[
    shorten start={10pt},
    color stroke={red},
    line width={2pt}
  ]{ (0cm,0cm) -- (2cm,1cm) }
\end{hawkdraw}
```

10.2 Offsetting paths

```
path/offset={‹clist›}
```

The option `offset` requires as argument a comma-separated list of dimensions each representing the length by which the current path should be offset. Arrow tips are positioned at the updated start or end point of the last path of the comma-separated list.

Note that the offset paths for curves, ellipses and arcs are approximated mostly by reconstructing the relevant path by three to twelve Bézier curves which may result in suboptimal output especially for relatively sharp curves. Furthermore, the position of the point where two path segments meet is approximated for curves and arcs.



```
\begin{hawkdraw}
  \stroke{ (0cm,0cm) -- (2cm,1cm) }
  \stroke[
    offset={2pt,-2pt},
    color stroke={red}
  ]{ (0cm,0cm) -- (2cm,1cm) }
\end{hawkdraw}
```

10.3 Softpath-related settings

```
settings/length precision={‹integer›}
```

In order to calculate the path length of curves, a number of points on the curve are identified and the sum of the distances between these points is calculated. The result becomes more precise with a larger number of points. This number can be set via the option `setting/length precision`. The default number is 25.

```
settings/bisection count={‹integer›}
```

In order to create offset paths for Bézier curves, arcs and ellipses, the relevant offset path is approximated using a number of cubic Bézier curves that are smoothened at their convergence points. The default number of curve segments to approximate a Bézier curve or an arc up to 90 degrees is 3. Arcs that have a delta angle larger than 90 degrees as well as ellipses are split up into parts of at most 90 degrees. The number of curve segments can be adjusted using the `setting/length precision` option.

11 Defining custom styles and colors

```
path/style set={‹options›}
path/style add={‹options›}
```

To simplify styling, custom options can be defined via the options `style set` and `style add`. Both options accept as value a key-value list consisting of one or multiple custom options whose relevant value consists of another key-value list describing the relevant style. It is possible to use `#1` as placeholder for one argument. It is also possible to reference to an already define custom option in the definition of another custom option.

The option `style set` defines custom options and sets their values. The option `style add` appends the relevant key-value list to the existing custom option as defined by `style set`. Note that it is not checked whether a custom option already exists and its definition will be overwritten without a warning.



```
\begin{hawkdraw}[
  style set={
    mystyle={
      stroke, color stroke={blue},
      line width={5pt}
    }
  }
]
  \path[mystyle]{
    (0cm,0cm) -- (2cm,1cm)
  }
\end{hawkdraw}
```

\hawkdrawsetcolor{‹color›}{‹model›}{‹value›}

The command `\hawkdrawsetcolor` is used to define colors using the `l3color` module. It is recommended to define custom colors in the preamble of the document. If the model is set to `named`, the command expects as value a color expression.



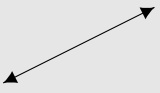
```
\hawkdrawsetcolor{colorA}{rgb}{0.1,0.5,0.8}
\hawkdrawsetcolor{colorB}{named}{yellow!90!red}
\begin{hawkdraw}[line width={5pt}]
  \path[
    stroke, color stroke={colorA},
    fill, color fill={colorB}
  ]{ (0cm,0cm) rectangle[corner={(2cm,1cm)}] }
\end{hawkdraw}
```

12 Arrow tips

12.1 Arrow tips at path ends

path/**arrow start**={‹string›}
path/**arrow end**={‹string›}

The options `arrow start` and `arrow end` set an arrow tip to the start and the end of the current path respectively. Both keys take as value a string representing the name of the relevant arrow tip. The `default` arrow tip is selected per default. Arrow tips are not attached to single points or closed paths.



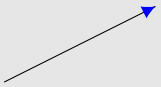
```
\begin{hawkdraw}
  \stroke[arrow start, arrow end]{
    (0cm,0cm) -- (2cm,1cm)
  }
\end{hawkdraw}
```

