

The l3draw package

Core drawing support

The L^AT_EX Project*

Released 2025-06-30

1 l3draw documentation

The l3draw package provides a set of tools for creating (vector) drawings in `expl3`. It is heavily inspired by the `pgf` layer of the `TikZ` system, with many of the interfaces having the same form. However, the code provided here is build entirely on core `expl3` ideas and uses the L^AT_EX3 FPU for numerical support.

Numerical expressions in l3draw are handled as floating point expressions, unless otherwise noted. This means that they may contain or omit explicit units. Where units are omitted, they will automatically be taken as given in (T_EX) points.

The code here is *highly* experimental.

*E-mail: latex-team@latex-project.org

1.1 Drawings

```
\draw_begin: \draw_begin:  
\draw_end:   ...  
           \draw_end:
```

Each drawing should be created within a `\draw_begin:/\draw_end:` function pair. The `begin` function sets up a number of key data structures for the rest of the functions here: unless otherwise specified, use of `\draw_...` functions outside of this “environment” is *not supported*.

The drawing created within the environment will be inserted into the typesetting stream by the `\draw_end:` function, which will switch out of vertical mode if required.



```
\dim_new:N \l_mypos_dim  
\draw_begin:  
  \draw_path_moveto:n { 0cm , \l_mypos_dim }  
  \draw_path_lineto:n { 1cm , \l_mypos_dim }  
  \dim_set:Nn \l_mypos_dim { 1cm }  
  \draw_path_lineto:n { 1cm , \l_mypos_dim }  
  \draw_path_close:  
  \draw_path_use_clear:n { stroke }  
\draw_end:
```

Within a drawing, the only commands that should appear are those directly aimed at drawing (from `l3draw`) and those which produce *no* typeset output. For example, it is possible to include loops inside a drawing using `\int_step_function:nnnn` or similar. On the other hand, text should not be included directly in drawings, but should rather be inserted using the appropriate `l3draw` command.

The drawing environment sets the following standard behaviors

- Non-zero rule for fill overlaps
- Butt caps for lines
- Mitering for line joins with a miter factor of 10
- Solid line strokes

Within a drawing, there are different ways of referring to a position. The coordinates of a point are given relative to the current *drawing axes*. These can be manipulated and tracked at the code level. Underlying this is the `<canvas>`, which is at the `draw` level essentially fixed and in line with the paper. Initially, the two sets of axes are coincident. (It is possible to manipulate the canvas axes at the driver level: this is then “transparent” to the `draw` level, and so should be used only when strictly required.)

The bounding box of the drawing is determined by tracking the size of the `draw` commands between the start and end. The bounding box is (roughly) the smallest box that contains all of the coordinates mentioned within the drawing. This can include those automatically generated, for example the supporting points needed to construct an arc.

`\draw_set_baseline:n` `\draw_set_baseline:n {⟨length⟩}`

As standard, the baseline of the bounding box of a drawing is calculated automatically at the bottom of the drawing. It is possible to adjust this using the `\draw_set_baseline:n` function. If the drawing coordinates lead to lower y -axis values than the `⟨length⟩`, then the drawing will have a depth as well as a height.

```

text
\draw_begin:
  \draw_path_rectangle:nn { 0 , 0 } { 2ex , 1ex }
  \draw_path_use:n { stroke }
\draw_end:
text
\draw_begin:
  \draw_path_rectangle:nn { 0 , 1ex } { 2ex , 1ex }
  \draw_set_baseline:n { 0pt }
  \draw_path_use:n { stroke }
\draw_end:
text
\draw_begin:
  \draw_path_rectangle:nn { 0 , -1ex } { 2ex , 1ex }
  \draw_set_baseline:n { -0.5ex }
  \draw_path_use:n { stroke }
\draw_end:
text

```

`\draw_suspend_begin:` `\draw_suspend_begin:`
`\draw_suspend_end:` `...`

`\draw_suspend_end:`

Suspends all of the drawing mechanisms to allow “normal” material to be created. Typically, this environment will be applied inside a box which may contain nested pictures.

```

\draw_begin:
  \draw_path_moveto:n { 0cm , 0cm }
  \hbox_set:Nn \l_tmpa_box
  {
    \draw_suspend_begin:
      This~is~normal~text.
      \draw_begin: % A subpicture
        \draw_path_moveto:n { 1cm , 0cm }
        \draw_path_lineto:n { 1cm , 1cm }
        \draw_path_use_clear:n { stroke }
      \draw_end:
        More~text.
    \draw_suspend_end:
  }
  \draw_box_use:N \l_tmpa_box
  \draw_path_lineto:n { 0cm , 1cm }
  \draw_path_use_clear:n { stroke }
\draw_end:

```

`\g_draw_id_int` Each drawing is given a unique identity: this is incremented each time `\draw_begin:` is used.

1.2 Setting the graphics state

Within the drawing environment, a number of functions control how drawings will appear. Note that these all apply *globally*, though some are reset at the start of each drawing (`\draw_begin:`).

`\l_draw_default_linewidth_dim`

The default value of the linewidth for stokes, set at the start of every drawing (`\draw_begin:`).

