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 0.1.1, released on 5 June 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 epseically 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 L3 kernel is experimental as well and hence might change possibly breaking the functionality of this package. Although a row of fall-backs exist to ensure backward compatibility, in order to ensure the functionality of the `hawkdraw` 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.

2 Contents

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. It is recommended to use the package with an up-to-date TeX distribution.

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 key without a value is equivalent with setting the key 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 colon 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 cosnist 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 key. Note that a comma-separated list is typically not surrounded by parenthesis.
`options`	A key-value list consists of comma separated entities of keys 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.
none	Some keys do not accept any data type as they are only executing some function.

Keys in the `hawkdraw` package are divided into sets. Depending on where the keys are used in the document, only specific sets can be made use of. If a command or environment accepts specific sets of keys, 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 keys. If keys of other sets should be called, they need to be called from the root level.

For example keys of the `picture/` set can be called using `\hawkdrawset{picture/<key>}`.

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={0pt}
] \stroke[stroke color={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.

6 Scopes and layers

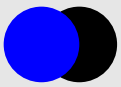
```
\begin{scope}[‹options›]
\end{scope}
```

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}[fill color={blue}]
  \fill (0cm,0cm) circle[radius={5pt}] ;
  \begin{scope}[fill color={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[fill color={blue}]
    (0cm,0cm) circle[radius={0.5cm}] ;
  \begin{scope}[layer={background}]
    \fill (0.5cm,0cm) circle[radius={0.5cm}] ;
  \end{scope}
\end{hawkdraw}
```

7 Path syntax

7.1 Path command and options

```
\path ‹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. Each `\path` command must end with a semicolon. 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 seocnd item can contain a comma to define two radii.

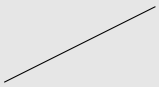
```
(‹string›)
```

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

7.3 Straight lines

```
-- (‹point›)
```

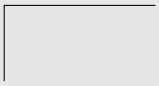
Using `--` a line to the next point is drawn.



```
\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.



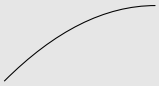
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    |- (2cm,1cm) ;
\end{hawkdraw}
```

```
\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).



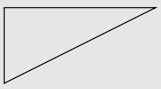
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    .. (1cm,1cm)
    .. (2cm,1cm) ;
\end{hawkdraw}
```

```
\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

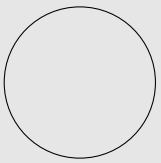
As for options, the following operators only accept the relevant subset to the `path/` set.

```
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}
```

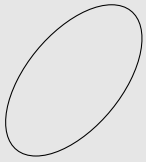
```
ellipse[‹options›]
```

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 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 with the options `radius x` and `radius y` that define the radii in the x- and y-dimension. It is possible to use the `radius` option to define a uniform radius for both dimensions. Alternatively, the options `vector a` and `vector b` can be used to specify the vectors of the arc as tuples of 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}
```

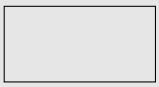
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    arc[
      vector a={(0.5cm,1cm)},
      vector b={(0.75cm,0cm)},
      angle start={45},
      angle end={135}
    ] ;
\end{hawkdraw}
```

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.



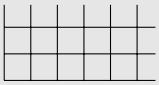
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    rectangle[corner={(2cm,1cm)}] ;
\end{hawkdraw}
```

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›}

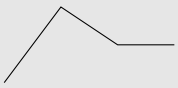
The plot function can be selected using the `plot/function` key that expects the name of a plot function as string. Currently, two plot functions are predefined, `sharp` and `smooth`, the latter being selected per default.

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.



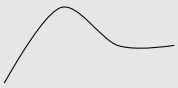
```
\begin{hawkdraw}
  \stroke[plot/function={sharp}] (0cm,0cm)
    plot { (0.75cm,1cm) (1.5cm,.5cm) (2.25cm,.5cm) } ;
\end{hawkdraw}
```

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 can be adjusted using the key `plot/smooth/tension` which is set to 0.15 by default



```
\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 exacct) 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.

7.8.1 Points

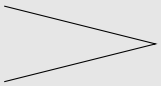
```
point[‹options›]
```

Using the `point` operation, a point at the current position is defined and named with the given name. As for options, the `point` operator only accepts the `path/point/` subset.

