Package: dee (via r-universe)

May 17, 2026

Type Package

Title Tools to Construct SVG Path 'd' Attributes

Version 0.1.0-29

URL https://github.com/trevorld/dee,
https://trevorldavis.com/R/dee/dev/

BugReports https://github.com/trevorld/dee/issues

Description Provides helper functions to construct the SVG path 'd' <https://developer.mozilla.org/en-US/docs/Web/SVG/Reference/Attribute/d> attribute string. Supports all standard SVG path commands including 'moveto', 'lineto', 'closepath', quadratic and cubic Bézier curves, and elliptical arc curves. Also provides convenience wrappers for common shapes such as arcs, circles, ellipses, polygons, slashes, and stars. Coordinates may be supplied as numeric vectors or as 'affiner' coordinate objects and an option is available to automatically round them to a target number of decimal places for more compact svg strings. An option is also available to use a bottom-left origin (as is conventional in R graphics) rather than the top-left origin used by SVG.

Depends R (>= 4.2)

Imports affiner (>= 0.3.1), grid, rlang, utils

Suggests grDevices, nanosvgr, omsvg, polyclip, testthat

Additional_repositories https://trevorld.r-universe.dev

License MIT + file LICENSE

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Encoding UTF-8

Config/roxygen2/version 8.0.0

Repository https://trevorld.r-universe.dev

Date/Publication 2026-05-15 21:28:56 UTC

RemoteUrl https://github.com/trevorld/dee

RemoteRef HEAD

RemoteSha 7402cddf0cbf3a1dee998737ca1237115dfb2686

Contents

`A` *The elliptical arc curve commands*

Description

`A()` and `aa()` draw elliptical arc curve commands.

Usage

```
A(
  rx,
  ry = rx,
  x_axis_rotation = -a,
  large_arc_flag = big,
  sweep_flag = cw,
  x,
  y = NULL,
```

```
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf),
  a = 0,
  cw = FALSE,
  big = FALSE
)

aa(
  rx,
  ry = rx,
  x_axis_rotation = 0,
  large_arc_flag = FALSE,
  sweep_flag = FALSE,
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

AZ(...)

az(...)
```

Arguments

rx, ry	Radius of ellipse.
x_axis_rotation, a	Angle (in degrees) to rotate ellipse clockwise. Will be coerced by affiner::degrees(). If isTRUE(origin_at_bottom) then will be multiplied by -1 in the output. a is an alternative that instead rotates the ellipse counter-clockwise.
large_arc_flag, big	If TRUE then the arc will be one of two larger arc sweeps else one of the two smaller arc sweeps. big is an alias.
sweep_flag, cw	If TRUE then the arc will be one of the two possible arcs in the "clockwise" direction. else one of the two in a "counter-clockwise" direction. If isTRUE(origin_at_bottom) then will be inverted in the output. cw is an alias.
x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
...	Passed to related function.
sep	Either "," or " ".

`origin_at_bottom`, `height`
: If `origin_at_bottom` is `TRUE` then `y` (and any `y1` and `y2`) coordinates is transformed by `height - y` for absolute coordinates and `-y` for relative coordinates.

`digits`
: `x` and `y` (and any `x1`, `y1`, `x2`, `y2`) will be transformed by `round(, digits)`. An `Inf` (default) means no rounding.

Value

A `dee()` object.

Examples

```
M(1, 1) + A(rx = 1, x = 2, y = 2) + Z()
M(1, 1) + aa(rx = 1, x = 1, y = 1) + zz()
d_small_ccw <- M(40, 70) + A(rx = 20, ry = 30, x = 20, y = 20)
d_small_cw <- M(40, 70) + A(rx = 20, ry = 30, x = 20, y = 20, cw = TRUE)
d_big_ccw <- M(40, 70) + A(rx = 20, ry = 30, x = 20, y = 20, big = TRUE)
d_big_cw <- M(40, 70) + A(rx = 20, ry = 30, x = 20, y = 20, big = TRUE, cw = TRUE)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(c(d_small_ccw, d_small_cw, d_big_ccw, d_big_cw),
    height = 100, width = 100,
    stroke = c("black", "grey", "blue", "cyan"),
    fill = "none", stroke_width = 4)
}
```

---

as_omsvg *Convert to an* `omsvg` *"svg" class object*

Description

`as_omsvg()` converts a dee object to an `omsvg` "svg" class object using `omsvg::SVG()` and `omsvg::svg_path()`.