`\draw_set_linewidth:n` `\draw_set_linewidth:n {width}`

Sets the width to be used for stroking to the *width* (an *fp expr*).

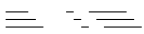
`\draw_set_dash_pattern:nn` `\draw_set_dash_pattern:nn {pattern} {phase}`

Specifies a dash pattern. The *pattern* itself is a comma-separated list of entries which represent the “on” and “off” parts of the line. These are all *fp expr* and repeat as required. Thus the *pattern* may be of arbitrary length. The *phase* specifies where during the first “on” line the pattern should start.

```

\draw_begin:
  \draw_set_dash_pattern:nn
    { 0.5cm , 0.5cm , 0.1cm , 0.2cm }
    { 0cm }
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_lineto:n { 2cm , 0cm }
  \draw_path_use_clear:n { stroke }
  \draw_set_dash_pattern:nn
    { 0.5cm , 0.5cm , 0.1cm , 0.2cm }
    { 0.1cm }
  \draw_path_moveto:n { 0cm , 1mm }
  \draw_path_lineto:n { 2cm , 1mm }
  \draw_path_use_clear:n { stroke }
  \draw_set_dash_pattern:nn
    { 0.5cm , 0.5cm , 0.1cm , 0.2cm }
    { 0.2cm }
  \draw_path_moveto:n { 0cm , 2mm }
  \draw_path_lineto:n { 2cm , 2mm }
  \draw_path_use_clear:n { stroke }
\draw_end:

```



Setting an empty pattern will produce a solid line.

Note the *pattern* interface here is different from that in `pgf`: the list is comma-separated not given in brace groups.

<code>\draw_set_nonzero_rule:</code>	<code>\draw_set_nonzero_rule:</code>
<code>\draw_set_evenodd_rule:</code>	Active either the non-zero winding number or the even-odd rule, respectively, for determining what is inside a fill or clip area. For technical reasons, these command are not influenced by scoping and apply on an ongoing basis.

<code>\draw_set_cap_but:</code>	<code>\draw_set_cap_but:</code>
<code>\draw_set_cap_rectangle:</code>	Sets the style of terminal stroke position to one of butt, rectangle or round.
<code>\draw_set_cap_round:</code>	

<code>\draw_set_join_bevel:</code>	<code>\draw_set_join_miter:</code>
<code>\draw_set_join_miter:</code>	Sets the style of stroke joins to one of bevel, miter or round.
<code>\draw_set_join_round:</code>	

<code>\draw_set_miterlimit:n</code>	<code>\draw_set_miterlimit:n</code> $\{\langle factor \rangle\}$
	Sets the miter $\langle factor \rangle$ of lines joined as a miter, as described in the PDF and PostScript manuals. The $\langle factor \rangle$ is an $\langle fp\ expr \rangle$.

1.3 Scoping drawing elements

Scoping drawing elements is necessary to allowing nesting of subparts. These features have specific use requirements: the preconditions must be met. In particular, each type of drawing scope also constitutes a group (`\group_begin:/\group_end:` pair): as such, they must be nested correctly.

```

\draw_scope_begin: \draw_scope_begin:
\draw_scope_end:   ...
\draw_scope_end:

```

Creates a scope for localization of state settings within a drawing. A scope forms a \TeX group but will also localize global state variables (such as `\l_draw_default_linewidth_dim`), and driver-level concepts such as the termination of lines.

```

\draw_begin:
\draw_scope_begin:
\group_begin:
\draw_set_linewidth:n { 2pt }
\draw_path_rectangle:nn { 0 , 0 } { 2ex , 2ex }
\draw_path_use:n { stroke }
□□□
\group_end:
\draw_path_rectangle:nn { 3ex , 0ex } { 2ex , 2ex }
\draw_path_use:n { stroke }
\draw_scope_end:
\draw_path_rectangle:nn { 6ex , 0ex } { 2ex , 2ex }
\draw_path_use_clear:n { stroke }
\draw_end:

```

Global graphical concepts restricted by scope are

- Line width
- Stroke and fill color
- Dash pattern
- Line joining and capping, including the miter limit
- Clipping paths
- Canvas (driver) transformations

1.4 Points

Functions supporting the calculation of points (coordinates) are expandable and may be used outside of the drawing environment. The outputs of all of the point functions are tuples. This output form is then suitable as *input* for subsequent point calculations, i.e., where a *(point)* is required it may be given as a tuple. This *may* include units and surrounding parentheses, for example

```

1,2
(1,2)
1cm,3pt
(1pt,2cm)
2 * sind(30), 2^4in

```

are all valid input forms. Notice that each part of the tuple may itself be a float point expression.

Point coordinates are relative to the canvas axes, but can be transformed by `\draw_point_transform:n`. These manipulation is applied by many higher-level functions, for

example path construction, and allows parts of a drawing to be rotated, scaled or skewed. This occurs before writing any data to the driver, and so such manipulations are tracked by the drawing mechanisms. See `\@@_backend_cm:nnn` for backend-level manipulation of the canvas axes themselves.

Notice that in contrast to `pgf` it is possible to give the positions of points *directly*.

