Package 'omsvg'

October 14, 2022

Type Package

Version 0.1.0

Title Build and Transform 'SVG' Objects

Description Build 'SVG' components using element-based functions. With an 'svg' object, we can modify its graphical elements with a suite of transform functions.

License MIT + file LICENSE

URL https://github.com/rich-iannone/omsvg

BugReports https://github.com/rich-iannone/omsvg/issues

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 7.1.1

Depends R (>= 3.2.1)

Imports dplyr (>= 1.0.3), gt (>= 0.2.2), htmltools (>= 0.5.1.1), magrittr, rlang (>= 0.4.5), sass (>= 0.3.0), xml2 (>= 1.3.2)

Suggests covr, knitr, testthat

NeedsCompilation no

Author Richard Iannone [aut, cre] (<https://orcid.org/0000-0003-3925-190X>)

Maintainer Richard Iannone <riannone@me.com>

Repository CRAN

Date/Publication 2021-02-10 10:50:06 UTC

R topics documented:

anims 2
anim_opacity 5
anim_position 6
anim_rotation 7

anim_scale 8
cubic_bezier 9
ease_in 10
ease_in_out 10
ease_out 11
filter_dilate 11
filter_drop_shadow 12
filter_erode 13
filter_gaussian_blur 14
filter_image 15
filter_offset 16
info_lineawesome 17
linear 18
step_end 18
step_start 18
SVG 19
SVG_ 20
svg_attrs_pres 21
svg_circle 25
svg_ellipse 27
svg_filter 28
svg_group 30
svg_image 31
SVG_import 33
SVG_la 34
svg_line 35
svg_path 37
svg_polygon 38
svg_polyline 40
svg_rect 41
SVG_t 44
svg_text 45

Index **47**

`anims` *Express animations for an element*

Description

All SVG element functions in **omsvg** (the `svg_*()` functions) are animatable through their `anims` argument. The `anims()` function should be used with that argument should we want to express animations for the element. Within the `anims()` function call, we can insert a list of formulas that incorporate calls to any of the `anim_*()` functions (e.g., `anim_position()`, `anim_rotation()`, etc.), and, have keyframe times as part of the formula.

Usage

```
anims(...)
```

Arguments

`...`	One or more animations that included the use of `anim_*()` functions, expressed as two-sided formulas. The LHS provides the keyframe time (in units of seconds) and the RHS is the associated `anim_*()` call.

Details

A useful template to use for an `anims()` call within an `svg_*()` function is:

```
anims = anims(
  <time_i> ~ <anim_fn>(...),
  ...,
  <time_n> ~ <anim_fn>(...)
  )
```

We can also use multiple calls to `anim_*()` functions for each distinct keyframe time by placing those calls in a list:

```
anims = anims(
  <time_i> ~ list(
    <anim_fn_x>(...),
    <anim_fn_y>(...)
    ),
  ...,
  <time_n> ~ list(
    <anim_fn_x>(...),
    <anim_fn_y>(...)
    )
  )
```

Value

A tibble of animation directives.

Examples

```
if (interactive()) {

# Basic animation of an element's
# position (moving to a new `x` and
# `y` position)
svg_1 <-
  SVG(width = 300, height = 300) %>%
  svg_rect(
    x = 50, y = 50,
    width = 50, height = 50,
    attrs = svg_attrs_pres(
      stroke = "magenta",
      fill = "lightblue"
    ),
    anims = anims(
      2.0 ~ anim_position(x = 100, y = 50)
    )
  )

# We can define multiple animations
# for a single element: put them in a
# `list()`; the `easing_fn` function for
# both `anim_*()` function is no longer
# linear but now eases in and out
svg_2 <-
  SVG(width = 300, height = 300) %>%
  svg_rect(
    x = 50, y = 50,
    width = 50, height = 50,
    attrs = svg_attrs_pres(
      stroke = "black",
      fill = "yellow"
    ),
    anims = anims(
      0.5 ~ list(
        anim_position(x = 50, y = 50, easing_fn = ease_in_out()),
        anim_rotation(0, easing_fn = ease_in_out())
      ),
      2.0 ~ list(
        anim_position(x = 200, y = 50, easing_fn = ease_in_out()),
        anim_rotation(90, easing_fn = ease_in_out())
      )
    )
  )

# The initial state of the element
# can be used in any `anim_*()`
# function with `initial = TRUE`
svg_3 <-
  SVG(width = 300, height = 300) %>%
  svg_rect(
    x = 50, y = 50,
    width = 50, height = 50,
    attrs = svg_attrs_pres(
      stroke = "black",
      fill = "yellow"
    ),
    anims = anims(
      1.0 ~ list(
        anim_position(initial = TRUE),
        anim_rotation(initial = TRUE)
      ),
      3.0 ~ list(
        anim_position(x = 200, y = 50),
        anim_rotation(90)
      ),
      5.0 ~ list(
        anim_position(initial = TRUE),
        anim_rotation(initial = TRUE)
      )
    )
  )
}
```

anim_opacity *Animate an element through an opacity change*

Description

Within an anims() call, itself passed to any anims argument, the anim_opacity() function can be used to express an animation where the target element undergoes a change in opacity with time.

Usage

```
anim_opacity(opacity = NULL, easing_fn = NULL, initial = FALSE)
```

Arguments

opacity	The opacity value of the element at the keyframe time (given as the LHS value in the anims() call).
easing_fn	The timing or easing function to use for the animation. If not provided, the linear() timing function will be used (which is doesn't use any easing in the animation, just a linear movement). The other timing and easing functions are: step_start(), step_end(), ease_in(), ease_out(), and ease_in_out().
initial	Should this opacity value be the initial opacity value of the element? If so, use TRUE and any value provided to opacity will be disregarded.

Value

An anim_opacity object, which is to be used as part of an anims() call.

Examples

```
if (interactive()) {

# Basic animation of an element's
# opacity value (moving to a new
# `opacity` value of `0`)
SVG(width = 300, height = 300) %>%
  svg_rect(
    x = 50, y = 50,
    width = 50, height = 50,
    attrs = svg_attrs_pres(
      stroke = "magenta",
      fill = "lightblue"
    ),
    anims = anims(
      2.0 ~ anim_opacity(opacity = 0)
    )
  )
}
```

---

anim_position — *Animate the position of an element*

Description

Within an `anims()` call, itself passed to any `anims` argument, the `anim_position()` function can be used to express an animation where the position of the target element changes with time.

Usage

```
anim_position(x = NULL, y = NULL, easing_fn = NULL, initial = FALSE)
```

Arguments

`x, y`	The position of the element, expressed as `x` and `y`, at the keyframe time (given as the LHS value in the `anims()` call).
`easing_fn`	The timing or easing function to use for the animation. If not provided, the `linear()` timing function will be used (which is doesn't use any easing in the animation, just a linear movement). The other timing and easing functions are: `step_start()`, `step_end()`, `ease_in()`, `ease_out()`, and `ease_in_out()`.
`initial`	Should this position be the initial position of the element? If so, use `TRUE` and any values provided to `x` and `y` will be disregarded.

Value

An `anim_opacity` object, which is to be used as part of an `anims()` call.

