Package: minisvg (via r-universe)

May 24, 2026

Type Package

Title SVG Document Builder

Version 0.1.12

Description Build SVG documents with R.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

URL https://github.com/coolbutuseless/minisvg,https://coolbutuseless.github.io/package/minisvg/

Imports R6, glue

Suggests knitr, rmarkdown, xml2, purrr, base64enc, htmltools, rsvg, testthat (>= 2.1.0)

VignetteBuilder knitr

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

Date/Publication 2020-05-02 01:37:06 UTC

RemoteUrl https://github.com/coolbutuseless/minisvg

RemoteRef HEAD

RemoteSha d010891c3daab088699081710ba4707203cf44c9

Contents

add_method 2
as.character.svg_property 3
as.character.SVGDocument 3
as.character.SVGElement 4
as.character.SVGLiteral 4
as.character.SVGNode 5
knit_print.SVGDocument 5
knit_print.SVGFilter 6

knit_print.SVGPattern 6
parse_inner 7
parse_svg_doc 7
pres 8
print.svg_property 10
show_svg 10
stag 11
svg_prop 11
SVGDocument 11
SVGElement 13
SVGFilter 29
SVGLiteral 30
SVGNode 32
SVGPattern 36
SVGPatternList_to_svg 38

Index **39**

`add_method` *Add a method to the SVGElement class which creates a new child SVGElement of the specified type*

Description

Add a method to the SVGElement class which creates a new child SVGElement of the specified type

Usage

```
add_method(method_name, element_name, args)
```

Arguments

`method_name`	name of method in SVGElement class
`element_name`	the name of the SVG element to add
`args`	character vector of argument names

as.character.svg_property

Convert a CSS 'property' object to a string

Description

Convert a CSS 'property' object to a string

Usage

```
## S3 method for class 'svg_property'
as.character(x, ...)
```

Arguments

x	property object
...	other arguments

as.character.SVGDocument

Character representation of SVGDocument

Description

Character representation of SVGDocument

Usage

```
## S3 method for class 'SVGDocument'
as.character(x, include_declaration = TRUE, ...)
```

Arguments

x	SVGDocument
include_declaration	Include the SVG declaration at the top? Default: TRUE
...	other arguments

as.character.SVGElement

Character representation of SVGElement

Description

Character representation of SVGElement

Usage

```
## S3 method for class 'SVGElement'
as.character(x, ...)
```

Arguments

x	SVGElement
...	other arguments

as.character.SVGLiteral

Character representation of SVGLiteral

Description

Character representation of SVGLiteral

Usage

```
## S3 method for class 'SVGLiteral'
as.character(x, ...)
```

Arguments

x	SVGLiteral
...	other arguments

as.character.SVGNode *Character representation of SVGNode*

Description

Character representation of SVGNode

Usage

```
## S3 method for class 'SVGNode'
as.character(x, ...)
```

Arguments

x	SVGNode
...	other arguments

knit_print.SVGDocument *knitr/rmarkdown compatibility*

Description

knitr/rmarkdown compatibility

Usage

```
knit_print.SVGDocument(x, ...)
```

Arguments

x	SVGDocument
...	other arguments

knit_print.SVGFilter *knitr/rmarkdown compatibility*

Description

knitr/rmarkdown compatibility

Usage

```
knit_print.SVGFilter(x, ...)
```

Arguments

x	SVGFilter
...	other arguments

knit_print.SVGPattern *knitr/rmarkdown compatibility*

Description

knitr/rmarkdown compatibility

Usage

```
knit_print.SVGPattern(x, ...)
```

Arguments

x	SVGPattern
...	other arguments

parse_inner *Recursively parse an XML2 node tree into a 'minixml' document*

Description

This uses 'xml2' package to do the parsing.

Usage

```
parse_inner(xml2_node, as_document = TRUE, as_pattern = FALSE)
```

Arguments

xml2_node	root node of a document or an element node
as_document	parse the root node as a document node. Default: TRUE
as_pattern	parse the root node as a pattern node. Default: FALSE

parse_svg_doc *Parse SVG text or file into an SVGDocument or SVGElement*

Description

Parse SVG text or file into an SVGDocument or SVGElement

Usage

```
parse_svg_doc(x, encoding = "", ..., as_html = FALSE, options = "NOBLANKS")

parse_svg_elem(
  x,
  encoding = "",
  ...,
  as_html = FALSE,
  options = "NOBLANKS",
  as_pattern = FALSE
)
```

Arguments

x, encoding, ..., as_html, options	options passed to xml2::read_xml()
as_pattern	parse the root node as a pattern node. Default: FALSE

Value

XMLDocument or XMLElement

pres *Create a list of presentation attributes.*

Description

This function creates a named list. It's purpose is mainly as a helper - by having most presentation as named arguments we can use code auto-completion to help remember the 50+ possible attributes.