1.4.1 Basic point functions

```
\draw_point_polar:nn * \draw_point_polar:nn {<radius>} {<angle>}
\draw_point_polar:nnn * \draw_point_polar:nnn {<radius-a>} {<radius-b>} {<angle>}
```

Gives the coordinates of the point at `<angle>` (an `<fp expr>` in *degrees*) and `<radius>`. The three-argument version accepts two radii of different lengths.

Note the interface here is somewhat different from that in `pgf`: the one- and two-radii versions in `l3draw` use separate functions, whilst in `pgf` they use the same function and a keyword.

```
\draw_point_unit_vector:n * \draw_point_unit_vector:n {<point>}
```

Expands to the coordinates of a unit vector in the direction of the `<point>` from the origin. If the `<point>` is at the origin, a vertical unit vector is returned

```
\draw_point_transform:n * \draw_point_transform:n {<point>}
```

Evaluates the position of the `<point>` subject to the current transformation matrix. This operation is applied automatically by most higher-level functions (*e.g.* path manipulations).

1.4.2 Points on a vector basis

As well as giving explicit values, it is possible to describe points in terms of underlying direction vectors. The latter are initially co-incident with the standard Cartesian axes, but may be altered by the user.

```
\draw_xvec:n \draw_xvec:n {<point>}
```

```
\draw_yvec:n
```

```
\draw_zvec:n
```

Defines the appropriate base vector to point toward the `<point>` on the canvas. The standard settings for the *x*- and *y*-vectors are 1 cm along the relevant canvas axis, whilst for the *z*-vector an appropriate direction is taken.

```
\draw_point_vec:nn * \draw_point_vec:nn {<xscale>} {<yscale>}
\draw_point_vec:nnn * \draw_point_vec:nnn {<xscale>} {<yscale>} {<zscale>}
```

Expands to the coordinate of the point at `<xscale>` times the *x*-vector and `<yscale>` times the *y*-vector. The three-argument version extends this to include the *z*-vector.

```

\draw_point_vec_polar:nn * \draw_point_vec_polar:nn {<radius>} {<angle>}
\draw_point_vec_polar:nmn * \draw_point_vec_polar:nmn {<radius-a>} {<radius-b>} {<angle>}

```

Gives the coordinates of the point at $\langle angle \rangle$ (an $\langle fp \ expr \rangle$ in *degrees*) and $\langle radius \rangle$, relative to the prevailing x - and y -vectors. The three-argument version accepts two radii of different lengths.

Note the interface here is somewhat different from that in `pgf`: the one- and two-radii versions in `l3draw` use separate functions, whilst in `pgf` they use the same function and a keyword.

1.4.3 Intersections

```

\draw_point_intersect_lines:nmnn * \draw_point_intersect_lines:nmnn {<point1>} {<point2>} {<point3>}
\draw_point_intersect_lines:nmnn * \draw_point_intersect_lines:nmnn {<point4>}

```

Evaluates the point at the intersection of one line, joining $\langle point1 \rangle$ and $\langle point2 \rangle$, and a second line joining $\langle point3 \rangle$ and $\langle point4 \rangle$. If the lines do not intersect, or are coincident, and error will occur.

```

\draw_point_intersect_circles:nmnm * \draw_point_intersect_circles:nmnm
\draw_point_intersect_circles:nmnm * \draw_point_intersect_circles:nmnm {<center1>} {<radius1>} {<center2>} {<radius2>} {<root>}

```

Evaluates the point at the intersection of one circle with $\langle center1 \rangle$ and $\langle radius1 \rangle$, and a second circle with $\langle center2 \rangle$ and $\langle radius2 \rangle$. If the circles do not intersect, or are coincident, and error will occur.

Note the interface here has a different argument ordering from that in `pgf`, which has the two centers then the two radii.

```

\draw_point_intersect_line_circle:nmnm * \draw_point_intersect_line_circle:nmnm
\draw_point_intersect_line_circle:nmnm * \draw_point_intersect_line_circle:nmnm {<point1>} {<point2>} {<center>} {<radius>} {<root>}

```

Evaluates the point at the intersection of one line, joining $\langle point1 \rangle$ and $\langle point2 \rangle$, and a circle with $\langle center \rangle$ and $\langle radius \rangle$. If the lines and circle do not intersect and error will occur.

1.4.4 Interpolations

```

\draw_point_interpolate_line:nnn * \draw_point_interpolate_line:nnn {<part>} {<point1>} {<point2>}

```

Expands to the point which is $\langle part \rangle$ way along the line joining $\langle point1 \rangle$ and $\langle point2 \rangle$. The $\langle part \rangle$ may be an interpolation or an extrapolation, and is a floating point value expressing a percentage along the line, *e.g.* a value of 0.5 would be half-way between the two points.

```

\draw_point_interpolate_distance:nnn * \draw_point_interpolate_distance:nnn {<distance>} {<point expr1>}
\draw_point_interpolate_distance:nnn * \draw_point_interpolate_distance:nnn {<point expr2>}

```

Expands to the point which is $\langle distance \rangle$ way along the line joining $\langle point1 \rangle$ and $\langle point2 \rangle$. The $\langle distance \rangle$ may be an interpolation or an extrapolation.

```
\draw_point_interpolate_curve:nnnnnn * \draw_point_interpolate_curve:nnnnnn {<part>}  
{\<start>} {\<control1>} {\<control2>} {\<end>}
```

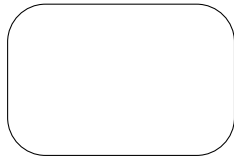
Expands to the point which is $\langle part \rangle$ way along the curve between $\langle start \rangle$ and $\langle end \rangle$ and defined by $\langle control1 \rangle$ and $\langle control2 \rangle$. The $\langle part \rangle$ may be an interpolation or an extrapolation, and is a floating point value expressing a percentage along the curve, *e.g.* a value of 0.5 would be half-way along the curve.