Examples

```
if (interactive()) {

# Basic animation of an element's
# position (moving to a new `x` and
# `y` position)
SVG(width = 300, height = 300) %>%
  svg_rect(
    x = 50, y = 50,
    width = 50, height = 50,
    attrs = svg_attrs_pres(
      stroke = "magenta",
      fill = "lightblue"
    ),
    anims = anims(
      2.0 ~ anim_position(x = 100, y = 50)
    )
  )
}
```

`anim_rotation` *Animate an element through rotation*

Description

Within an `anims()` call, itself passed to any `anims` argument, the `anim_rotation()` function can be used to express an animation where the target element undergoes a rotation change with time.

Usage

```
anim_rotation(
  rotation = NULL,
  anchor = "center",
  easing_fn = NULL,
  initial = FALSE
)
```

Arguments

`rotation`	The rotation value of the element at the keyframe time (given as the LHS value in the `anims()` call).
`anchor`	The location of the element anchor about which rotation will occur. By default, this is the keyword `"center"`.
`easing_fn`	The timing or easing function to use for the animation. If not provided, the `linear()` timing function will be used (which is doesn't use any easing in the animation, just a linear movement). The other timing and easing functions are: `step_start()`, `step_end()`, `ease_in()`, `ease_out()`, and `ease_in_out()`.
`initial`	Should this rotation value be the initial rotation state of the element? If so, use `TRUE` and any value provided to `rotation` will be disregarded.

Value

An `anim_opacity` object, which is to be used as part of an `anims()` call.

Examples

```
if (interactive()) {

# This is a basic animation of an
# element's rotation state (moving to
# a new `rotation` value)
SVG(width = 300, height = 300) %>%
  svg_rect(
    x = 50, y = 50,
    width = 50, height = 50,
    attrs = svg_attrs_pres(
      stroke = "magenta",
      fill = "lightblue"
    ),
    anims = anims(
      2.0 ~ anim_rotation(rotation = 180)
    )
  )
}
```

---

anim_scale *Animate an element through scaling*

Description

Within an anims() call, itself passed to any anims argument, the anim_scale() function can be used to express an animation where the target element undergoes a scaling change with time.

Usage

```
anim_scale(scale = NULL, easing_fn = NULL)
```

Arguments

scale	The scale value of the element at the keyframe time (given as the LHS value in the anims() call). If providing a single scaling value, the scaling will operate in the x and y directions (relative to the center of the element). If two values are provided, these will be taken as scaling values in the x and y directions.
easing_fn	The timing or easing function to use for the animation. If not provided, the linear() timing function will be used (which is doesn't use any easing in the animation, just a linear movement). The other timing and easing functions are: step_start(), step_end(), ease_in(), ease_out(), and ease_in_out().

Value

An anim_opacity object, which is to be used as part of an anims() call.

Examples

```
if (interactive()) {

# Basic animation of an element's
# scaling state (moving to a new
# `scale` value)
SVG(width = 300, height = 300) %>%
  svg_rect(
    x = 50, y = 50,
    width = 50, height = 50,
    attrs = svg_attrs_pres(
      stroke = "magenta",
      fill = "lightblue"
    ),
    anims = anims(
      2.0 ~ anim_scale(scale = 2)
    )
  )
}
```

`cubic_bezier` *Create a custom easing function for animation*

Description

Create a custom easing function for animation

Usage

```
cubic_bezier(x1 = 0.5, y1 = 0.5, x2 = 0.5, y2 = 0.5)
```

Arguments

`x1, y1, x2, y2`	The `x` and `y` values for the first and second bezier control points.

Value

A `cubic-bezier` function call as a string for use as a CSS property.

ease_in *Use an 'easing in' animation*

Description

The `ease_in()` function can be used as a value for the `easing_fn` argument, which is available in every `anim_*()` function (e.g., `anim_position()`).

Usage

```
ease_in(power = "basic")
```

Arguments

`power`	The preset to use for the easing in cubic bezier function.

Value

A `cubic-bezier` function call as a string for use as a CSS property.

ease_in_out *Use an 'easing in and out' animation*

Description

The `ease_in_out()` function can be used as a value for the `easing_fn` argument, which is available in every `anim_*()` function (e.g., `anim_position()`).

Usage

```
ease_in_out(power = "basic")
```

Arguments

`power`	The preset to use for the easing in cubic bezier function.

Value

A `cubic-bezier` function call as a string for use as a CSS property.

`ease_out` *Use an 'easing out' animation*

Description

The `ease_out()` function can be used as a value for the `easing_fn` argument, which is available in every `anim_*()` function (e.g., `anim_position()`).

Usage

```
ease_out(power = "basic")
```

Arguments

`power`	The preset to use for the easing in cubic bezier function.

Value

A `cubic-bezier` function call as a string for use as a CSS property.

`filter_dilate` *Filter: add a dilation effect to an element*

Description

The `filter_dilate()` filter applies a dilation effect to a source graphic by a given `radius` value. The higher the `radius`, the greater the dilation potential.

Usage

```
filter_dilate(radius = 1)
```

Arguments

`radius`	The extent to which the source graphic will be dilated. If a vector of two values are provided, the first value represents the x-radius and the second one the y-radius. If one value is provided, then that value is used for both x and y.

Value

An `svg` object.

Examples

```
if (interactive()) {

# Add a text element to an
# SVG drawing and erode it with
# the `filter_dilate()` filter
SVG(width = 200, height = 100) %>%
  svg_filter(
    id = "dilate",
    filters = list(
      filter_dilate(radius = c(0, 1))
    )
  ) %>%
  svg_text(
    x = 10, y = 40,
    text = "Dilation",
    attrs = svg_attrs_pres(
      font_size = "3em",
      filter = "dilate"
    )
  )
}
```

`filter_drop_shadow` *Filter: add a drop shadow to an element*

Description

With the `filter_drop_shadow()` drop shadow appears beneath the input image or shape and its offset is controlled by `dx` and `dy`. The blurring of the drop shadow is set by the `stdev` value.

Usage

```
filter_drop_shadow(dx = 0.2, dy = 0.2, stdev = 1, color = "black", opacity = 1)
```

Arguments

`dx, dy`	The offset of the drop shadow compared to the position of the input image or shape.
`stdev`	The number of standard deviations for the blur effect.
`color`	The color of the drop shadow.
`opacity`	The opacity of the drop shadow. We can use a real number from `0` to `1` or a value in percentage units.

Value

An svg object.

Examples

```
if (interactive()) {

# Apply a drop shadow filter on a
# text element (orange in color,
# and semi-opaque)
SVG(width = 250, height = 100) %>%
  svg_filter(
    id = "shadow",
    filters = list(
      filter_drop_shadow(
        dx = 1, dy = 2,
        color = "orange",
        opacity = 0.5
      )
    )
  ) %>%
  svg_text(
    x = 10, y = 40,
    text = "Shadowed",
    attrs = svg_attrs_pres(
      font_size = "2em",
      fill = "#555555",
      font_weight = "bolder",
      filter = "shadow"
    )
  )
}
```

`filter_erode` *Filter: add an erosion effect to an element*

Description

The `filter_erode()` filter effectively thins out a source graphic by a given `radius` value. The higher the `radius`, the greater the extent of thinning.