```
path/point/name={‹string›}
```

The point can be referenced later by using the name in parentheses (see above).

The command `\point` is a shortcut for `\path (0pt,0pt) point`

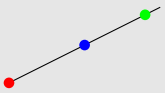


```
\begin{hawkdraw}
  \path (2cm,0.5cm) point[name={A}] ;
  \path[stroke] (0cm,0cm) -- (A) -- (0cm,1cm) ;
\end{hawkdraw}
```

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

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

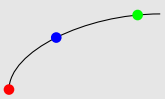
The position of a point can also be adjusted using the `at part` option which accespts a floating-point number that defines the position on the last path segment with 0 being the previous and 1 being the current point.



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

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

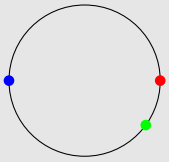
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    .. (1cm,1cm)
    .. (2cm,1cm)
    point[name={A}, at part={0}]
    point[name={B}, at part={0.5}]
    point[name={C}, at part={0.9}] ;
  \fill[fill color=red] (A) circle[radius={2pt}] ;
  \fill[fill color=blue] (B) circle[radius={2pt}] ;
  \fill[fill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



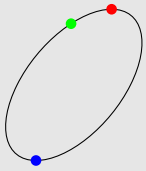
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    .. (0cm,0.5cm) & (1cm,1cm)
    .. (2cm,1cm)
    point[name={A}, at part={0}]
    point[name={B}, at part={0.5}]
    point[name={C}, at part={0.9}] ;
  \fill[fill color=red] (A) circle[radius={2pt}] ;
  \fill[fill color=blue] (B) circle[radius={2pt}]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

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

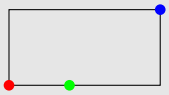
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    arc[
      vector a={(0.5cm,1cm)},
      vector b={(0.75cm,0cm)},
      angle start={45},
      angle end={135}
    ]
    point[name={A}, at part={0}]
    point[name={B}, at part={0.5}]
    point[name={C}, at part={0.9}] ;
  \fill[fill color=red] (A)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



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



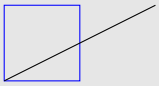
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    ellipse[
      vector a={(0.5cm,1cm)},
      vector b={(0.75cm,0cm)}
    ]
    point[name={A}, at part={0}]
    point[name={B}, at part={0.5}]
    point[name={C}, at part={0.9}] ;
  \fill[fill color=red] (A) circle[radius={2pt}] ;
  \fill[fill color=blue]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



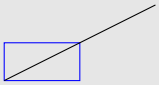
```
\begin{hawkdraw}
  \path[stroke] (0cm,0cm)
    rectangle[
      corner={(2cm,1cm)}
    ]
    point[name={A}, at part={0}]
    point[name={B}, at part={0.5}]
    point[name={C}, at part={0.9}] ;
  \fill[fill color=red] (A) circle[radius={2pt}] ;
  \fill[fill color=blue] (B) circle[radius={2pt}] ;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

\getpoint{‹point›} ★

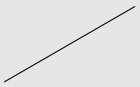
Inside a `hawkdraw` environment, the command `\getpoint` can be used to access the coordinates of a point on the current path. The point is given as an argument. The command returns the coordinates of the point as a tuple of dimensions and can be used in subsequent calculations.