```
path/arrow style set={‹options›}
path/arrow style add={‹options›}
path/arrow start style set={‹options›}
path/arrow start style add={‹options›}
path/arrow end style set={‹options›}
path/arrow end style add={‹options›}
```

Arrows can be styled by adding the relevant path options to the options `arrow start style set` and `arrow end style set`. The option `arrow style set` sets the same style for start and end arrow tips.

The options `arrow start style add`, `arrow end style add` and `arrow style add` append the relevant styles to the existing styles. All options that are available for paths are also available for arrow tips.

Note that these options set the style of the following arrow tips. If multiple arrow tips with different styles should be applied to the same path, the options need to be set before setting each arrow tip respectively.



```
\begin{hawkdraw}
  \stroke[
    arrow style set={fill, color fill=blue},
    arrow end
  ]{
    (0cm,0cm) -- (2cm,1cm)
  }
\end{hawkdraw}
```

The following arrow tips are available:

Arrow tip	Name
	`none`
	`default`
	`stealth`
	`triangle`
	`kite`
	`circle`
	`simple`
	`straight`
	`bar`
	`slash`
	`barslash`
	`tail`

12.2 Arrow tips on paths

```
tip[‹options›] {‹arrow›}
```

Using the `tip` operation, an arrow head is drawn at the current point on the path.

```
path/tip/at part={‹float›}
path/tip/at distance={‹dimension›}
```

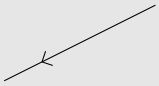
The position of the arrow head can be adjusted with the `at part` option which accepts a unit-less floating-point number between 0 and 1 representing the position on the path. Alternatively, it can be adjusted with the `at distance` option which accepts a dimension (see above).



```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (2cm,1cm)
    tip[at part={0.25}] {straight}
  }
\end{hawkdraw}
```

path/tip/**flipped**={‹boolean›}

If the boolean option `flipped` is set, the arrow head is rotated to the other direction.



```
\begin{hawkdraw}
  \stroke{
    (0cm,0cm) -- (2cm,1cm)
    tip[at part={0.25}, flipped] {straight}
  }
\end{hawkdraw}
```

13 Patterns

Note that in order to be able to use patterns, `\DocumentMetadata{}` needs to be set.

path/**pattern**={‹string›}

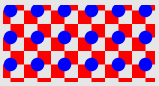
The option `pattern` fills the current path with a pattern. The key takes as value a string representing the name of the relevant pattern. It is possible to additionally set a color as fill to the same path.

path/**pattern style set**={‹options›}
path/**pattern style add**={‹options›}

Patterns can be styled by passing the relevant path options to the options `pattern style set` and `pattern style add`. The option `pattern style set` sets the style for the pattern of the current path. The options `pattern style add` appends the relevant styles to the existing styles. All options that are available for paths are also available for patterns.

Note that patterns in general inherit the style set to the original path. Therefore, if the option `corner arc` is set to a non-zero value for the original path, the pattern will also be drawn with rounded corners. If this is not desired, the option `corner arc` can be set to 0 pt using the option `pattern style set` or `pattern style add`.

Further note that these options set the style of the following pattern. If multiple patterns with different styles should be applied to the same path, the options need to be set before setting each pattern respectively.



```
\begin{hawkdraw}
  \path[
    pattern style set={color fill=red},
    pattern={check},
    pattern style set={color fill=blue},
    pattern={dots}
  ]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

path/**pattern tile size**={‹tuple›}

The option `pattern tile size` accepts a tuple of two dimensions representing the size of the pattern tile. The default size of a pattern tile is 10 pt × 10 pt. Transformations applied to the pattern via `pattern style set` or `pattern style add` don't affect the size of the pattern tile.