Usage

```
filter_erode(radius = 1)
```

Arguments

`radius`	The extent to which the source graphic will be eroded. If a vector of two values are provided, the first value represents the x-radius and the second one the y-radius. If one value is provided, then that value is used for both x and y.

Value

An svg object.

Examples

```
if (interactive()) {

# Add a text element to an
# SVG drawing and erode it with
# the `filter_erode()` filter
SVG(width = 200, height = 100) %>%
  svg_filter(
    id = "erode",
    filters = list(
      filter_erode(radius = c(1, 0))
    )
  ) %>%
  svg_text(
    x = 10, y = 40,
    text = "Erosion",
    attrs = svg_attrs_pres(
      font_size = "3em",
      font_weight = "bolder",
      filter = "erode"
    )
  )
}
```

`filter_gaussian_blur` *Filter: add a gaussian blur to an element*

Description

A gaussian blur effectively blurs an input image or shape by the amount specified in `stdev`. The standard deviation of `stdev` is in direct reference to the gaussian distribution that governs the extent of blurring.

Usage

```
filter_gaussian_blur(stdev = 1, what = "source")
```

Arguments

`stdev`	The number of standard deviations for the blur effect.
`what`	What exactly should be blurred? By default, it is the `"source"` image.

Value

An svg object.

Examples

```
if (interactive()) {

# Add a green ellipse to an SVG and
# then apply the `filter_gaussian_blur()`
# filter to blur the edges
SVG(width = 200, height = 100) %>%
  svg_filter(
    id = "blur",
    filters = list(
      filter_gaussian_blur(stdev = 2)
    )
  ) %>%
  svg_ellipse(
    x = 40, y = 40,
    width = 50, height = 30,
    attrs = svg_attrs_pres(
      fill = "green",
      filter = "blur"
    )
  )
}
```

`filter_image` *Filter: display an image*

Description

Display an image using a URL or a relative path to an on-disk resource.

Usage

```
filter_image(image)
```

Arguments

`image`	A link or path to an image resource.

Value

An svg object.

Examples

```
if (interactive()) {

# Place an image (obtained via an image
# link) within a rectangle element using
# the `filter_image()` filter
SVG(width = 500, height = 500) %>%
  svg_filter(
    id = "image",
    filters = list(
      filter_image(
        image = "https://www.r-project.org/logo/Rlogo.png"
      )
    )
  ) %>%
  svg_rect(
    x = 25, y = 25,
    width = "50%", height = "50%",
    attrs = svg_attrs_pres(filter = "image")
  )
}
```

`filter_offset` *Filter: offset an element a specified amount*

Description

The offset filter applies an offset in the x and y directions to an existing element. The offset is handled by setting values for `dx` and `dy`.

Usage

```
filter_offset(dx = NULL, dy = NULL, what = "source")
```

Arguments

`dx, dy`	The offset of the element position compared to its initial position.
`what`	What exactly should be offset? By default, it is the `"source"` image.

Value

An svg object.

Examples

```
if (interactive()) {

# Add a circle element to an
# SVG drawing and offset it
# by 10px to the right
SVG(width = 150, height = 150) %>%
  svg_filter(
    id = "offset_right",
    filters = list(
      filter_offset(dx = 50, dy = 0)
    )
  ) %>%
  svg_circle(
    x = 30, y = 30,
    diameter = 40,
    attrs = svg_attrs_pres(
      fill = "red",
      filter = "offset_right"
    )
  )
}
```

`info_lineawesome` *Get an information table showing all Line Awesome icons*

Description

This informative table shows which Line Awesome icons are available inside of **omsvg**. The icons are composed of lines and they look awesome! There are plenty to choose from also, nearly *1400* icons across *69* categories. Just take note of the ones you like and get their names, you'll need them when using the `SVG_la()` function.

Usage

```
info_lineawesome()
```

Value

Invisibly returns `NULL`. The side effect of displaying a table of icons is the purpose of this function.

`linear` *Use a linear movement for animation*

Description

The `linear()` function can be used as a value for the `easing_fn` argument, which is available in every `anim_*()` function (e.g., `anim_position()`).

Usage

```
linear()
```

Value

A `linear` function call as a string for use as a CSS property.

`step_end` *Use a 'step-end' animation*

Description

The `step_end()` function can be used as a value for the `easing_fn` argument, which is available in every `anim_*()` function (e.g., `anim_position()`).

Usage

```
step_end()
```

Value

A `step-end` function call as a string for use as a CSS property.

`step_start` *Use a 'step-start' animation*

Description

The `step_start()` function can be used as a value for the `easing_fn` argument, which is available in every `anim_*()` function (e.g., `anim_position()`).

Usage

```
step_start()
```

Value

A `step-start` function call as a string for use as a CSS property.

`SVG` *Create an* `svg` *object*

Description

The `SVG()` function is the entry point for building an SVG from the ground up. We can provide predefined `height` and `width` attributes that define the canvas size for the SVG. From here, we would want to use functions that add elements to the SVG object (e.g., `svg_rect()`, `svg_circle()`, etc.) and thus progressively build the graphic.

Usage

```
SVG(
  width = NULL,
  height = NULL,
  viewbox = NULL,
  title = NULL,
  desc = NULL,
  incl_xmlns = FALSE,
  oneline = FALSE,
  anim_iterations = "infinite"
)
```

Arguments

`width, height`	The width and height attributes on the top-level <svg> element. Both of these attributes are optional but, if provided, take in a variety of dimensions and keywords. If numerical values are solely used, they are assumed to be 'px' length values. Dimensions can be percentage values (i.e., `"75%"`) or length values with the following units: `"em"`, `"ex"`, `"px"`, `"in"`, `"cm"`, `"mm"`, `"pt"`, and `"pc"`. Using `NULL`, the default, excludes the attribute.
`viewbox`	An optional set of dimensions that defines the SVG `viewBox` attribute. The `viewBox` for an SVG element is the position and dimension, in user space, of an SVG viewport. If supplied, this could either be in the form of a four-element, numeric vector corresponding to the `"min-x"`, `"min-y"`, `"width"`, and `"height"` of the rectangle, or, as `TRUE` which uses the vector `c(0, 0, width, height)`. Using `NULL`, the default, excludes this attribute.
`title`	The <`title`> tag for the finalized SVG.
`desc`	The <desc> tag for the finalized SVG.
`incl_xmlns`	Should the `xmlns` attribute be included in the <svg> tag? This attribute is only required on the outermost `svg` element of SVG documents, and, it's unnecessary for inner `svg` elements or inside of HTML documents. By default, this is set to `FALSE`.
`oneline`	An option to compress the resulting SVG tags such that they are reduced to one line.