Usage

```
pres(
  ...,
  alignment_baseline,
  baseline_shift,
  clip_path,
  clip_rule,
  color,
  color_interpolation,
  color_interpolation_filters,
  color_profile,
  color_rendering,
  cursor,
  direction,
  display,
  dominant_baseline,
  enable_background,
  fill,
  fill_opacity,
  fill_rule,
  filter,
  flood_color,
  flood_opacity,
  font,
  font_family,
  font_size,
  font_size_adjust,
  font_stretch,
  font_style,
  font_variant,
  font_weight,
  glyph_orientation_vertical,
  image_rendering,
  kerning,
  letter_spacing,
  lighting_color,
  marker,
  marker_end,
```

```
  marker_mid,
  marker_start,
  mask,
  opacity,
  overflow,
  pointer_events,
  shape_rendering,
  stop_color,
  stop_opacity,
  stroke,
  stroke_dasharray,
  stroke_dashoffset,
  stroke_linecap,
  stroke_linejoin,
  stroke_miterlimit,
  stroke_opacity,
  stroke_width,
  text_anchor,
  text_decoration,
  text_rendering,
  unicode_bidi,
  visibility,
  word_spacing,
  writing_mode
)
```

Arguments

`...`	other named parameters
`alignment_baseline, baseline_shift, clip_path, clip_rule, color, color_interpolation, color_interpolation_filters, color_profile, color_rendering, cursor, direction, display, dominant_baseline, enable_background, fill, fill_opacity, fill_rule, filter, flood_color, flood_opacity, font, font_family, font_size, font_size_adjust, font_stretch, font_style, font_variant, font_weight, glyph_orientation_vertical, image_rendering, kerning, letter_spacing, lighting_color, marker, marker_end, marker_mid, marker_start, mask, opacity, overflow, pointer_events, shape_rendering, stop_color, stop_opacity, stroke, stroke_dasharray, stroke_dashoffset, stroke_linecap, stroke_linejoin, stroke_miterlimit, stroke_opacity, stroke_width, text_anchor, text_decoration, text_rendering, unicode_bidi, visibility, word_spacing, writing_mode`	named parameters (included to help when using auto-complete)

Details

For convenience, any underscores in the names will be replaced by dashes. This is because no sane CSS attributes are named with an underscore, but names with dashes are clunky to write in R.

Reference: https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/Presentation

Value

a list of presentation attributes

---

print.svg_property *Print a CSS 'property' object*

Description

Print a CSS 'property' object

Usage

```
## S3 method for class 'svg_property'
print(x, ...)
```

Arguments

x	property object
...	other arguments

---

show_svg *SVG shower*

Description

SVG shower

Usage

```
show_svg(svg, viewer = getOption("viewer", utils::browseURL))
```

Arguments

svg	SVG text or object
viewer	viewer

stag *SVG helper*

Description

SVG builder functions. Similar in purpose to `shiny::tags`, but with auto-complete of attribute names as part of the function call.

Examples

```
## Not run:
stag$circle(cx = 10, cy = 10, r = 15)
# <circle cx="10" cy="10" r="15" />

## End(Not run)
```

svg_prop *SVG property helper*

Description

Uses autocomplete to help write some standard propertys

Usage

```
svg_prop
```

Format

An object of class `list` of length 42.

SVGDocument *SVGDocument Class*

Description

SVGDocument Class

SVGDocument Class

Details

This is a specialized subclass of `SVGElement` containing some methods specific to the top level SVG node.