1.5 Paths

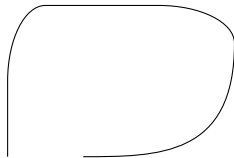
Paths are constructed by combining one or more operations before applying one or more actions. Thus until a path is “used”, it may be manipulated or indeed discarded entirely. Only one path is active at any one time, and the path is *not* affected by T_EX grouping.

`\draw_path_corner_arc:nn` `\draw_path_corner_arc:nn {<length1>} {<length2>}`

Sets the degree of rounding applied to corners in a path: the two `<length>` values are the distances from the corner at which the curving should start. The first `<length>` applies to the part of the path “leading in” to the corner (i.e., from the previous path operation), and the second to that “leading out”. If both values are `0pt` then corners will not be rounded. The values apply within the scope of the current `TEX` group.

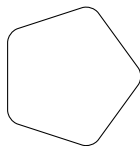


```
\draw_begin:
  \draw_path_corner_arc:nn { 5mm } { 5mm }
  \draw_path_rectangle_corners:nn
    { 0cm , 0cm } { 3cm , 2cm }
  \draw_path_use_clear:n { stroke }
\draw_end:
```



```
\draw_begin:
  \draw_path_corner_arc:nn { 10mm } { 5mm }
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_lineto:n { 0cm , 2cm }
  \draw_path_lineto:n { 3cm , 2cm }
  \draw_path_curveto:nnn
    { 3cm , 0cm } { 2cm , 0cm } { 1cm , 0cm }
  \draw_path_use_clear:n { stroke }
\draw_end:
```

The corners created are quarter-circles for exact right-angles and close approximations otherwise. Closing a path will result in rounding correctly.



```
\draw_begin:
  \draw_path_corner_arc:nn { 4pt } { 4pt }
  \draw_path_moveto:n
    { \draw_point_polar:nn { 1cm } { 0 } }
  \int_step_inline:nnnn { 72 } { 72 } { 359 }
  {
    \draw_path_lineto:n
      { \draw_point_polar:nn { 1cm } { #1 } }
  }
  \draw_path_close:
  \draw_path_use_clear:n { stroke }
\draw_end:
```

`\draw_path_close:` `\draw_path_close:`

Closes the current part of the path by appending a straight line from the current point to the starting point of the path. In general, any path to be *filled* should include a close instructions.

`\draw_path_use:n` `\draw_path_use:n {⟨action(s)⟩}`

`\draw_path_use_clear:n` Inserts the current path, carrying out one or more possible *⟨actions⟩* (a comma list):

- **stroke** Draws a line along the current path
- **draw** A synonym for **stroke**
- **fill** Fills the interior of the path with the current fill color
- **clip** Clips any content outside of the path

Actions are applied in the order given irrespective of the input order. Note that whilst it is possible to use a path without clearing it, the normal use case would be to clear the path (this resets data structures at the macro level).

`\draw_path_replace_bb:` `\draw_path_replace_bb:`

Replaces the current bounding box of the drawing with one specified by the current path: this will be applied even when `\l_draw_bb_update_bool` is **false**. The current path is then cleared. Note that `\l_draw_bb_update_bool` is *not* changed by this function: the user may wish to set it to **false** so that the bounding box is then left unchanged by further operations.

1.5.1 Path operations on drawing axes


The standard path functions are all influenced by the active transformation matrix, i.e., the work relative to the drawing axes rather than the canvas.

`\draw_path_moveto:n` `\draw_path_moveto:n {⟨point⟩}`


Moves the reference point of the path to the *⟨point⟩*, but will not join this to any previous point.

`\draw_path_lineto:n` `\draw_path_lineto:n {⟨point⟩}`

Joins the current path to the `⟨point⟩` with a straight line. In general, for reliable treatment by viewers, a `\draw_path_moveto:n` operation should precede the first use of a `\draw_path_lineto:n` on a path.




```
\draw_begin:  
  \draw_path_moveto:n { 0cm , 0cm }  
  \draw_path_lineto:n { 1cm , 1cm }  
  \draw_path_lineto:n { 2cm , 1cm }  
  \draw_path_lineto:n { 3cm , 0.5cm }  
  \draw_path_lineto:n { 3cm , 0cm }  
  \color_fill:n { yellow!80!black }  
  \draw_path_use_clear:n { fill , stroke }  
\draw_end:
```



```
\draw_begin:  
  \draw_path_moveto:n { 0cm , 0cm }  
  \draw_path_lineto:n { 1cm , 1cm }  
  \draw_path_lineto:n { 2cm , 1cm }  
  \draw_path_moveto:n { 2cm , 1cm } % Begins a new part  
  \draw_path_lineto:n { 3cm , 0.5cm }  
  \draw_path_lineto:n { 3cm , 0cm }  
  \color_fill:n { yellow!80!black }  
  \draw_path_use_clear:n { fill , stroke }  
\draw_end:
```