Usage

```
as_omsvg(
  x,
  ...,
  height = getOption("dee.height"),
  width = getOption("dee.width"),
  background_color = getOption("dee.background_color", "none"),
  stroke = getOption("dee.stroke"),
  stroke_width = getOption("dee.stroke_width"),
  fill = getOption("dee.fill"),
  attrs = getOption("dee.attrs")
)
```

Arguments

x	An object of class "dee".
...	Passed to omsvg::svg_path().
height, width	svg width and height.
background_color	Background color.
stroke, stroke_width, fill, attrs	Passed to omsvg::svg_path().

Examples

```
s <- affiner::star_inner_radius(n = 5, d = 2)
d <- d_star(x = 50, y = 50, r = 30, s = s, n = 5, digits = 0)
if (requireNamespace("omsvg", quietly = TRUE)) {
  svg <- as_omsvg(d, height = 100, width = 100,
                  fill = "fill", stroke = "none")
  paste(svg, collapse = "\n") |> cat("\n")
}
```

C *The cubic Bézier curve commands*

Description

C() and cc() draw cubic Bézier curves S() and ss() draw cubic Bézier curves assuming the first control point is the reflection of the previous Bézier curve command.

Usage

```
C(
  x1,
  y1 = NULL,
  x2,
  y2 = NULL,
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

cc(
  x1,
  y1 = NULL,
```

```
  x2,
  y2 = NULL,
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

CZ(...)

cz(...)

S(
  x2,
  y2 = NULL,
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

ss(
  x2,
  y2 = NULL,
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

SZ(...)

sz(...)
```

Arguments

x1	If y1 is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y1	Either NULL or a numeric vector.

x2	If y2 is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y2	Either NULL or a numeric vector.
x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
...	Passed to related function.
sep	Either "," or " ".
origin_at_bottom, height	If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.
digits	x and y (and any x1, y1, x2, y2) will be transformed by round(, digits). An Inf (default) means no rounding.

Value

A dee() object.

Examples

```
M(1, 1) + C(2, 2, 3, 3, 4, 4) + Z()
M(1, 1) + cc(1, 1, 2, 2, 3, 3) + zz()
```

---

d_aabb *Axis-aligned bounding box convenience wrapper*

Description

d_aabb() is a wrapper around d_rect() to create an axis-aligned bounding box path.

Usage

```
d_aabb(x, y = NULL, ...)
```

Arguments

x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
...	Passed to d_polygon().

Value

A dee() object.

See Also

d_rect(), d_polygon()

Examples

```
d_aabb(x = 2:6, y = 4:8)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_aabb(x = 2:6, y = 4:8),
       height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

---

d_arc1 *Elliptical arc path convenience wrapper*

Description

d_arc1() d_arc2(), d_arc3(), d_arc4(), d_arc12(), d_arc23(), d_arc34(), d_arc41(), d_arc123(), d_arc234(), d_arc341(), and d_arc412() are wrappers to create elliptical arc paths. They are vectorized in their y_top, x_right, y_bottom, x_left, and w arguments. The numbers after d_arc refer to the quadrants of a unit circle (in counterclockwise order). and give an indication of what the shape of the arc looks like.

Usage

```
d_arc1(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc2(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc3(
  y_top,
  x_right,
```

```
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc4(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc12(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc23(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc34(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
```

```
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc41(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc123(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc234(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)

d_arc341(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)
```

```
d_arc412(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL)
)
```

Arguments

y_top, x_right, y_bottom, x_left	The most extreme x and y values of the quarter circular arc shape.
w	The (stroke) width of the arc "line".
...	Passed to M(), A(), and L().
origin_at_bottom, height	If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.

Value

A dee() object.

See Also

d_fslash() and d_bslash()

Examples

```
d_1 <- d_arc1(2, 8, 8, 2, 1)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_1, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
d_2 <- d_arc2(2, 8, 8, 2, 1)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_2, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
d_34 <- d_arc34(2, 8, 8, 2, 1)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_34, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
d_412 <- d_arc412(2, 8, 8, 2, 1)
```

```
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_412, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

`d_ellipse` *Ellipse path convenience wrapper*

Description

`d_ellipse()` is a wrapper around `M()` and `AZ()` to create ellipse shaped paths. It's vectorized in its `x`, `y`, `rx`, and `ry` arguments. `d_circle()` is a special case to create circle shaped paths. It's vectorized in its `x`, `y`, and `r` arguments.