`anim_iterations`
: How many should an SVG animation (if defined by use of the `anims()` function) be played? By default this is `"infinite"` (i.e., looped indefinitely) but we can specify the animation iteration count as a positive number.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with nothing drawn
# within it
svg <- SVG(width = 200, height = 100)

# Add a rectangle and then a circle
svg <-
  svg %>%
  svg_rect(x = 20, y = 20, width = 40, height = 40) %>%
  svg_circle(x = 100, y = 40, diameter = 40)
}
```

SVG_ *Create a compact* svg *object*

Description

The `SVG_()` function is a variation on `SVG()` (the entry point for building an SVG) in that the output tags will be as compact as possible (fewer linebreaks, less space characters). This is a reasonable option if the eventual use for the generated SVG is as inline SVG within HTML documents.

Usage

```
SVG_(width = NULL, height = NULL, viewbox = TRUE)
```

Arguments

`width`
: The width and height attributes on the top-level <svg> element. Both of these attributes are optional but, if provided, take in a variety of dimensions and keywords. If numerical values are solely used, they are assumed to be 'px' length values. Dimensions can be percentage values (i.e., `"75%"`) or length values with the following units: `"em"`, `"ex"`, `"px"`, `"in"`, `"cm"`, `"mm"`, `"pt"`, and `"pc"`. Using `NULL`, the default, excludes the attribute.

`height`	The width and height attributes on the top-level <svg> element. Both of these attributes are optional but, if provided, take in a variety of dimensions and keywords. If numerical values are solely used, they are assumed to be 'px' length values. Dimensions can be percentage values (i.e., `"75%"`) or length values with the following units: `"em"`, `"ex"`, `"px"`, `"in"`, `"cm"`, `"mm"`, `"pt"`, and `"pc"`. Using `NULL`, the default, excludes the attribute.
`viewbox`	An optional set of dimensions that defines the SVG `viewBox` attribute. The `viewBox` for an SVG element is the position and dimension, in user space, of an SVG viewport. If supplied, this could either be in the form of a four-element, numeric vector corresponding to the `"min-x"`, `"min-y"`, `"width"`, and `"height"` of the rectangle, or, as `TRUE` which uses the vector `c(0, 0, width, height)`. Using `NULL`, the default, excludes this attribute.

Value

An svg object.

Examples

```
if (interactive()) {

# Create a simple SVG with a rectangle and a circle
svg <-
  SVG_(width = 100, height = 50) %>%
  svg_rect(x = 0, y = 0, width = 30, height = 20) %>%
  svg_circle(x = 50, y = 10, diameter = 20)
}
```

`svg_attrs_pres` *Define SVG presentation attributes for an element*

Description

The `svg_attrs_pres()` helper function can be used to more easily generate a valid presentation attribute list for the `attrs` argument that is present in every SVG element function (e.g., `svg_rect()`, `svg_text()`, etc.). All of the presentation attributes formally included here as options can be animated.

Usage

```
svg_attrs_pres(
  stroke = NULL,
  stroke_width = NULL,
  stroke_opacity = NULL,
  fill = NULL,
  fill_opacity = NULL,
  font_family = NULL,
  font_size = NULL,
  font_weight = NULL,
  font_style = NULL,
  text_decoration = NULL,
  transform = NULL,
  filter = NULL,
  mask = NULL,
  clip_path = NULL,
  clip_rule = NULL,
  stroke_dasharray = NULL,
  stroke_dashoffset = NULL,
  stroke_linecap = NULL,
  stroke_linejoin = NULL,
  stroke_miterlimit = NULL,
  fill_rule = NULL,
  color = NULL,
  opacity = NULL,
  color_interpolation = NULL,
  color_interpolation_filters = NULL,
  lighting_color = NULL,
  flood_color = NULL,
  flood_opacity = NULL,
  stop_color = NULL,
  stop_opacity = NULL,
  font_variant = NULL,
  font_stretch = NULL,
  font_size_adjust = NULL,
  text_anchor = NULL,
  letter_spacing = NULL,
  word_spacing = NULL,
  dominant_baseline = NULL,
  alignment_baseline = NULL,
  baseline_shift = NULL,
  direction = NULL,
  writing_mode = NULL,
  overflow = NULL,
  marker_start = NULL,
  marker_mid = NULL,
  marker_end = NULL,
  pointer_events = NULL,
  cursor = NULL,
  vector_effect = NULL,
  shape_rendering = NULL,
  color_rendering = NULL,
  text_rendering = NULL,
  image_rendering = NULL,
  display = NULL,
  visibility = NULL
```