`\draw_path_curveto:nnn` `\draw_path_curveto:nnn {⟨control1⟩} {⟨control2⟩} {⟨end⟩}`


Joins the current path to the `⟨end⟩` with a curved line defined by cubic Bézier points `⟨control1⟩` and `⟨control2⟩`. The bounding box of the path (and image) will fully-contain the curve and control points, i.e., it may be bigger than strictly necessary to contain the curve *alone*.



```
\draw_begin:  
  \draw_path_moveto:n { 0cm , 0cm }  
  \draw_path_curveto:nnn  
    { 1cm , 1cm } % First control  
    { 2cm , 1cm } % Second control  
    { 3cm , 0cm } % End  
  \color_fill:n { yellow!80!black }  
  \draw_path_use_clear:n { fill , stroke }  
\draw_end:
```

`\draw_path_curveto:nn` `\draw_path_curveto:nn {<control>} {<end>}`

Joins the current path to the `<end>` with a curved line defined by quadratic Bézier point `<control>`. The bounding box of the path (and image) will fully-contain the curve and computed (cubic) control points, i.e., it may be bigger than strictly necessary to contain the curve *alone*.




```
\draw_begin:
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_curveto:nn
    { 1cm , 1cm }
    { 2cm , 0cm }
  \color_fill:n { yellow!80!black }
  \draw_path_use_clear:n { fill , stroke }
\draw_end:
```

`\draw_path_arc:nnn` `\draw_path_arc:nnn {<angle1>} {<angle2>} {<radius>}`


`\draw_path_arc:nmmn` `\draw_path_arc:nmmn {<angle1>} {<angle2>} {<radius-a>} {<radius-b>}`

Joins the current path with an arc between `<angle1>` and `<angle2>` and of `<radius>`. The four-argument version accepts two radii of different lengths.



```
\draw_begin:
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_lineto:n { 0cm , 1cm }
  \draw_path_arc:nnn { 180 } { 90 } { 0.5cm }
  \draw_path_lineto:n { 3cm , 1.5cm }
  \draw_path_arc:nnn { 90 } { -45 } { 0.5cm }
  \draw_path_use_clear:n { fill }
\draw_end:
```

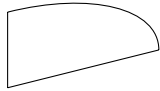
Note the interface here has a different argument ordering from that in `pgf`, which has the two centers then the two radii.



```
\draw_begin:
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_arc:nmmn { 180 } { 45 } { 2cm } { 1cm }
  \draw_path_use_clear:n { stroke }
\draw_end:
```

`\draw_path_arc_axes:nmm` `\draw_path_arc_axes:nnn` $\langle angle1 \rangle$ $\langle angle2 \rangle$ $\langle vector1 \rangle$ $\langle vector2 \rangle$

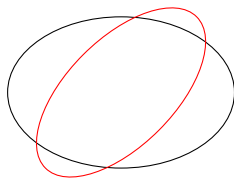
Appends the portion of an ellipse from $\langle angle1 \rangle$ to $\langle angle2 \rangle$ of an ellipse with axes along $\langle vector1 \rangle$ and $\langle vector2 \rangle$ to the current path.



```
\draw_begin:
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_lineto:n { 2cm , 5mm }
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_lineto:n { 0cm , 1cm }
  \draw_path_moveto:n { 2cm , 5mm }
  \draw_path_arc_axes:nmm { 0 } { 90 }
    { 2cm , 5mm } { 0cm , 1cm }
  \draw_path_use_clear:n { stroke }
\draw_end:
```

`\draw_path_ellipse:nm` `\draw_path_ellipse:nnn` $\langle center \rangle$ $\langle vector1 \rangle$ $\langle vector2 \rangle$

Appends an ellipse at $\langle center \rangle$ with axes along $\langle vector1 \rangle$ and $\langle vector2 \rangle$ to the current path. A new part is started if the path is already non-empty. Notice that the underlying drawing is constructed from arcs with appropriate moves: the interface is a more efficient convenience.



```
\draw_begin:
  \draw_path_ellipse:nnn
    { 1cm , 0cm }
    { 1.5cm , 0cm }
    { 0cm , 1cm }
  \draw_path_use_clear:n { stroke }
  \color_select:n { red }
  \draw_path_ellipse:nnn
    { 1cm , 0cm }
    { 1cm , 1cm }
    { -0.5cm , 0.5cm }
  \draw_path_use_clear:n { stroke }
\draw_end:
```

Note that any transformation is applied to the completed ellipse rather than to the axes.