Usage

```
d_ellipse(
  x,
  y = NULL,
  rx,
  ry = rx,
  a = 0,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height")
)

d_circle(x, y = NULL, r, ...)
```

Arguments

`x`	If `y` is `NULL` will be coerced by `affiner::as_coord2d()`. Else a numeric vector.
`y`	Either `NULL` or a numeric vector.
`rx`, `ry`	The radii of the ellipse.
`a`	The angle of rotation (will be coerced by `affiner::degrees()`).
`...`	Passed to `M()` and `AZ()`.
`origin_at_bottom`, `height`	If `origin_at_bottom` is `TRUE` then `y` (and any `y1` and `y2`) coordinates is transformed by `height - y` for absolute coordinates and `-y` for relative coordinates.
`r`	The radius of the circle.

Value

A `dee()` object.

Examples

```
d_circle(x = 5, y = 5, r = 2)
d_ellipse(x = 5, y = 5, rx = 2, ry = 3)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_ellipse(x = 5, y = 5, rx = 2, ry = 3),
       height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

`d_fslash` *Forward/backward slash path convenience wrapper*

Description

`d_fslash()` is a wrapper around `MZ()` to create a forward slash path convenience wrapper. `d_bslash()` is a wrapper around `MZ()` to create a backward slash path convenience wrapper. They are vectorized in their `y_top`, `x_right`, `y_bottom`, `x_left`, and `w` arguments.

Usage

```
d_fslash(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  nib = c("horizontal", "diagonal", "square", "vertical"),
  left = nib,
  right = nib
)

d_bslash(
  y_top,
  x_right,
  y_bottom,
  x_left,
  w,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  nib = c("horizontal", "diagonal", "square", "vertical"),
  left = nib,
  right = nib
)
```

Arguments

`y_top`, `x_right`, `y_bottom`, `x_left`
: The most extreme x and y values of the slash shape.

`w`
: The (stroke) width of the slash "line".

`...`
: Passed to `MZ()`.

`origin_at_bottom`, `height`
: If `origin_at_bottom` is `TRUE` then `y` (and any `y1` and `y2`) coordinates is transformed by `height - y` for absolute coordinates and `-y` for relative coordinates.

`nib`, `left`, `right`
: The shape of the "nib" tracing the line. This only affects the left and right ends.

Value

A `dee()` object.

See Also

`width_slash_left()`, `width_slash_right()`, `height_slash_left()`, `height_slash_right()`

`d_arc1()`, `d_arc2()`, `d_arc3()`, and `d_arc4()` for curved (quarter-elliptical arc) lines.

Examples

```
d_h <- d_fslash(2, 8, 8, 2, 1)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_h, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
d_s <- d_bslash(2, 8, 8, 2, 1, nib = "square")
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_s, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
d_v <- d_fslash(2, 8, 8, 2, 1, nib = "vertical")
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_v, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
d_d <- d_bslash(2, 8, 8, 2, 1, nib = "diagonal")
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_d, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

d_isotoxal_2ngon *Isotoxal* 2n*-gon path convenience wrapper*

Description

`d_isotoxal_2ngon()` is a wrapper around `d_polygon()` to create isotoxal 2n-gon polygon shaped paths. Its vectorized in its `x`, `y`, `r`, `n`, `a`, `s`, and `offset` arguments. `d_star()` is provided as an alias.