```
)
```

Arguments

`stroke`	The color used to paint the outline of the shape.
`stroke_width`	The width of the stroke to be applied to the shape. Can be expressed in `px` or percentage units.
`stroke_opacity`	The opacity of the stroke of a shape. We can use a real number from `0` to `1` or a value in percentage units.
`fill`	The color used to fill the inside of the element.
`fill_opacity`	The opacity of the color or the content the current object is filled with. We can use a real number from `0` to `1` or a value in percentage units.
`font_family`	Which font family will be used to render the text of the element?
`font_size`	The size of the font.
`font_weight`	The weight or boldness of the font. Possible values are `"normal"`, `"bold"`, `"lighter"`, `"bolder"`, and the values `100`, `200`, and so on, up to `900`.
`font_style`	Whether a font should be styled with a `"normal"`, `"italic"`, or `"oblique"` face from its `font_family`.
`text_decoration`	Add decorative lines on text. Options are `"underline"`, `"overline"`, `"line-through"`, and `"blink"`.
`transform`	A list of transform definitions that are applied to an element and the element's children.
`filter`	The filter effects defined by a `<filter>` element that shall be applied to its element. Requires a reference to a `<filter>` `id` attribute.
`mask`	The mask defined by a `<mask>` element that shall be applied to its element. Requires a reference to a `<mask>` `id` attribute.
`clip_path`	The clipping path defined by a `<clipPath>` element that shall be applied to its element. Requires a reference to a `<clipPath>` `id` attribute.
`clip_rule`	A rule for determining what side of a path is inside of a shape in order to know how `clip_path` should clip its target. Options are `"nonzero"`, `"evenodd"`, and `"inherit"`.
`stroke_dasharray`	The pattern of dashes and gaps used to paint the outline of the shape.
`stroke_dashoffset`	Defines an offset on the rendering of the associated dash array.
`stroke_linecap`	The shape to be used at the end of open subpaths when they are stroked. We can use the options `"butt"`, `"round"`, or `"square"`.
`stroke_linejoin`	The shape to be used at the corners of paths when they are stroked (`"arcs"`, `"bevel"`, `"miter"`, `"miter-clip"`, and `"round"`).
`stroke_miterlimit`	The limit on the ratio of the miter length to the `stroke_width` Used to draw a miter join. A numeric value should be used to define the limit.

`fill_rule`	A rule for determining what side of a path is inside of a shape. Options are `"nonzero"`, `"evenodd"`, and `"inherit"`.
`color`	Potentially provides an indirect value (as the `currentColor`) for `fill`, `stroke`, `stop_color`, `flood_color` and `lighting_color` options.
`opacity`	Specifies the transparency of an object or a group of objects. We can use a real number from `0` to `1` or a value in percentage units.
`color_interpolation`	The color space for gradient interpolations, color animations, and alpha compositing. Allowed values are: `"auto"`, `"sRGB"`, `"linearRGB"`, and `"inherit"`.
`color_interpolation_filters`	The color space for imaging operations performed via filter effects. Allowed values are: `"auto"`, `"sRGB"`, `"linearRGB"`, and `"inherit"`.
`lighting_color`	The color of the light source for filter primitives elements `<feSpecularLighting>` and `<feDiffuseLighting>`.
`flood_color, flood_opacity`	The color and opacity level to use to flood the current filter primitive subregion defined through the `<feFlood>` or `<feDropShadow>` element.
`stop_color, stop_opacity`	Sets the color and opacity at a gradient stop.
`font_variant`	Determines whether a font should be used with some of their variation such as small caps or ligatures.
`font_stretch`	Allows for a selection of a normal, condensed, or expanded face from a font.
`font_size_adjust`	Specifies that the font size should be chosen based on the height of lowercase letters rather than the height of capital letters.
`text_anchor`	The vertical alignment a string of text. We can use the values `"start"`, `"middle"`, `"end"`, or `"inherit"`.
`letter_spacing, word_spacing`	The spacing between text characters and between words.
`dominant_baseline`	The baseline used to align the box's text and inline-level contents. The options for this are: `"auto"`, `"text-bottom"`, `"alphabetic"`, `"ideographic"`, `"middle"`, `"central"`, `"mathematical"`, `"hanging"`, and `"text-top"`.
`alignment_baseline`	Determines how an object is to be aligned along the font baseline with respect to its parent. Allowed values are: `"auto"`, `"baseline"`, `"before-edge"`, `"text-before-edge"`, `"middle"`, `"central"`, `"after-edge"`, `"text-after-edge"`, `"ideographic"`, `"alphabetic"`, `"hanging"`, `"mathematical"`, and `"inherit"`.
`baseline_shift`	An option for repositioning of the dominant-baseline relative to the dominant-baseline of the parent text content element. Valid options are: `"auto"`, `"baseline"`, `"super"`, `"sub"`, `"inherit"`, a length value, or a percentage value.
`direction`	The base writing direction of text. Can be either `"ltr"`, `"rtl"`, or `"inherit"`.
`writing_mode`	The initial inline-progression-direction for a `<text>` element (can be left-to-right, right-to-left, or top-to-bottom). Valid values are `"lr-tb"`, `"rl-tb"`, `"tb-rl"`, `"lr"`, `"rl"`, `"tb"`, or `"inherit"`.

`overflow`	The overflow behavior for the content of a block-level element when it overflows the element's box. Options are: `"visible"`, `"hidden"`, `"scroll"`, `"auto"`, and `"inherit"`.
`marker_start, marker_mid, marker_end`	The arrowhead or polymarker that will be drawn at the first node, the final node, or, the in-between nodes. This applies to a `<path>` element or a basic shape. These attributes can be applied to any element but only have an effect on the following seven elements: `<rect>`, `<circle>`, `<ellipse>`, `<line>`, `<path>`, `<polygon>`, and `<polyline>`. Requires a reference to a `<marker>` `id` attribute (defined within the SVG's `<defs>` area).
`pointer_events`	Defines whether or when an element may be the target of a mouse event. Options are: `"bounding-box"`, `"visiblePainted"`, `"visibleFil"`, `"visibleStroke"`, `"visible"`| `"painted"`, `"fill"`, `"stroke"`, `"all"`, and `"none"`.
`cursor`	The mouse cursor displayed when the mouse pointer is over an element.
`vector_effect`	The vector effect to use when drawing an object. Options are: `"default"`, `"non-scaling"`, `"stroke"`, and `"inherit"`.
`shape_rendering, color_rendering, text_rendering, image_rendering`	A quality setting parameter for shapes, color interpolation and compositing, text, and image processing. All of the rendering attributes can use the `"auto"` and `"optimizeSpeed"` directives. For shape rendering, we can elect for `"crispEdges"`, `"geometricPrecision"`, or just `"inherit"`. When rendering color, additional choices are `"optimizeQuality"` and `"inherit"`. Text rendering allows us the additional `"optimizeLegibility"`, `"geometricPrecision"`, and `"inherit"` options. With image rendering, we can furthermore choose to `"optimizeSpeed"`.
`display`	Allows for control of the rendering of graphical or container elements. A value of `"none"` indicates that the given element and its children will not be rendered. Any value other than `"none"` or `"inherit"` indicates that the given element will be rendered by the browser.
`visibility`	The visibility attribute lets us control the visibility of graphical elements. With a value of `"hidden"` or `"collapse"`, the element is invisible.

Value

A named list of presentational SVG properties. This object can be used as a value for the `attrs` argument, which is present in every SVG element function (e.g,. `svg_rect()`).

`svg_circle` *Addition of a* `circle` *element*

Description

The `svg_circle()` function adds a circle to an `svg` object. The position of the circle is given by `x` and `y`, and this refers to the center point of the point of the circle. The `diameter` of the circle is given in units of `px`.

Usage

```
svg_circle(
  svg,
  x,
  y,
  diameter,
  stroke = NULL,
  stroke_width = NULL,
  fill = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

`svg`	The `svg` object that is created using the `SVG()` function.
`x, y`	The `x` and `y` positions of the center of the circle to be drawn. The `x` and `y` values are relative to upper left of the SVG drawing area.
`diameter`	The diameter of the circle shape in units of `px`.
`stroke`	The color of the stroke applied to the element (i.e., the outline).
`stroke_width`	The width of the stroke in units of pixels.
`fill`	The fill color of the element.
`opacity`	The opacity of the element. Must be a value in the range of `0` to `1`.
`attrs`	A presentation attribute list. The helper function `svg_attrs_pres()` can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.
`anims`	An animation directive list for the element. This should be structured using the `anims()` function.
`filters`	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
`id`	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with a single
# circle element
svg <-
  SVG(width = 80, height = 80) %>%
    svg_circle(
      x = 30, y = 30,
      diameter = 40,
      stroke = "magenta",
      fill = "olive"
    )
}
```

---

svg_ellipse *Addition of an* `ellipse` *element*

Description

The `svg_ellipse()` function adds an ellipse to an `svg` object. The position of the ellipse is given by `x` and `y`, and they refer to the center point of the point of the ellipse. The `width` and the `height`, both in units of `px`, provide the horizontal and vertical extents of the ellipse.

Usage

```
svg_ellipse(
  svg,
  x,
  y,
  width,
  height,
  stroke = NULL,
  stroke_width = NULL,
  fill = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

`svg`	The `svg` object that is created using the `SVG()` function.
`x, y`	The `x` and `y` positions of the center of the ellipse to be drawn. The `x` and `y` values are relative to upper left of the SVG drawing area.

`width, height`	The `width` and `height` of the ellipse that is to be drawn. The `width` is the overall width of the ellipse in the 'x' direction, centered on point `x`. The `height` is the distance in the 'y' direction, centered on point `y`.
`stroke`	The color of the stroke applied to the element (i.e., the outline).
`stroke_width`	The width of the stroke in units of pixels.
`fill`	The fill color of the element.
`opacity`	The opacity of the element. Must be a value in the range of `0` to `1`.
`attrs`	A presentation attribute list. The helper function `svg_attrs_pres()` can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.
`anims`	An animation directive list for the element. This should be structured using the `anims()` function.
`filters`	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
`id`	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with a single
# ellipse element
svg <-
  SVG(width = 60, height = 60) %>%
    svg_ellipse(
      x = 30, y = 30,
      width = 50, height = 30,
      fill = "purple"
    )
}
```

`svg_filter` *Build an SVG* `<filter>`

Description

The `svg_filter()` let's us create a named `<filter>` element that we can apply to any SVG elements (such as shapes). We can bundle one or more filter elements by supplying a list of `filter_*()` calls to the `filters` argument.

Usage