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



```
\begin{hawkdraw}
  \point[name={A}, at={(1cm,0.5cm)}] ;
  \stroke[stroke color={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.

\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.

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 ccodes. 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/anchor={‹clist›}
```

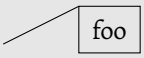
The node can be named by setting the `name` option which can be referenced later by using the name in parentheses (see above). Node names should not contain dots or dashes as these are reserved for referencing anchors. The position of the node can be adjusted with the `at` option which accepts a point (a tuple) or with the `at part` option which accespts a floating-point number (see above). The anchor 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 concatenated with a dash and separated by a dot from the node name. 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)`.

foo

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

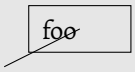

```
\begin{hawkdraw}
  \node[stroke, name={A}] {foo} ;
  \fill[fill color=red] (A.b-l) circle[radius={2pt}] ;
  \fill[fill color=red] (A.t-r) circle[radius={2pt}] ;
\end{hawkdraw}
```




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

path/node/**width**={‹dimension›}

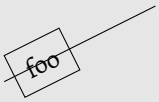
The width of the node can be set with the `width` option.




```
\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`.



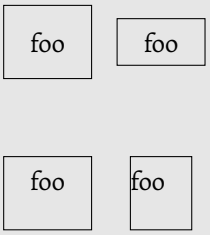

```
\begin{hawkdraw}
  \stroke (0cm,0cm) -- (2cm,1cm)
    node[
      stroke,
      at part={0.25},
      sloped
    ] {foo} ;
\end{hawkdraw}
```

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

Using the `text color` option, the color of the node contents is set. 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/frame={‹string›}
```

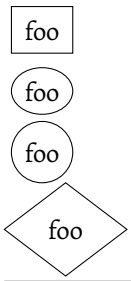
Using the `frame` command, the frame surrounding the node text can be selected. The `rectangle` frame is set per default. All options that are available for paths are also available for node frames.


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

The command `\node` is a shortcut for `\path (0pt,0pt) 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`



`rectangle`

`ellipse`

`circle`

`diamond`

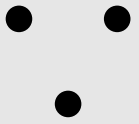
8 Mapping operation

```
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›}
```

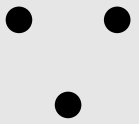
The option `list` sets the command to map over a comma-separated list of items. Each item in the list is passed as an argument to the code.



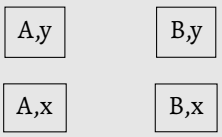
```
\begin{hawkdraw}
  \fill
    map[list={30,150,270}] {
      (0.75cm>#1) circle[radius={5pt}]
    } ;
\end{hawkdraw}
```

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` is a shortcut for `\path (0pt,0pt) map`



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




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

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

\hawkdrawmap[‹option›]{‹code›}

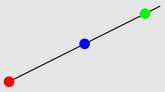
The command `\hawkdrawmap` can be used to repeat whole paths. The syntax is the same as for the `map` opation.

\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}
  \path[stroke] (0cm,0cm)
    -- (2cm,1cm)
    map[list={A,B,C}] {
      point[
        name={#1},
        at part={\hawkdrawarrayitem{myparts}{#2}}
      ]
    } ;
  \hawkdrawmap[list={A,B,C}] {
    \fill[
      fill color={\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 fill color, opacity, clipping

```
path/stroke
path/fill
path/clip
```

These options add a stroke to the path, fill the path, or use the path as a clipping region for subsequent paths. They can be used in combination. As the clipping region affects all following paths, it is often useful to use it in combination with scopes (see above).

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

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

The command `\clip` is a shortcut 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/stroke color={‹color›}
path/fill color={‹color›}
path/stroke opacity={‹float›}
path/fill opacity={‹float›}
```

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

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

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

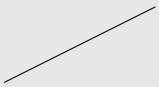
The options `stroke opacity` and `fill opacity` set the stroke opacity and the fill opacity for the path. The floating-point number should be in the range of 0 and 1. The default stroke opacity and fill opacity is 1 (fully opaque).

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`.

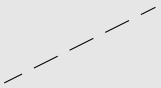
The option `miter limit` sets the miter limit for the path. The default miter limit is 10.

```
\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.

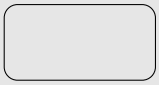


```
\begin{hawkdraw}
  \stroke[
    dash pattern={10pt,5pt},
    dash phase={2.5pt}
  ] (0cm,0cm) -- (2cm,1cm) ;
\end{hawkdraw}
```

9.3 Rounded corners and full 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›}
```

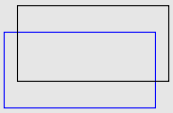
This option sets the fill rule for the path. The possible values are `non-zero` and `even-odd`. The default fill rule is `non-zero`. Note that this option is not influenced by scoping.