Usage

```
d_isotoxal_2ngon(
  x,
  y,
  r,
  n,
  a = 90,
  ...,
  s = affiner::isotoxal_2ngon_inner_radius(n, alpha = alpha, beta_ext = beta_ext, d = d),
  alpha = NULL,
  beta_ext = NULL,
  d = NULL,
  offset = 0,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height")
)

d_star(
  x,
  y,
  r,
  n,
  a = 90,
  ...,
  s = affiner::isotoxal_2ngon_inner_radius(n, alpha = alpha, beta_ext = beta_ext, d = d),
  alpha = NULL,
  beta_ext = NULL,
  d = NULL,
  offset = 0,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height")
)
```

Arguments

x	If y is `NULL` will be coerced by `affiner::as_coord2d()`. Else a numeric vector.
y	Either `NULL` or a numeric vector.

r	The outer radius of the isotoxal 2n-gon polygon.
n	The number of outer vertices.
a	The angle of the first outer vertex (will be coerced by affiner::degrees()).
...	Passed to d_polygon().
s	The inner radius of the isotoxal 2n-gon polygon as a **fraction** of the outer radius i.e. the inner radius = s * r. Defaults to affiner::isotoxal_2ngon_inner_radius() computed from exactly one of alpha, beta_ext, or d.
alpha	Interior angle of an outer vertex passed to affiner::isotoxal_2ngon_inner_radius() to compute s. Will be coerced by affiner::degrees().
beta_ext	Exterior angle of an inner vertex passed to affiner::isotoxal_2ngon_inner_radius() to compute s. Will be coerced by affiner::degrees().
d	Density (winding number) of the star polygon |n/d| passed to affiner::isotoxal_2ngon_inner_radi to compute s.
offset	If a positive number the distance for *outward* polygon offsetting. If a negative number the distance for *inward* polygon offsetting.
origin_at_bottom, height	If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.

Value

A dee() object.

See Also

d_polygon() and d_regular_ngon(). See https://en.wikipedia.org/wiki/Isotoxal_figure#Isotoxal_polygons and https://en.wikipedia.org/wiki/Star_polygon#Isotoxal_star_simple_polygons for more information on isotoxal polygons.

Examples

```
# A |5/2| star e.g. the *verda stelo* (using `d` shortcut)
d <- d_isotoxal_2ngon(x = 5, y = 5, r = 3, n = 5, d = 2)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 10, width = 10,
       fill = "green", stroke = "none")
}
# "lozenge" shape most often has acute angles of 45 degrees (using `alpha`)
d <- d_isotoxal_2ngon(x = 5, y = 5, r = 4, n = 2, alpha = 45)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 10, width = 10,
       fill = "cyan", stroke = "black", stroke_width = 4)
}
# `d_star()` is an alias for `d_isotoxal_2ngon()`
# Inner exterior angle of |8/3| star is 90 degrees (using `beta_ext`)
d <- d_star(x = 5, y = 5, r = 4, n = 8, a = 22.5, beta_ext = 90)
```

```
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 10, width = 10,
       fill = "yellow", stroke = "black", stroke_width = 4)
}

# Degenerate case of inner vertex with exterior angle of 180 degrees
# creates a regular `n`-gon.
d <- d_star(x = 5, y = 5, r = 4, n = 8, a = 22.5, beta_ext = 180)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

---

d_polygon *Polygon path convenience wrapper*

Description

d_polygon() is a wrapper around MZ() to create polygon shaped paths. If the argument offset is nonzero will use polyclip::polyoffset() to compute an offset region. It's vectorized in its offset argument.

Usage

```
d_polygon(
  x,
  y = NULL,
  ...,
  offset = 0,
  linejoin = c("miter", "round"),
  miterlimit = 4
)
```

Arguments

x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
...	Passed to MZ().
offset	If a positive number the distance for *outward* polygon offsetting. If a negative number the distance for *inward* polygon offsetting.
linejoin	If offset is nonzero the type of join operation to use at each vertex when computing the offset region.
miterlimit	Tolerance parameter if offset is nonzero and linejoin = "miter". See polyclip::polyoffset().

Value

A dee() object.

See Also

polyclip::polyoffset() for details on computing the offset region when offset is non-zero.

Examples