```
svg_filter(svg, id, width = NULL, height = NULL, filters = list())
```

Arguments

`svg`	The svg object that is created using the `SVG()` function.
`id`	The ID value to assign to the filter. This must be provided and it should be unique among all <`filter`> elements.
`width, height`	The lengths of `width` and `height` define the extent of the filter.
`filters`	A list of `filter_*()` function calls. Examples include `filter_image()` and `filter_gaussian_blur()`.

Value

An svg object.

Examples

```
if (interactive()) {

# Set up an `svg_filter()` (called
# `"blur"`) that has the blur effect
# (using the `filter_gaussian_blur()`
# function); have the ellipse element
# use the filter by referencing it
# by name via the `"filter"` attribute
SVG(width = 200, height = 100) %>%
  svg_filter(
    id = "blur",
    filters = list(
      filter_gaussian_blur(stdev = 2)
    )
  ) %>%
  svg_ellipse(
    x = 40, y = 40,
    width = 50, height = 30,
    attrs = svg_attrs_pres(
      fill = "green",
      filter = "blur"
    )
  )
}
```

svg_group *Addition of a group element*

Description

The `svg_group()` function allows for grouping of several SVG elements. This is useful if we'd like to pass presentation attributes to several elements at once.

Usage

```
svg_group(
  svg,
  ...,
  .list = list2(...),
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

`svg`	The svg object that is created using the `SVG()` function.
`...`	a collection of named arguments that consist of presentation attributes (e.g., `stroke = "blue"`) and formulas that represent elements (e.g, `~ svg_rect(., x = 60, y = 60, width = 50, height = 50)`).
`.list`	Allows for the use of a list as an input alternative to `...`.
`attrs`	A presentation attribute list. The helper function `svg_attrs_pres()` can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.
`anims`	An animation directive list for the element. This should be structured using the `anims()` function.
`filters`	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
`id`	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with two rectangles
# contained within a group
SVG(width = 300, height = 300) %>%
  svg_group(
    fill = "steelblue", stroke = "red", opacity = 0.5,
    ~ svg_rect(., x = 20, y = 20, width = 50, height = 50),
    ~ svg_rect(., x = 40, y = 40, width = 50, height = 50, fill = "red")
  )

# Create an SVG with two rectangles
# that are nested within two
# different groups
SVG(width = 300, height = 300) %>%
  svg_group(
    fill = "green", stroke = "red",
    ~ svg_rect(., x = 30, y = 30, width = 40, height = 50),
    ~ svg_group(.,
      fill = "steelblue", opacity = 0.5,
      ~ svg_rect(., x = 60, y = 60, width = 50, height = 50)
      )
    )
}
```

`svg_image` *Addition of an* `image` *element*

Description

The `svg_image()` function adds an image to an `svg` object. The starting position is defined by `x` and `y` values. The image `width` and `height` are also required. All of these attributes are expressed in units of `px`.

Usage

```
svg_image(
  svg,
  x,
  y,
  image,
  width = NULL,
  height = NULL,
  preserve_aspect_ratio = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

svg	The svg object that is created using the `SVG()` function.
x, y	The x and y positions of the upper left of the image to be included. The x and y values are relative to upper left of the SVG drawing area itself.
image	The URL for the image file.
width, height	The width and height of the rectangle in which the image will be placed. If both are not provided, the image's original dimensions will be used. If one of these is provided, then the image will be scaled to the provided value with the aspect ratio intact. Providing both will result in the image placed in center of the rectangle with the aspect ratio preserved.
preserve_aspect_ratio	Controls how the aspect ratio of the image is preserved. Use `"none"` if the image's original aspect ratio should not be respected; this will fill the rectangle defined by `width` and `height` with the image (and this is only if both values are provided).
opacity	The opacity of the element. Must be a value in the range of `0` to `1`.
attrs	A presentation attribute list. The helper function `svg_attrs_pres()` can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.
anims	An animation directive list for the element. This should be structured using the `anims()` function.
filters	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
id	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with an SVG image
# (the R logo) contained within it
svg <-
  SVG(width = 300, height = 300) %>%
    svg_image(
      x = 20, y = 20,
      width = 100,
      height = 100,
      image = "https://www.r-project.org/logo/Rlogo.svg"
    )
}
```

`SVG_import` *Import an SVG file and create an* `svg` *object*

Description

Import an SVG file and create an `svg` object

Usage

```
SVG_import(
  data = NULL,
  width = NULL,
  height = NULL,
  viewbox = NULL,
  title = NULL,
  desc = NULL,
  incl_xmlns = FALSE,
  oneline = FALSE,
  anim_iterations = "infinite"
)
```

Arguments

`data`	Either a file path to an SVG file or the SVG code itself as a character vector of length 1.
`width`	The width and height attributes on the top-level <svg> element. Both of these attributes are optional but, if provided, take in a variety of dimensions and keywords. If numerical values are solely used, they are assumed to be 'px' length values. Dimensions can be percentage values (i.e., `"75%"`) or length values with the following units: `"em"`, `"ex"`, `"px"`, `"in"`, `"cm"`, `"mm"`, `"pt"`, and `"pc"`. Using `NULL`, the default, excludes the attribute.
`height`	The width and height attributes on the top-level <svg> element. Both of these attributes are optional but, if provided, take in a variety of dimensions and keywords. If numerical values are solely used, they are assumed to be 'px' length values. Dimensions can be percentage values (i.e., `"75%"`) or length values with the following units: `"em"`, `"ex"`, `"px"`, `"in"`, `"cm"`, `"mm"`, `"pt"`, and `"pc"`. Using `NULL`, the default, excludes the attribute.

viewbox	An optional set of dimensions that defines the SVG viewBox attribute. The viewBox for an SVG element is the position and dimension, in user space, of an SVG viewport. If supplied, this could either be in the form of a four-element, numeric vector corresponding to the "min-x", "min-y", "width", and "height" of the rectangle, or, as TRUE which uses the vector c(0, 0, width, height). Using NULL, the default, excludes this attribute.
title	The <title> tag for the finalized SVG.
desc	The <desc> tag for the finalized SVG.
incl_xmlns	Should the xmlns attribute be included in the <svg> tag? This attribute is only required on the outermost svg element of SVG documents, and, it's unnecessary for inner svg elements or inside of HTML documents. By default, this is set to FALSE.
oneline	An option to compress the resulting SVG tags such that they are reduced to one line.
anim_iterations	How many should an SVG animation (if defined by use of the anims() function) be played? By default this is "infinite" (i.e., looped indefinitely) but we can specify the animation iteration count as a positive number.

Value

An svg object.

SVG_la *Create an svg object with a Line Awesome glyph*

Description

Create an svg object with a Line Awesome glyph

Usage