Has only been tested with MacOS and Rstudio

Super classes

`minisvg::SVGNode` -> `minisvg::SVGElement` -> `SVGDocument`

Public fields

`width`, `height` dimensions of document

Active bindings

`width`, `height` dimensions of document

Methods

Public methods:

- `SVGDocument$new()`
- `SVGDocument$save_html()`
- `SVGDocument$show()`
- `SVGDocument$inline_css()`
- `SVGDocument$print()`
- `SVGDocument$clone()`

Method `new()`**:** Initialise a new SVG document

Usage:

```
SVGDocument$new(
  ...,
  width = 400,
  height = 400,
  viewBox = NULL,
  preserveAspectRatio = NULL,
  xmlns = "http://www.w3.org/2000/svg",
  xmlns_xlink = "http://www.w3.org/1999/xlink"
)
```

Arguments:

`...` further arguments. Named arguments treated as attributes, unnamed arguments treated as child nodes

`width`, `height` SVG dimensions. default: 400x400

`viewBox` if NULL, then set to "0 0 width height"

`preserveAspectRatio`, `xmlns`, `xmlns_xlink` standard SVG attributes

Method `save_html()`**:** Save a complete HTML document containing this SVG document

Usage:

```
SVGDocument$save_html(filename)
```

Arguments:

`filename` HTML filename

Method `show()`**:** Render the SVG in the current viewer.

Usage:

```
SVGDocument$show(viewer = getOption("viewer", utils::browseURL))
```

Arguments:

`viewer` which viewer to use?

Method `inline_css()`**:** Use the supplied string as the inline CSS for this document

Usage:

```
SVGDocument$inline_css(css)
```

Arguments:

`css` string containing CSS

Method `print()`**:** Print the SVGDocument object

Usage:

```
SVGDocument$print(include_declaration = TRUE, ...)
```

Arguments:

`include_declaration` Include the XML declaration? default: TRUE

`...` other arguments passed to `$as_character()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SVGDocument$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`SVGElement` *A class representing a single SVG element.*

Description

A class representing a single SVG element.

A class representing a single SVG element.

Super class

`minisvg::SVGNode` -> `SVGElement`

Public fields

`name` Tag name for this node e.g. "circle"

`attribs` named list of attributes for this node

`children` ordered list of direct child nodes (kept in insertion order)

`child` lists of child nodes indexed by tag name.

Methods

Public methods:

- `SVGElement$a()`
- `SVGElement$animate()`
- `SVGElement$animateMotion()`
- `SVGElement$animateTransform()`
- `SVGElement$circle()`
- `SVGElement$clipPath()`
- `SVGElement$defs()`
- `SVGElement$desc()`
- `SVGElement$discard()`
- `SVGElement$ellipse()`
- `SVGElement$feBlend()`
- `SVGElement$feColorMatrix()`
- `SVGElement$feComponentTransfer()`
- `SVGElement$feComposite()`
- `SVGElement$feConvolveMatrix()`
- `SVGElement$feDiffuseLighting()`
- `SVGElement$feDisplacementMap()`
- `SVGElement$feFlood()`
- `SVGElement$feFuncR()`
- `SVGElement$feFuncG()`
- `SVGElement$feFuncB()`
- `SVGElement$feFuncA()`
- `SVGElement$feGaussianBlur()`
- `SVGElement$feImage()`
- `SVGElement$feMerge()`
- `SVGElement$feMergeNode()`
- `SVGElement$feMorphology()`
- `SVGElement$feOffset()`
- `SVGElement$feSpecularLighting()`
- `SVGElement$feTile()`
- `SVGElement$feTurbulence()`
- `SVGElement$filter()`
- `SVGElement$foreignObject()`

- SVGElement$g()
- SVGElement$image()
- SVGElement$line()
- SVGElement$linearGradient()
- SVGElement$marker()
- SVGElement$mask()
- SVGElement$mpath()
- SVGElement$path()
- SVGElement$pattern()
- SVGElement$radialGradient()
- SVGElement$rect()
- SVGElement$script()
- SVGElement$set()
- SVGElement$stop()
- SVGElement$style()
- SVGElement$switch()
- SVGElement$symbol()
- SVGElement$text()
- SVGElement$textPath()
- SVGElement$title()
- SVGElement$use()
- SVGElement$matrix()
- SVGElement$translate()
- SVGElement$scale()
- SVGElement$rotate()
- SVGElement$skewX()
- SVGElement$skewY()
- SVGElement$polygon()
- SVGElement$polyline()
- SVGElement$new()
- SVGElement$update()
- SVGElement$append()
- SVGElement$update_child_list()
- SVGElement$rebuild_child_list()
- SVGElement$add()
- SVGElement$remove()
- SVGElement$reset_transform()
- SVGElement$as_character_inner()
- SVGElement$as_character()
- SVGElement$save()
- SVGElement$has_attribs()
- SVGElement$find()