```
l <- list(x = c(2, 5, 8, 5), y = c(5, 8, 5, 2))
d <- d_polygon(l, offset = c(1, 0, -1))
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 10, width = 10,
       attrs = list(fill_rule = "evenodd"),
       fill = "red", stroke = "black", stroke_width = 4)
}
```

---

d_rect *Rectangle path convenience wrapper*

Description

d_rect() is a wrapper around d_polygon() to create rectangle shaped paths. It's vectorized in its x, y, w, and h arguments.

Usage

```
d_rect(
  x,
  y = NULL,
  w,
  h,
  a = 0,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height")
)
```

Arguments

x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
w, h	The width and height of the rectangle.
a	The angle of rotation (will be coerced by affiner::degrees()).
...	Passed to d_polygon().

origin_at_bottom, height
: If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.

Value

A dee() object.

See Also

d_aabb(), d_polygon()

Examples

```
d_rect(x = 5, y = 5, w = 4, h = 6)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d_rect(x = 5, y = 5, w = 4, h = 6),
       height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

---

d_regular_ngon *Regular* n*-gon path convenience wrapper*

Description

d_regular_ngon() is a wrapper around d_polygon() to create regular polygon shaped paths. Its vectorized in its x, y, r, n, a, and offset arguments.

Usage

```
d_regular_ngon(
  x,
  y,
  r,
  n,
  a = 90,
  ...,
  offset = 0,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height")
)
```

Arguments

x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
r	The radius of the regular polygon (where the vertices lay on).
n	The number of vertices.
a	The angle of the first vertex (will be coerced by affiner::degrees()).
...	Passed to d_polygon().
offset	If a positive number the distance for *outward* polygon offsetting. If a negative number the distance for *inward* polygon offsetting.
origin_at_bottom, height	If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.

Value

A dee() object.

See Also

d_polygon() and d_isotoxal_2ngon().

Examples

```
# A pentagon
d5 <- d_regular_ngon(x = 5, y = 5, r = 4, n = 5)
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d5, height = 10, width = 10,
       fill = "magenta", stroke = "black", stroke_width = 4)
}
# A hexagon
d6 <- d_regular_ngon(x = 5, y = 5, r = 4, n = 6, offset = c(0, -1))
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d6, height = 10, width = 10,
       attrs = list(fill_rule = "evenodd"),
       fill = "magenta", stroke = "black", stroke_width = 4)
}
```

---

dee *Build an object of class "dee"*

Description

dee() casts a string to an object of class "dee".

Usage

```
dee(x)
```

Arguments

x	A character vector.

Value

An object of class "dee".

Examples

```
d <- dee("M 10,30") + dee("V 30")
c(d, d)
```

---

dee_options *Get dee options*

Description

`dee_options()` returns the dee package's global options.

Usage

```
dee_options(..., default = FALSE)
```

Arguments

...	dee package options using `name = value`. The return list will use any of these instead of the current/default values.
default	If `TRUE` return the default values instead of current values.

Value

A list of option values. Note this function **does not** set option values itself but this list can be passed to `options()`, `withr::local_options()`, or `withr::with_options()`.

See Also

dee-package for a high-level description of relevant global options.

Examples

```
dee_options()

dee_options(default = TRUE)

dee_options(dee.height = 100, dee.width = 100, dee.origin_at_bottom = TRUE)
```

height_slash_left *Height/width of slash ends*

Description

height_slash_left(), height_slash_right(), width_slash_left(), and width_slash_right() return the height or width of the left or right end of the equivalent d_fslash() or d_bslash() shapes.

Usage

```
height_slash_left(
  dx,
  dy,
  w,
  ...,
  nib = c("horizontal", "diagonal", "square", "vertical"),
  left = nib,
  right = nib
)

height_slash_right(
  dx,
  dy,
  w,
  ...,
  nib = c("horizontal", "diagonal", "square", "vertical"),
  left = nib,
  right = nib
)

width_slash_left(
  dx,
  dy,
  w,
  ...,
  nib = c("horizontal", "diagonal", "square", "vertical"),
  left = nib,
  right = nib
)

width_slash_right(
  dx,
  dy,
  w,
  ...,
  nib = c("horizontal", "diagonal", "square", "vertical"),
```

```
  left = nib,
  right = nib
)
```

Arguments

dx	The difference x_right - x_left in d_fslash() and d_bslash().
dy	The difference y_bottom - y_top in d_fslash() and d_bslash().
w	The (stroke) width of the slash "line".
...	Must be empty.
nib, left, right	The shape of the "nib" tracing the line. This only affects the left and right ends.