`\draw_path_circle:nn` `\draw_path_circle:nn` $\langle center \rangle$ $\langle radius \rangle$

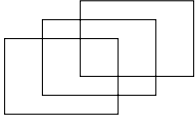

Appends a circle of $\langle radius \rangle$ at $\langle center \rangle$ to the current path. This is a shortcut for `\draw_path_ellipse:nnn`.

```

\draw_path_rectangle:nn      \draw_path_rectangle:nn {<lower-left>} {<displacement>}
\draw_path_rectangle_corners:nn \draw_path_rectangle_corners:nn {<lower-left>} {<top-right>}

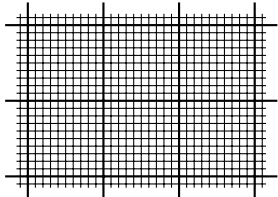
```

Appends a rectangle starting at *<lower-left>* to the current path, with the size of the rectangle determined either by a *<displacement>* or the position of the *<top-right>*.

	<pre> \draw_begin: \draw_path_rectangle:nn { 1cm , 0cm } { 1.5cm , 1cm } \draw_path_rectangle:nn { 1.5cm , 0.25cm } { 1.5cm , 1cm } \draw_path_rectangle:nn { 2cm , 0.5cm } { 1.5cm , 1cm } \draw_path_use_clear:n { draw } \draw_end: </pre>
	<pre> \draw_begin: \draw_path_rectangle_corners:nn { 1cm , 0cm } { 1.5cm , 1cm } \draw_path_use_clear:n { draw } \draw_end: </pre>

```
\draw_path_grid:nnnn <xstep> <ystep> <lower-left> <upper-right>
```

Constructs a grid of $\langle xstep \rangle$ and $\langle ystep \rangle$ inside the rectangle defined by the $\langle lower-left \rangle$ and the $\langle upper-right \rangle$, and appends this to the current path. The grid will be aligned such that grid lines pass through the origin, which may result in “protruding” ends if the start/end positions do not fully align.

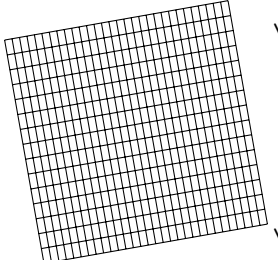


```

\draw_begin:
  \draw_set_linewidth:n { 0.8pt }
  \draw_path_grid:nnnn
    { 1cm } { 1cm }
    { -3mm , -3mm }
    { 33mm , 23mm }
  \draw_path_use_clear:n { stroke }
  \draw_set_linewidth:n { 0.4pt }
  \draw_path_grid:nnnn
    { 1mm } { 1mm }
    { -1.5mm , -1.5mm }
    { 31.5mm , 21.5mm }
  \draw_path_use_clear:n { stroke }
\draw_end:

```

Any transformation is applied to the finished grid.



```

\draw_begin:
  \draw_transform_rotate:n { 10 }
  \draw_path_grid:nnnn
    { 1mm } { 2mm }
    { 0mm , 0mm }
    { 30mm , 30mm }
  \draw_path_use_clear:n { stroke }
\draw_end:

```

1.5.2 Path scope

```

\draw_path_scope_begin: \draw_path_scope_begin:
\draw_path_scope_end:   ...
\draw_path_scope_end:

```

Suspends (and saves) the current (partial) path, initializing a new path within the scope. Path operations are written to output only when used, so the scoped path is stored at the `expl3` level, not in the output.

1.5.3 Path operations on canvas axes

For *specialist* work, a small number of functions are provided which work relative to the canvas axes, i.e., these functions ignore the transformation matrix.

```
\draw_path_canvas_moveto:n <canvas point>
```

Moves the reference point of the path to the $\langle canvas point \rangle$, but will not join this to any previous point.

`\draw_path_canvas_lineto:n` `\draw_path_canvas_lineto:n` `{\langle canvas point \rangle}`
Joins the current path to the `\langle canvas point \rangle` with a straight line.

`\draw_path_canvas_curveto:nnn` `\draw_path_canvas_curveto:nnn` `{\langle control1 \rangle}` `{\langle control2 \rangle}` `{\langle end \rangle}`
Joins the current path to the `\langle end \rangle` with a curved line defined by cubic Bézier points `\langle control1 \rangle` and `\langle control2 \rangle`. These positions are given as canvas points.

1.6 Bounding box

`\g_draw_bb_xmin_dim` The values for the corners of the automatically-calculated bounding box. Note that this
`\g_draw_bb_ymin_dim` is the apparent size of the drawing; material may be placed outside of the bounding box,
`\g_draw_bb_xmax_dim` see `\l_draw_bb_update_bool`.
`\g_draw_bb_ymax_dim`

`\l_draw_bb_update_bool` All functions automatically update the bounding box of the image, unless specified otherwise. This behavior is selectable using the `\l_draw_bb_update_bool` boolean.

In \LaTeX , the absolute position of the origin for each drawing is recorded as a property: the attributes `xpos`, `ypos` and `abspage` are saved. The drawings are identified as `draw.\langle id \rangle`, where `\langle id \rangle` is the value of `\g_draw_id_int`.