```
SVG_la(
  name = "500px",
  height = "0.75em",
  width = NULL,
  viewbox = NULL,
  title = NULL,
  desc = NULL,
  incl_xmlns = FALSE,
  anim_iterations = "infinite"
)
```

Arguments

`name`	The name of the Line Awesome glyph.
`height`	The width and height attributes on the top-level <svg> element. Both of these attributes are optional but, if provided, take in a variety of dimensions and keywords. If numerical values are solely used, they are assumed to be 'px' length values. Dimensions can be percentage values (i.e., `"75%"`) or length values with the following units: `"em"`, `"ex"`, `"px"`, `"in"`, `"cm"`, `"mm"`, `"pt"`, and `"pc"`. Using `NULL`, the default, excludes the attribute.
`width`	The width and height attributes on the top-level <svg> element. Both of these attributes are optional but, if provided, take in a variety of dimensions and keywords. If numerical values are solely used, they are assumed to be 'px' length values. Dimensions can be percentage values (i.e., `"75%"`) or length values with the following units: `"em"`, `"ex"`, `"px"`, `"in"`, `"cm"`, `"mm"`, `"pt"`, and `"pc"`. Using `NULL`, the default, excludes the attribute.
`viewbox`	An optional set of dimensions that defines the SVG `viewBox` attribute. The `viewBox` for an SVG element is the position and dimension, in user space, of an SVG viewport. If supplied, this could either be in the form of a four-element, numeric vector corresponding to the `"min-x"`, `"min-y"`, `"width"`, and `"height"` of the rectangle, or, as `TRUE` which uses the vector `c(0, 0, width, height)`. Using `NULL`, the default, excludes this attribute.
`title`	The `<title>` tag for the finalized SVG.
`desc`	The <desc> tag for the finalized SVG.
`incl_xmlns`	Should the `xmlns` attribute be included in the <svg> tag? This attribute is only required on the outermost `svg` element of SVG documents, and, it's unnecessary for inner `svg` elements or inside of HTML documents. By default, this is set to `FALSE`.
`anim_iterations`	How many should an SVG animation (if defined by use of the `anims()` function) be played? By default this is `"infinite"` (i.e., looped indefinitely) but we can specify the animation iteration count as a positive number.

Value

An `svg` object.

`svg_line` *Addition of an* `line` *element*

Description

The `svg_line()` function adds a line to an `svg` object. The line is drawn using a start point (`x1` and `y1`) and an end point (`x2` and `y2`) points. These positions are in units of `px`.

Usage

```
svg_line(
  svg,
  x1,
  y1,
  x2,
  y2,
  stroke = NULL,
  stroke_width = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

`svg`	The svg object that is created using the `SVG()` function.
`x1, y1`	The `x` and `y` positions of the line's start point.
`x2, y2`	The `x` and `y` positions of the line's end point.
`stroke`	The color of the stroke applied to the element (i.e., the outline).
`stroke_width`	The width of the stroke in units of pixels.
`opacity`	The opacity of the element. Must be a value in the range of `0` to `1`.
`attrs`	A presentation attribute list. The helper function `svg_attrs_pres()` can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.
`anims`	An animation directive list for the element. This should be structured using the `anims()` function.
`filters`	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
`id`	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with a single
# line element
svg <-
  SVG(width = 100, height = 50) %>%
    svg_line(
      x1 = 5, y1 = 5,
      x2 = 95, y2 = 45,
      stroke = "blue"
    )
}
```

`svg_path` *Addition of an* `path` *element*

Description

The `svg_path()` function adds a path to an `svg` object. A path can potentially be quite complex (with an interplay of line and curve commands), so, a hand-encoded `path` string is not often done by hand. For this reason, the `path` argument accepts only a formatted string that complies with the input requirements for the `d` attribute of the SVG `<path>` tag. All point positions are in units of `px`.

Usage

```
svg_path(
  svg,
  path,
  stroke = NULL,
  stroke_width = NULL,
  fill = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

`svg`	The `svg` object that is created using the `SVG()` function.
`path`	A single-length character vector that holds the formatted path string.
`stroke`	The color of the stroke applied to the element (i.e., the outline).
`stroke_width`	The width of the stroke in units of pixels.
`fill`	The fill color of the element.
`opacity`	The opacity of the element. Must be a value in the range of `0` to `1`.
`attrs`	A presentation attribute list. The helper function `svg_attrs_pres()` can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.

`anims`	An animation directive list for the element. This should be structured using the `anims()` function.
`filters`	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
`id`	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with a single
# path element
svg <-
  SVG(width = 300, height = 300) %>%
    svg_path(
      path = "M 50 160 q 100 -300 200 0",
      stroke = "magenta",
      stroke_width = 5,
      fill = "lightblue"
    )
}
```

`svg_polygon` *Addition of an* `polygon` *element*

Description

The `svg_polygon()` function adds a polygon to an svg object. In the context of an SVG shape a polygon is similar to a polyline (defined by a series of points) except that the path will be automatically closed (i.e., last point connects to the first point). Like a polyline, a polygon is drawn by connecting a series of points with straight lines. The points can be provided as a vector that's exactly divisible by two, or, as a formatted string that adheres to the specification of the `points` attribute of the SVG <polygon> tag. All point positions are in units of px.

Usage

```
svg_polygon(
  svg,
  points,
  stroke = NULL,
  stroke_width = NULL,
  fill = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

svg	The svg object that is created using the SVG() function.
points	A numeric vector of points (with alternating values for x and y positions) that define the polygon. This can also be a single-length character vector that holds the formatted points string (space-separated x and y values, and comma-separated points).
stroke	The color of the stroke applied to the element (i.e., the outline).
stroke_width	The width of the stroke in units of pixels.
fill	The fill color of the element.
opacity	The opacity of the element. Must be a value in the range of 0 to 1.
attrs	A presentation attribute list. The helper function svg_attrs_pres() can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.
anims	An animation directive list for the element. This should be structured using the anims() function.
filters	A filter directive list for the element. This is easily created by using a list of filter_*() functions (e.g., list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))).
id	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with a single
# polygon element
svg <-
  SVG(width = 300, height = 300) %>%
    svg_polygon(
      points = "100,10 40,198 190,78 10,78 160,198",
      stroke = "orange",
      stroke_width = 4,
      fill = "yellow"
    )
}
```

svg_polyline *Addition of an* polyline *element*

Description

The svg_polyline() function adds a polyline to an svg object. The polyline is drawn by connecting a series of points with straight lines. The points can be provided as a vector that's exactly divisible by two, or, as a formatted string that adheres to the specification of the points attribute of the SVG <polyline> tag. All point positions are in units of px.

Usage

```
svg_polyline(
  svg,
  points,
  stroke = NULL,
  stroke_width = NULL,
  fill = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

svg	The svg object that is created using the SVG() function.
points	A numeric vector of points (with alternating values for x and y positions) that define the polyline. This can also be a single-length character vector that holds the formatted points string (space-separated x and y values, and comma-separated points).
stroke	The color of the stroke applied to the element (i.e., the outline).
stroke_width	The width of the stroke in units of pixels.
fill	The fill color of the element.
opacity	The opacity of the element. Must be a value in the range of 0 to 1.
attrs	A presentation attribute list. The helper function svg_attrs_pres() can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.