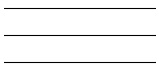
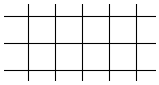
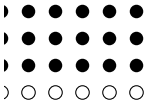
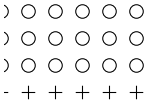
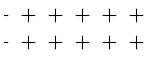
path/**pattern type**={‹clist›}

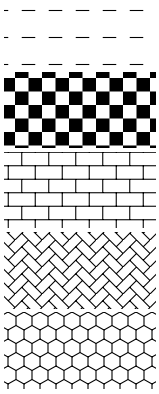
The option `pattern type` accepts a comma-separated list of strings representing the type of pattern that should be used for the current path. The available pattern types are `fill` and `stroke`. The default pattern type is `fill`.



```
\begin{hawkdraw}
  \path[
    pattern={dots},
    line width={10pt},
    pattern type={stroke}
  ]{
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

The following patterns are available:

Pattern	Name
	`lines`
	`waves`
	`hatch`
	`dots`
	`circles`
	`plus`



`minus`

`check`

`masonry`

`herringbone`

`hexagonal`

14 Opacity and shadings

Note that in order to be able to use opacity and shadings, `\DocumentMetadata{}` needs to be set. The functionality in this section is relatively advanced and may not be supported by all PDF viewers. It is recommended to use the latest version of Adobe Acrobat Reader.

14.1 Opacity groups

An opacity group treats multiple paths as a single object for the purpose of applying opacity. This means that the opacity is applied to the entire scope, rather than to each individual path within the scope.

`scope/`**`opacity group`**

The option `opacity group` sets the current scope to an opacity group. The key does not take any value. Note that the option `opacity group` is only available for scopes.



```
\begin{hawkdraw}
  \begin{scope}[
    opacity fill=0.5
  ]
    \fill{
      (0cm,0cm)
      circle[radius={0.5cm}]
    }
    \fill{
      (-1cm,-0.25cm)
      rectangle[corner={1cm,0.25cm}]
    }
  \end{scope}
\end{hawkdraw}
```



```
\begin{hawkdraw}
  \begin{scope}[
    opacity fill=0.5,
    opacity group
  ]
    \fill{
      (0cm,0cm)
      circle[radius={0.5cm}]
    }
    \fill{
      (-1cm,-0.25cm)
      rectangle[corner={1cm,0.25cm}]
    }
  \end{scope}
\end{hawkdraw}
```

14.2 Soft masks

Soft masks are used to create transparency effects by defining a mask that determines the opacity of different parts of an object. Soft masks are defined by a grayscale image or drawing where white parts in the soft mask are fully transparent while black parts are fully opaque.

```
\\\begin{hawkdrawsoftmask}{‹string›}[‹options›]
\end{hawkdrawsoftmask}
```

A soft mask can be defined using the `hawkdrawsoftmask` environment. It requires as first argument a string that represents the name of the soft mask. The soft mask is defined by the contents of the environment. The optional argument can be used to set the options for the soft mask which are the same as for the `hawkdraw` environment.

```
scope/soft mask use={‹string›}
```

The option `soft mask use` applies a soft mask of the current scope. The option takes as string the name of the previously defined soft mask to be applied. Note that the option `soft mask use` is only available for scopes.

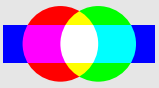
```
\begin{hawkdrawsoftmask}{custom}
  \fill[color fill={black!50}]{
    (0cm,0cm)
    circle[radius={0.5cm}]
  }
  \fill[color fill={black!25}]{
    (-1cm,-0.25cm)
    rectangle[corner={(1cm,0.25cm)}]
  }
\end{hawkdrawsoftmask}
\begin{hawkdraw}
  \begin{scope}[
    soft mask use={custom}
  ]
    \path[
      shading={radial},
      shading colors={red,blue,green}
    ]{ (0cm,0cm) circle[radius={0.75cm}] }
  \end{scope}
\end{hawkdraw}
```

14.3 Blend modes

Blend modes change the way colors of overlapping objects are combined.

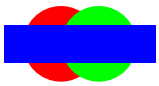
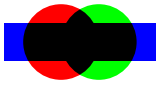
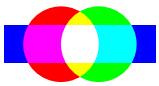
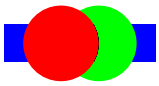
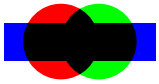
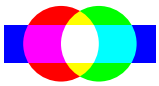
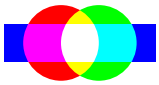
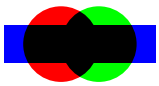
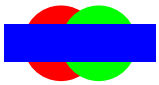
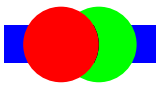
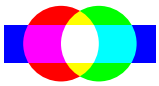
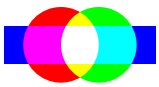
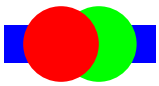
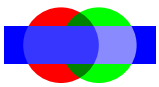
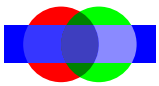
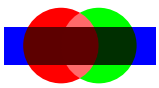
```
scope/blend mode={‹string›}
```

The option `blend mode` sets the blend mode of the current scope. The key takes as string one of the following values. Note that the option `blend mode` is only available for scopes. Some PDF viewers render blend modes differently when they are not applied inside an opacity group, therefore, it is advised to use blend modes inside a blend mode in general.