1.7 Boxes and coffins

```
\draw_box_use:N \draw_box_use:N <box>
\draw_box_use:Nn \draw_box_use:Nn <box> {<point>}
```

Inserts the `<box>` into a drawing, taking account of the current transformation matrix and shift, and adjusting the drawing bounding box to contain the (apparent) size of the box if this is active (see `\l_draw_bb_update_bool`).

```

\draw_begin:
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_lineto:n { 0cm , 1cm }
  \draw_path_use_clear:n { stroke }
  \hbox_set:Nn \l_tmpa_box
    { This~is~text. }
  \draw_box_use:N \l_tmpa_box
\draw_end:
```

This is text.

```
\draw_begin:
  \draw_transform_matrix_absolute:nmmm { 2 } { 0 } { 1 } { 2 }
  \draw_path_moveto:n { 0cm , 0cm }
  \draw_path_lineto:n { 0cm , 1cm }
  \draw_path_use_clear:n { stroke }
  \hbox_set:Nn \l_tmpa_box
    { This~is~text. }
  \draw_box_use:N \l_tmpa_box
\draw_end:
```

This is text.

If the `<point>` is given, the reference point of the box is placed there: otherwise it is positioned at the origin of the drawing.

<code>\draw_coffin_use:Nnn</code>	<code>\draw_coffin_use:Nnn <coffin> {<hpole>} {<vpole>}</code>
<code>\draw_coffin_use:Nnnn</code>	<code>\draw_coffin_use:Nnnn <coffin> {<hpole>} {<vpole>} {<point>}</code>

Inserts the `<coffin>` into a drawing, taking account of the current transformation matrix and shift, and adjusting the drawing bounding box to contain the (apparent) size of the box if this is active (see `\l_draw_bb_update_bool`). The alignment point of the coffin to the origin is specified by the intersection of the `<hpole>` and the `<vpole>`.

This is text.	<pre> \draw_begin: \draw_path_moveto:n { 0cm , 0cm } \draw_path_lineto:n { 0cm , 1cm } \draw_path_use_clear:n { stroke } \hcoffin_set:Nn \l_tmpa_coffin { This~is~text. } \draw_coffin_use:Nnn \l_tmpa_coffin { hc } { vc } \draw_end: </pre>
---------------	---

If the `<point>` is given, the pole intersection of the coffin is placed there: otherwise it is positioned at the origin of the drawing.

1.8 Transformations

Points are normally used unchanged relative to the canvas axes. This can be modified by applying a transformation matrix. The canvas axes themselves may be adjusted using `\@@_backend_cm:n`: note that this is transparent to the drawing code so is not tracked.

<code>\draw_transform_matrix:n</code>	<code>\draw_transform_matrix:n</code>
<code>\draw_transform_matrix_absolute:n</code>	<code>{<a>} {} {<c>} {<d>}</code>

Applies the transformation matrix $[\langle a \rangle \langle b \rangle \langle c \rangle \langle d \rangle]$. The basic applies the transformation in addition to those active; the `absolute` version overwrites any active transformation. This assignment is local.

<code>\draw_transform_shift:n</code>	<code>\draw_transform_shift:n {<vector>}</code>
<code>\draw_transform_shift_absolute:n</code>	

Applies the transformation `<vector>` to points. The basic applies the vector in addition to those active; the `absolute` version overwrites any active vector. Any active transformation matrix is applied to the shifts each time they are adjusted. This assignment is local.

<code>\draw_transform_triangle:n</code>	<code>\draw_transform_triangle:n</code>
<code>\draw_transform_triangle:n</code>	<code>{<origin>} {<point1>} {<point2>}</code>

Applies a transformation such that the coordinates (0,0), (1,0) and (0,1) are given by the `<origin>`, `<point1>` and `<point2>`, respectively. This assignment is local.

<code>\draw_transform_rotate:n</code>	<code>\draw_transform_rotate:n {<angle>}</code>
---------------------------------------	---

Applies a rotation by the `<angle>`, measured anti-clockwise in degrees. This rotation is *additional* to any prevailing transformation. This assignment is local.

`\draw_transform_scale:n` `\draw_transform_scale:n {⟨scale⟩}`
`\draw_transform_xscale:n`
`\draw_transform_yscale:n` Applies the `⟨scale⟩` in either `x` or `y` (or both). This scale is *added* to any prevailing transformation. This assignment is local.

`\draw_transform_xshift:n` `\draw_transform_xshift:n {⟨xshift⟩}`
`\draw_transform_yshift:n` Applies an `⟨xshift⟩` or `⟨yshift⟩`, as appropriate. This shift is *added* to any prevailing one. This assignment is local.

`\draw_transform_xslant:n` `\draw_transform_xslant:n {⟨slant⟩}`
`\draw_transform_yslant:n` Applies the `⟨slant⟩` (a factor) in either `x` or `y`. This slant is *added* to any prevailing transformation. This assignment is local.

`\draw_transform_matrix_invert:` `\draw_transform_matrix_invert:`
`\draw_transform_shift_invert:`

Inverts the current transformation matrix or shift vector, as appropriate. This assignment is local.

`\draw_transform_matrix_reset:` `\draw_transform_matrix_reset:`
`\draw_transform_shift_reset:`

Resets the current transformation matrix or shift vector, as appropriate. This assignment is local.

1.9 Layers

Drawing layers may be used to alter the way in which elements are stacked on top of one another. In particular, they are useful when the nature of an element depends on another, but where it needs to be behind its “parent”. A classic example is a filled background: this needs to know the size of the material it is behind.