Value

A numeric value.

See Also

d_fslash() and d_bslash()

Examples

```
width_slash_left(6, 6, 1, nib = "horizontal")
width_slash_left(6, 6, 1, left = "horizontal", right = "vertical")
height_slash_left(6, 6, 1, nib = "diagonal")
height_slash_right(6, 6, 1, nib = "diagonal")
width_slash_right(6, 6, 1, nib = "vertical")
```

L *The "lineto" commands*

Description

L() and ll() draw straight lines, H() and hh() draw horizontal lines, and V() and vv() draw vertical lines.

Usage

```
L(
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)
```

```
ll(
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

LZ(...)

lz(...)

H(x, ..., digits = getOption("dee.digits", Inf))

hh(x, ..., digits = getOption("dee.digits", Inf))

HZ(...)

hz(...)

V(
  y,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

vv(
  y,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

VZ(...)

vz(...)
```

Arguments

x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
...	Passed to related function.

sep — Either "," or " ".

origin_at_bottom, height — If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.

digits — x and y (and any x1, y1, x2, y2) will be transformed by round(, digits). An Inf (default) means no rounding.

Value

A dee() object.

Examples

```
M(1, 1) + L(2, 2) + Z()
M(1, 1) + ll(1, 1) + zz()
d <- M(1,1) + L(5,8) + H(8) + V(3) + Z()
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

M *The "moveto" commands*

Description

M() and mm() move the "pen" to a new point.

Usage

```
M(
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

mm(
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
```

```
  digits = getOption("dee.digits", Inf)
)

MZ(...)

mz(...)
```

Arguments

x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
...	Passed to related function.
sep	Either ",", or " ".
origin_at_bottom, height	If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.
digits	x and y (and any x1, y1, x2, y2) will be transformed by round(, digits). An Inf (default) means no rounding.

Value

A dee() object.

Examples

```
M(1, 1) + L(2, 2) + Z()
M(1, 1) + ll(1, 1) + zz()
d <- MZ(x = c(2, 5, 8, 5), y = c(5, 8, 5, 2))
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 10, width = 10,
       fill = "red", stroke = "black", stroke_width = 4)
}
```

---

plot.dee *Plot an object of class "dee"*

Description

plot() an object of class "dee" using omsvg::SVG(), omsvg::svg_path(), and nanosvgr::nsvg_read().

Usage

```
## S3 method for class 'dee'
plot(
  x,
  ...,
  height = getOption("dee.height"),
  width = getOption("dee.width"),
  background_color = getOption("dee.background_color", "none"),
  stroke = getOption("dee.stroke"),
  stroke_width = getOption("dee.stroke_width"),
  fill = getOption("dee.fill"),
  attrs = getOption("dee.attrs")
)
```

Arguments

x	An object of class "dee".
...	Passed to omsvg::svg_path().
height, width	svg width and height.
background_color	Background color.
stroke, stroke_width, fill, attrs	Passed to omsvg::svg_path().

Details

This function requires the package nanosvgr which (as of May 2026) is not available on CRAN.
You can install it with remotes::install_github('coolbutuseless/nanosvgr') or utils::install.packages('nanos
repos = c('https://trevorld.r-universe.dev', 'https://cloud.r-project.org')).

Value

invisible(NULL)

Examples

```
d <- M(10, 30) +
     A(20, 20, 0, 0, 1, 50, 30) +
     A(20, 20, 0, 0, 1, 90, 30) +
     Q(90, 60, 50, 90) +
     Q(10, 60, 10, 30) +
     Z()
if (requireNamespace("omsvg", quietly = TRUE) &&
    requireNamespace("nanosvgr", quietly = TRUE)) {
  plot(d, height = 100, width = 100, background_color = "cyan",
       fill = "red", stroke = "black", stroke_width = 4)
}
```

Q *The quadratic Bézier curve commands*

Description

`Q()` and `qq()` draw quadratic Bézier curves `T()` and `tt()` draw quadratic Bézier curves assuming the control point is the reflection of the previous Bézier curve command.

Usage

```
Q(
  x1,
  y1 = NULL,
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

qq(
  x1,
  y1 = NULL,
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

QZ(...)

qz(...)