```
\begin{hawkdraw}
  \begin{scope}[opacity group]
    \begin{scope}[
      blend mode={screen}
    ]
      \fill[color fill={red}]{
        (-0.25cm,0cm)
        circle[radius={0.5cm}]
      }
      \fill[color fill={green}]{
        (0.25cm,0cm)
        circle[radius={0.5cm}]
      }
      \fill[color fill={blue}]{
        (-1cm,-0.25cm)
        rectangle[corner={1cm,0.25cm}]
      }
    \end{scope}
  \end{scope}
\end{hawkdraw}
```

The following blend modes are available:

Blend mode	Name
	`normal`
	`multiply`
	`screen`
	`overlay`
	`darken`
	`lighten`
	`color dodge`
	`color burn`
	`hard light`
	`soft light`
	`difference`
	`exclusion`
	`saturation`
	`color`
	`hue`
	`luminosity`

14.4 Linear, radial and conic shadings

path/**shading**={‹string›}

The option `shading` fills the current path with a shading. The key takes as value a string representing the name of the relevant shading. Note that a shading is placed behind the path and therefore, an additional fill or pattern will cover the shading.

path/**shading colors**={‹clist›}

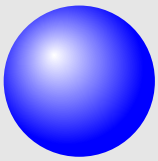
The option `shading colors` accepts a comma-separated list of strings that represent names of predefined colors.

path/**shading rotate**={‹float›}

The option `shading rotate` accepts a unit-less floating-point number representing the angle of the rotation of the shading.

path/**shading offset**={‹float›}

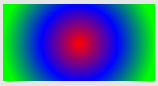
The option `shading offset` accepts a unit-less floating-point number ranging between the values 0 and 1 representing the offset of the center of a radial shading. A value of 0 means that the center has no offset, a value of 1 means that the center sits at the right side of the shading.



```
\begin{hawkdraw}
  \path[
    shading={radial},
    shading colors={white,blue},
    shading offset={0.5},
    shading rotate={135}
  ]{ (0cm,0cm) circle[radius={1cm}] }
\end{hawkdraw}
```

path/**shading circular**={‹boolean›}

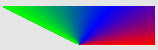
The option `shading circular` sets radial shadings to be circular instead of adhering to the aspect ratio which they do per default.



```
\begin{hawkdraw}
  \path[
    shading={radial},
    shading colors={red,blue,green},
    shading circular
  ]{ (0cm,0cm) rectangle[corner={(2cm,1cm)}] }
\end{hawkdraw}
```

path/**shading angle delta**={‹float›}

The option `shading angle delta` sets conic shadings to only extend to the unit-less floating-point number as degrees given as value to the option. Per default, conic shadings will extend from 0 to 360 degrees. The area of the path not covered by the shading remains transparent.