All drawings feature a layer called `main`. This layer should always be present, and is the one which “non-layered” content is added to.

`\draw_layer_new:n` `\draw_layer_new:n {⟨layer⟩}`

Creates a new `⟨layer⟩` which can then be used in drawing. The layer `main` is pre-defined.

`\draw_layer_begin:n` `\draw_layer_begin:n {⟨layer⟩}`
`\draw_layer_end:` ...

`\draw_layer_end:`
Begin and end collection of material for the `⟨layer⟩`, which should previously have been created (and which cannot be `main`). Material collected for the layer is available globally within the current drawing.

`\l_draw_layers_clist` The list of active layers: may be given anywhere before `\draw_end:`. The `main` layer should always be included.

Index

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

Symbols

@@ commands:

`\@@_backend_cm:nmnn` *7, 19*

D

draw commands:

`\l_draw_bb_update_bool` *11, 17–19*
`\g_draw_bb_xmax_dim` *17*
`\g_draw_bb_xmin_dim` *17*
`\g_draw_bb_ymax_dim` *17*
`\g_draw_bb_ymin_dim` *17*
`\draw_begin:` *2, 4*
`\draw_box_use:N` *18*
`\draw_box_use:Nn` *18*
`\draw_coffin_use:Nnn` *19*
`\draw_coffin_use:Nmnn` *19*
`\l_draw_default_linewidth_dim` .. *4, 6*
`\draw_end:` *2, 20*
`\g_draw_id_int` *4, 17*
`\draw_layer_begin:n` *20*
`\draw_layer_end:` *20*
`\draw_layer_new:n` *20*
`\l_draw_layers_clist` *20*
`\draw_path_arc:nmn` *13*
`\draw_path_arc:nmnn` *13*
`\draw_path_arc_axes:nmn` *14*
`\draw_path_arc_axes:nmnn` *14*
`\draw_path_canvas_curveto:nmn` ... *17*
`\draw_path_canvas_lineto:n` *17*
`\draw_path_canvas_moveto:n` *16*
`\draw_path_circle:nn` *14*
`\draw_path_close:` *10*
`\draw_path_corner_arc:nm` *10*
`\draw_path_curveto:nmn` *12*
`\draw_path_ellipse:nmn` *14*
`\draw_path_grid:nmnn` *16*
`\draw_path_lineto:n` *12*
`\draw_path_moveto:n` *11, 12*
`\draw_path_rectangle:nn` *15*
`\draw_path_rectangle_corners:nn` . *15*
`\draw_path_replace_bb:` *11*
`\draw_path_scope_begin:` *16*
`\draw_path_scope_end:` *16*
`\draw_path_use:n` *11*
`\draw_path_use_clear:n` *11*
`\draw_point_interpolate_curve:nmnmnn`
..... *9*

`\draw_point_interpolate_distance:nmn`
..... *8*
`\draw_point_interpolate_line:nmn` . *8*
`\draw_point_intersect_circles:nmnmnn`
..... *8*
`\draw_point_intersect_line_-`
`circle:nmnmnn` *8*
`\draw_point_intersect_lines:nmnn` . *8*
`\draw_point_polar:nn` *7*
`\draw_point_polar:nmn` *7*
`\draw_point_transform:n` *6, 7*
`\draw_point_unit_vector:n` *7*
`\draw_point_vec:nn` *7*
`\draw_point_vec:nmn` *7*
`\draw_point_vec_polar:nn` *8*
`\draw_point_vec_polar:nmn` *8*
`\draw_scope_begin:` *6*
`\draw_scope_end:` *6*
`\draw_set_baseline:n` *3*
`\draw_set_cap_but:` *5*
`\draw_set_cap_rectangle:` *5*
`\draw_set_cap_round:` *5*
`\draw_set_dash_pattern:nm` *4*
`\draw_set_evenodd_rule:` *5*
`\draw_set_join_bevel:` *5*
`\draw_set_join_miter:` *5*
`\draw_set_join_round:` *5*
`\draw_set_linewidth:n` *4*
`\draw_set_miterlimit:n` *5*
`\draw_set_nonzero_rule:` *5*
`\draw_suspend_begin:` *3*
`\draw_suspend_end:` *3*
`\draw_transform_matrix:nmnn` *19*
`\draw_transform_matrix_absolute:nmnn`
..... *19*
`\draw_transform_matrix_invert:` .. *20*
`\draw_transform_matrix_reset:` ... *20*
`\draw_transform_rotate:n` *19*
`\draw_transform_scale:n` *20*
`\draw_transform_shift:n` *19*
`\draw_transform_shift_absolute:n` *19*
`\draw_transform_shift_invert:` ... *20*
`\draw_transform_shift_reset:` ... *20*
`\draw_transform_triangle:nmn` ... *19*
`\draw_transform_xscale:n` *20*
`\draw_transform_xshift:n` *20*
`\draw_transform_xslant:n` *20*
`\draw_transform_yscale:n` *20*

\draw_transform_yshift:n	20				
\draw_transform_yslant:n	20				
\draw_xvec:n	7	group commands:			
\draw_yvec:n	7	\group_begin:			5
\draw_zvec:n	7	\group_end:			5

G