`anims`	An animation directive list for the element. This should be structured using the `anims()` function.
`filters`	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
`id`	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with a single
# polyline element; here `points`
# is a numeric vector where pairs
# of values are the `x` and `y`
# point position
svg_1 <-
  SVG(width = 300, height = 300) %>%
    svg_polyline(
      points = c(
        10, 10, 15, 20, 20, 15, 25, 30, 30, 25,
        35, 40, 40, 35, 45, 50, 50, 45
      ),
      stroke = "blue"
    )

# Create the same SVG with a single
# polyline element; this time `points`
# is a formatted points string
svg_2 <-
  SVG(width = 300, height = 300) %>%
    svg_polyline(
      points =
        "10,10 15,20 20,15 25,30 30,25 35,40 40,35 45,50 50,45",
      stroke = "blue"
    )
}
```

`svg_rect` *Addition of a* `rect` *element*

Description

The `svg_rect()` function adds a rectangle to an `svg` object. The position of the rectangle is given by `x` and `y`, and this refers to the upper left point of the rectangle. The `width` and the `height` are the dimensions of the rectangle. All of these dimensions are in units of `px`. The optional `rx` and `ry` parameter are corner radius values (again, in `px` units) that define x and y radius of the corners of the rectangle.

Usage

```
svg_rect(
  svg,
  x,
  y,
  width,
  height,
  rx = NULL,
  ry = NULL,
  stroke = NULL,
  stroke_width = NULL,
  fill = NULL,
  opacity = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

`svg`	The `svg` object that is created using the `SVG()` function.
`x, y`	The `x` and `y` positions of the upper left point of the rectangle to be drawn. The `x` and `y` values are relative to upper left of the SVG drawing area.
`width, height`	The `width` and `height` of the element that is to be drawn. The `width` is the distance in the 'x' direction from point `x` (proceeding right) and the `height` is the distance in the 'y' direction from point `y` (proceeding downward).
`rx, ry`	Optional corner radius values in the 'x' and 'y' directions. Applies to all corners of the rectangle. If only one value is provided (say, just for `rx`) then the unset value will take that set value as well.
`stroke`	The color of the stroke applied to the element (i.e., the outline).
`stroke_width`	The width of the stroke in units of pixels.
`fill`	The fill color of the element.
`opacity`	The opacity of the element. Must be a value in the range of `0` to `1`.
`attrs`	A presentation attribute list. The helper function `svg_attrs_pres()` can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.

`anims`	An animation directive list for the element. This should be structured using the `anims()` function.
`filters`	A filter directive list for the element. This is easily created by using a list of `filter_*()` functions (e.g., `list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))`).
`id`	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Examples

```
if (interactive()) {

# Create an SVG with a single
# rectangle element
svg_1 <-
  SVG(width = 100, height = 100) %>%
    svg_rect(
      x = 20, y = 10,
      width = 40, height = 15,
      stroke = "blue", fill = "yellow"
    )

# Create an SVG with a single
# rectangle element that moves
# to new `x` positions
svg_2 <-
  SVG(width = 300, height = 300) %>%
    svg_rect(
      x = 50, y = 50,
      width = 50, height = 50,
      stroke = "magenta", fill = "lightblue",
      anims = anims(
        0.5 ~ list(
          anim_position(
            x = 50, y = 50,
            easing_fn = ease_out()
          ),
          anim_rotation(rotation = 0)
        ),
        2.0 ~ list(
          anim_position(
            x = 200, y = 50,
            easing_fn = ease_in_out()
          ),
          anim_rotation(rotation = 90)
        )
      )
    )
}
```

SVG_t *Create a text-height* svg *object*

Description

The SVG_t() function is a variation on SVG() (the entry point for building an SVG) in that the output tags will be both as compact as possible (fewer linebreaks, less space characters) and the height is relative to line height of text (at "0.75em"). This is a good option if the eventual use for the generated SVG is to be integrated with text in HTML <p> elements. For scaling to function properly, the provision of the viewbox is required here.

Usage

```
SVG_t(height = "0.75em", viewbox)
```

Arguments

height	The height attribute on the top-level <svg> element. The default of "0.75em" is recommended here so that SVGs are scaled nicely to any adjacent text.
viewbox	An optional set of dimensions that defines the SVG viewBox attribute. The viewBox for an SVG element is the position and dimension, in user space, of an SVG viewport. If supplied, this could either be in the form of a four-element, numeric vector corresponding to the "min-x", "min-y", "width", and "height" of the rectangle, or, as TRUE which uses the vector c(0, 0, width, height). Using NULL, the default, excludes this attribute.

Value

An svg object.

Examples

```
if (interactive()) {

# Create a simple SVG with a rectangle and a circle
svg <-
  SVG_t(viewbox = c(0, 0, 60, 20)) %>%
  svg_rect(x = 0, y = 0, width = 30, height = 20) %>%
  svg_circle(x = 50, y = 10, diameter = 20)
}
```

svg_text — *Addition of a* text *element*

Description

The svg_text() function adds text to an svg object. As with many of the functions that create shape elements (such as svg_rect()), the starting position is defined by x and y values. All point positions are in units of px.

Usage

```
svg_text(
  svg,
  x,
  y,
  text,
  fill = NULL,
  opacity = NULL,
  path = NULL,
  attrs = list(),
  anims = list(),
  filters = list(),
  id = NULL
)
```

Arguments

svg	The svg object that is created using the SVG() function.
x, y	The x and y positions of the upper left of the text to be drawn. The x and y values are relative to upper left of the SVG drawing area itself.
text	A character vector that contains the text to be rendered.
fill	The color of the text.
opacity	The opacity of the element. Must be a value in the range of 0 to 1.
path	A single-length character vector that holds the formatted path string.
attrs	A presentation attribute list. The helper function svg_attrs_pres() can help us easily generate this named list object. For the most part, the list's names are the presentation attribute names and the corresponding values are the matching attribute values.
anims	An animation directive list for the element. This should be structured using the anims() function.
filters	A filter directive list for the element. This is easily created by using a list of filter_*() functions (e.g., list(filter_gaussian_blur(2), filter_drop_shadow(2, 2))).
id	An optional ID value to give to the built tag. This is useful for modifying this element in a later function call or for interacting with CSS.

Value

An svg object.

Index

anim_opacity, 5
anim_position, 6
anim_position(), 2, 10, 11, 18
anim_rotation, 7
anim_rotation(), 2
anim_scale, 8
anims, 2
anims(), 5–8, 20, 26, 28, 30, 32, 34–36, 38, 39, 41, 43, 45

cubic_bezier, 9

ease_in, 10
ease_in(), 5–8
ease_in_out, 10
ease_in_out(), 5–8
ease_out, 11
ease_out(), 5–8

filter_dilate, 11
filter_drop_shadow, 12
filter_erode, 13
filter_gaussian_blur, 14
filter_gaussian_blur(), 29
filter_image, 15
filter_image(), 29
filter_offset, 16

info_lineawesome, 17

linear, 18
linear(), 5–8

step_end, 18
step_end(), 5–8
step_start, 18
step_start(), 5–8
SVG, 19
SVG(), 20, 26, 27, 29, 30, 32, 36, 37, 39, 40, 42, 44, 45
SVG_, 20
svg_attrs_pres, 21
svg_attrs_pres(), 26, 28, 30, 32, 36, 37, 39, 40, 42, 45
svg_circle, 25
svg_circle(), 19
svg_ellipse, 27
svg_filter, 28
svg_group, 30
svg_image, 31
SVG_import, 33
SVG_la, 34
SVG_la(), 17
svg_line, 35
svg_path, 37
svg_polygon, 38
svg_polyline, 40
svg_rect, 41
svg_rect(), 19, 21, 25, 45
SVG_t, 44
svg_text, 45
svg_text(), 21