```
\begin{hawkdraw}
  \path[
    shading={conic},
    shading colors={red,blue,green},
    shading angle delta={135}
  ]{ (0cm,0cm) rectangle[corner={(2cm,1cm)}] }
\end{hawkdraw}
```

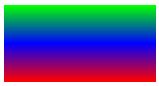
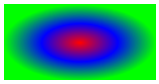
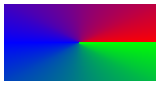
path/**shading type**={‹clist›}

The option `shading type` accepts a comma-separated list of strings representing the type of shading that should be used for the current path. The available shading types are `fill` and `stroke`. The default shading type is `fill`.



```
\begin{hawkdraw}
  \path[
    shading={linear},
    shading colors={red,blue,green},
    line width={10pt},
    shading type={stroke}
  ] {
    (0cm,0cm) rectangle[corner={(2cm,1cm)}]
  }
\end{hawkdraw}
```

The following shadings are available:

Shading	Name
	`linear`
	`radial`
	`conic`

15 Tagging support

The `hawkdraw` package supports tagging to provide accessible PDF structures. The following tags are available for the `hawkdraw` environment as well as for nodes:

tag/**text**
tag/**artifact**
tag/**actualtext**={‹string›}
tag/**alt**={‹string›}
tag/**off**

The `text` key, which is set per default, tags all paths of a `hawkdraw` environment as artifacts and the contents of all nodes as text in the order of their appearance in the code.

The option `artifact` tags all paths and nodes of a `hawkdraw` environment as artifact.

The option `actualtext` sets the relevant value as replacement text for the `hawkdraw` environment and tags it as inline span element.

The option `alt` sets the relevant value as replacement text for the `hawkdraw` environment and tags it as block (figure) element.

The option `off` turns off tagging for the `hawkdraw` environment.

tag/**tagging-setup**={‹options›}

To enable more complex tagging scenarios, the `tagging-setup` option can be used to configure the tagging behavior. It accepts as options the same options as described above. In addition, it accepts the option `tag` to define the element tag.

For example, `tagging-setup={alt={Water (H20)}, tag={Formula}}` would tag the relevant `hawkdraw` environment as a block element with the tag `Formula` and set the replacement text to "Water (H20)".

16 Bidirectionality support

global/**bidi**={‹boolean›}

The `hawkdraw` package supports for bidirectional typesetting when used with LuaTeX. To enable it, load the package with the `bidi` option in the document preamble.

In general, a right-to-left context affects horizontal dimensions which also affects how paths are rendered. This can result in unintended output and means that `hawkdraw` environments should always be placed in a left-to-right context. On the other hand, the contents of nodes should retain the original text direction in most cases. This means that an optimal approach is to store the current text direction, set the text direction to left-to-right for the drawing and reset the original text direction for each node content.

With LuaTeX and the above option active, these changes are done automatically. For other compilation engines such as PDFLaTeX or XeLaTeX, the following code can be inserted into the document preamble to implement a similar technique inserting commands to set and unset the direction into the relevant hooks.

```
\AddToHook{env/hawkdraw/begin}{\beginL}
\AddToHook{env/hawkdraw/end}{\endL}
\AddToHook{hawkdraw/node/begin}{\beginR}
\AddToHook{hawkdraw/node/end}{\endR}
```

17 Use with ConTeXt

The package provides a ConTeXt module that acts as a wrapper around the LaTeX package. The module loads PDFLaTeX from within ConTeXt to create a stand-alone PDF file from the relevant drawing that is then included into the ConTeXt document. To load the module in ConTeXt, call `\usemodule[hawkdraw]`.

\starthawkdraw[‹options›]
\stophawkdraw

The drawing code is placed between the commands `\starthawkdraw` and `\stophawkdraw`.