T(
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)
```

```
tt(
  x,
  y = NULL,
  ...,
  sep = getOption("dee.sep", ","),
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE),
  height = getOption("dee.height", NULL),
  digits = getOption("dee.digits", Inf)
)

TZ(...)

tz(...)
```

Arguments

x1	If y1 is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y1	Either NULL or a numeric vector.
x	If y is NULL will be coerced by affiner::as_coord2d(). Else a numeric vector.
y	Either NULL or a numeric vector.
...	Passed to related function.
sep	Either "," or " ".
origin_at_bottom, height	If origin_at_bottom is TRUE then y (and any y1 and y2) coordinates is transformed by height - y for absolute coordinates and -y for relative coordinates.
digits	x and y (and any x1, y1, x2, y2) will be transformed by round(, digits). An Inf (default) means no rounding.

Value

A dee() object.

Examples

```
M(1, 1) + Q(2, 2, 3, 3) + T(4, 4) + Z()
M(1, 1) + qq(1, 1, 2, 2) + tt(1, 1) + zz()
```

---

x_ellipse_left *Find ellipse coordinate*

Description

`x_ellipse_left()` returns the x-coordinate of the leftmost point and `x_ellipse_right()` returns the x-coordinate of the rightmost point on a (possibly rotated) ellipse at a given y value. `y_ellipse_top()` returns the y-coordinate of the topmost point and `y_ellipse_bottom()` returns the y-coordinate of the bottommost point on a (possibly rotated) ellipse at a given x value. `x_ellipse_left()` and `x_ellipse_right()` return `NaN` if `y` is outside the vertical extent of the ellipse. `y_ellipse_top()` and `y_ellipse_bottom()` return `NaN` if `x` is outside the horizontal extent of the ellipse.

Usage

```
x_ellipse_left(
  y,
  xc,
  yc,
  rx,
  ry = rx,
  a = 0,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE)
)

x_ellipse_right(
  y,
  xc,
  yc,
  rx,
  ry = rx,
  a = 0,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE)
)

y_ellipse_top(
  x,
  xc,
  yc,
  rx,
  ry = rx,
  a = 0,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE)
)

y_ellipse_bottom(
  x,
  xc,
  yc,
```

```
  rx,
  ry = rx,
  a = 0,
  ...,
  origin_at_bottom = getOption("dee.origin_at_bottom", FALSE)
)
```

Arguments

y	The y value at which to evaluate the ellipse.
xc, yc	The center of the ellipse.
rx, ry	The x and y radii of the (unrotated) ellipse.
a	The counter-clockwise angle of rotation of the ellipse (will be coerced by `affiner::degrees()`).
...	Should be empty.
origin_at_bottom	If TRUE, the origin is at the bottom of the coordinate system (i.e. y increases upward).
x	The x value at which to evaluate the ellipse.

Value

A numeric value.

See Also

`A()` and `d_ellipse()`.

Examples

```
# Unrotated ellipse centered at (5, 5) with rx=3, ry=4
x_ellipse_left(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 0)
x_ellipse_right(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 0)
# Rotated 90 degrees: rx and ry swap roles
x_ellipse_left(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 90)
x_ellipse_right(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 90)
# Unrotated ellipse centered at (5, 5) with rx=3, ry=4
y_ellipse_top(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 0)
y_ellipse_bottom(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 0)
# Rotated 90 degrees: rx and ry swap roles
y_ellipse_top(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 90)
y_ellipse_bottom(5, xc = 5, yc = 5, rx = 3, ry = 4, a = 90)
```

Z *The "closepath" commands*

Description

`Z()` and `zz()` connects the initial point of the current subpath with the last point (with a straight line if these are different). There is no difference between `Z()` and `zz()`.

Usage

```
Z()

zz()
```

Value

A `dee()` object.

Examples

```
M(1, 1) + L(2, 2) + Z()
M(1, 1) + ll(2, 2) + zz()
```

Index