```
\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).

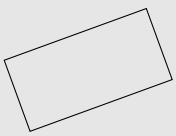


```
\begin{hawkdraw}
  \stroke[stroke color={blue}]
    (0cm,0cm) rectangle[corner={(2cm,1cm)}] ;
  \stroke[shift={(5pt,10pt)}]
    (0cm,0cm) rectangle[corner={(2cm,1cm)}] ;
\end{hawkdraw}
```

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

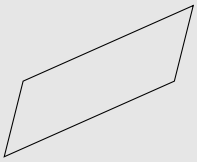
```
\begin{hawkdraw}
  \stroke[scale={0.25,0.5}]
    (0cm,0cm) rectangle[corner={(2cm,1cm)}] ;
\end{hawkdraw}
```

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[rotate={20}]
    (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[slant={0.25,0.5}]
    (0cm,0cm) rectangle[corner={(2cm,1cm)}] ;
\end{hawkdraw}
```

10 Defining custom styles and colors

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

To simplify styling, custom keys can be defined via the keys `style set` and `style add`. Both keys accept as value a key-value list consisting of one or multiple custom keys 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 key in the definition of another custom key.

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

```
\begin{hawkdraw}[
  style set={
    mystyle={
      stroke, stroke color={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.

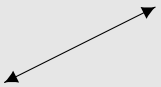
```
\hawkdrawsetcolor{colorA}{rgb}{0.1,0.5,0.8}
\hawkdrawsetcolor{colorB}{named}{yellow!90!red}
\begin{hawkdraw}[line width={5pt}]
  \path[
    stroke, stroke color={colorA},
    fill, fill color={colorB}
  ] (0cm,0cm) rectangle[corner={(2cm,1cm)}] ;
\end{hawkdraw}
```

11 Arrow tips

11.1 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.

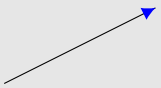


```
\begin{hawkdraw}
  \stroke[arrow start, arrow end]
    (0cm,0cm) -- (2cm,1cm) ;
\end{hawkdraw}
```

```
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›}
```

Arrow tips can be styled using 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.



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

The following arrow tips are available, of these the last three, `simple`, `straight` and `bar`, should be set together with the option `arrow style set={stroke}`:

Arrow tip	Name
	`none`
	`default`
	`stealth`
	`triangle`
	`circle`
	`simple`
	`straight`
	`bar`

11.2 Arrow tips on paths

tip[‹options›] {‹arrow›}

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

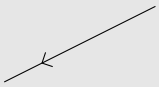
path/tip/**at part**={‹float›}

The position of the arrow head can be adjusted with the `at part` option which accespts a floating-point number (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}
```

12 Patterns

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

path/**fill pattern**={‹string›}

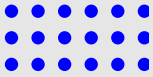
The option `fill pattern` sets a fill pattern for the current path. The key takes as value a string representing the name of the relevant pattern.

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

Patterns can be styled using 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.

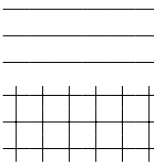
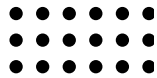
path/**pattern x**={‹dimension›}
path/**pattern y**={‹dimension›}

The options `pattern x` and `pattern y` set the dimensions of the pattern in the x and y directions, respectively. The default dimensions of a pattern in both directions are both 10 pt.



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

The following patterns are available:

Pattern	Name
	`lines`
	`hatch`
	`dots`

13 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, the setting `tagging-setup={alt={Water (H20)}, tag={Formula}}` would tag the `hawkdraw` environment as a block element with the tag `Formula` and set the replacement text to “Water (H20)”.

14 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]`. The drawing code can the be used between the commands `\starthawkdraw` and `\stophawkdraw`.

```
\usemodule[hawkdraw]

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

15 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.