```
\usemodule[hawkdraw]

\starttext
  \starthawkdraw
    \path[stroke]{ (0cm,0cm) -- (2cm,1cm) }
  \stophawkdraw
\stoptext
```

The `\starthawkdraw` command can take options as argument. All options that are allowed for `\setupframed` can be used. Also, the option `options` can be used to pass options as described above for the `hawkdraw` environment. For example, the below code can be used to set layers:

```
\starthawkdraw[options={layers={background,main}}]
  \fill[color fill={blue}]{
    (0cm,0cm) circle[radius={0.5cm}]
  }
  \begin{scope}[layer={background}]
    \fill{
      (0.5cm,0cm) circle[radius={0.5cm}]
    }
  \end{scope}
\stophawkdraw
```

\starthawkdrawsoftmask[‹string›][‹options›]
\stophawkdrawsoftmask

Using the macros `\starthawkdrawsoftmask` and `\stophawkdrawsoftmask` a soft mask can be defined. The two commands and the two arguments work in the same way as the environment `hawkdrawsoftmask` as described above.

\setuphawkdraw[‹options›]

Instead of locally passing options to the `\starthawkdraw` command, it is also possible to pass options to the `\setuphawkdraw` command.

18 Changes

v0.0.5 (2026/05/18) First public alpha release.

v0.0.6 (2026/05/20) Splitting up code into modules.

v0.0.7 (2026/05/25) Adding patterns; adding `at part` option.

v0.0.8 (2026/05/30) Improving mapping functionality; adding mechanism to place arrow heads on paths; adding `at part` option for circles, ellipses and rectangles; adding `sloped` option for nodes; adding support for referencing anchors of nodes; adding possibility to reference points in calculations.

v0.1.0 (2026/06/03) Adding tagging support; adding basic intersection commands; adding plot functions; fixing node baseline anchors and slopes.

v0.1.1 (2026/06/05) Fixing bug in arrow tip slopes; fixing bugs in tagging support; improving option setting; adding option to set bounding box; adding ConTeXt module.

v0.2.0 (2026/06/12) Adding softpath system; adding option to shorten paths; improving ConTeXt module.

v0.3.0 (2026/06/17) Adding option to offset paths; fixing a bug with positioning of points on arcs.

v0.3.1 (2026/06/24) Allowing for multiple softpath manipulations; improved offset mechanism; added arrow tips and patterns.

v0.4.0 (2026/07/01) Adding shadings, opacity groups, blend modes and soft masks; adding support for referencing anchors of nodes by degree.

v0.4.1 (2026/07/08) Enhancing pattern support; adding conic shadings.

v0.4.2 (2026/07/13) Fixing bug in grayscale conic shading; fixing bug in radial shading; adding angle delta option to conic shadings.

v0.5.0 (2026/07/20) Fixing bug in pattern module; improving syntax parser; improving style setting mechanism; improving anchoring of nodes by degree; adding option to unset stroke and fill.

v0.6.0 (2026/07/27) Overall code clean-up; improving support for XeTeX, dvipdfmx.

v0.6.1 (2026/08/01) Fixing bug in pattern module; improving handling of blend modes and opacity groups.

v0.7.0 (2026/08/05) Adding option to apply patterns to strokes; adding pattern and shading support to nodes; adding support for text rendering modes; adding soft mask support to ConTeXt module.

v0.7.1 (2026/08/20) Improving pattern module; fixing bug in tagging support; adding bidi support for LuaTeX.

v0.8.0 (2026/09/01) Adding background to nodes; fixing bug in parser; fixing bug in bidi support.

v0.9.0 (2026/09/07) Adding option to apply shadings to strokes; fixing bug in pattern module; fixing bug in setting miter limit; fixing bug in positioning of node contents.

v0.9.1 (2026/10/12) Adding option for absolute positioning; adding expansion to parsing mechanism; refactoring for recent LaTeX3 kernel updates.

v0.10.0 (2026/10/13) Fixing bug in sloped nodes; improving pattern module; improving arrow tip module; adding support for cross-referencing nodes; adding commands to retrieve page dimensions.

v0.11.0 (2026/10/20) ⚠ **Breaking change:** Removing semicolon as terminal mark from the path syntax; changing syntax for points; fixing bug that prevents use of em units.

v0.11.1 (2026/11/10) Fixing bug in cross-referencing nodes; preparing for next release.

v0.11.2 (2026/11/13) Fixing bug in bounding box for tagging.