- SVGElement$copy()
- SVGElement$clone()

Method `a()`**:**

Usage:

```
SVGElement$a(..., href = NULL)
```

Method `animate()`**:**

Usage:

```
SVGElement$animate(
  ...,
  attributeName = NULL,
  attributeType = NULL,
  from = NULL,
  to = NULL,
  dur = NULL,
  repeatCount = NULL,
  begin = NULL,
  end = NULL,
  calcMode = NULL,
  values = NULL,
  keyTimes = NULL,
  keySplines = NULL,
  by = NULL
)
```

Method `animateMotion()`**:**

Usage:

```
SVGElement$animateMotion(
  ...,
  calcMode = NULL,
  path = NULL,
  keyPoints = NULL,
  rotate = NULL,
  origin = NULL
)
```

Method `animateTransform()`**:**

Usage:

```
SVGElement$animateTransform(
  ...,
  attributeName = NULL,
  attributeType = NULL,
  by = NULL,
  from = NULL,
  to = NULL,
  type = NULL,
```

```
  dur = NULL,
  repeatCount = NULL
)
```

Method `circle()`**:**

Usage:

```
SVGElement$circle(..., cx = NULL, cy = NULL, r = NULL)
```

Method `clipPath()`**:**

Usage:

```
SVGElement$clipPath(..., id = NULL)
```

Method `defs()`**:**

Usage:

```
SVGElement$defs(...)
```

Method `desc()`**:**

Usage:

```
SVGElement$desc(...)
```

Method `discard()`**:**

Usage:

```
SVGElement$discard(..., begin = NULL, href = NULL)
```

Method `ellipse()`**:**

Usage:

```
SVGElement$ellipse(..., cx = NULL, cy = NULL, rx = NULL, ry = NULL)
```

Method `feBlend()`**:**

Usage:

```
SVGElement$feBlend(..., in_ = NULL, in2 = NULL, mode = NULL, result = NULL)
```

Method `feColorMatrix()`**:**

Usage:

```
SVGElement$feColorMatrix(
  ...,
  in_ = NULL,
  type = NULL,
  values = NULL,
  result = NULL
)
```

Method `feComponentTransfer()`**:**

Usage:

```
SVGElement$feComponentTransfer(..., in_ = NULL, result = NULL)
```

Method `feComposite()`**:**

Usage:

```
SVGElement$feComposite(
  ...,
  in_ = NULL,
  in2 = NULL,
  operator = NULL,
  k1 = NULL,
  k2 = NULL,
  k3 = NULL,
  k4 = NULL,
  result = NULL
)
```

Method `feConvolveMatrix()`**:**

Usage:

```
SVGElement$feConvolveMatrix(
  ...,
  in_ = NULL,
  order = NULL,
  kernelMatrix = NULL,
  divisor = NULL,
  bias = NULL,
  targetX = NULL,
  targetY = NULL,
  edgeMode = NULL,
  kernelUnitLength = NULL,
  preserveAlpha = NULL,
  result = NULL
)
```

Method `feDiffuseLighting()`**:**

Usage:

```
SVGElement$feDiffuseLighting(
  ...,
  in_ = NULL,
  surfaceScale = NULL,
  diffuseConstant = NULL,
  kernelUnitLength = NULL,
  result = NULL
)
```

Method `feDisplacementMap()`**:**

Usage:

```
SVGElement$feDisplacementMap(
  ...,
  in_ = NULL,
```

```
  in2 = NULL,
  scale = NULL,
  xChannelSelector = NULL,
  yChannelSelector = NULL,
  result = NULL
)
```

Method `feFlood()`**:**

Usage:

```
SVGElement$feFlood(
  ...,
  flood_color = NULL,
  flood_opacity = NULL,
  result = NULL
)
```

Method `feFuncR()`**:**

Usage:

```
SVGElement$feFuncR(..., type = NULL, tableValues = NULL)
```

Method `feFuncG()`**:**

Usage:

```
SVGElement$feFuncG(..., type = NULL, tableValues = NULL)
```

Method `feFuncB()`**:**

Usage:

```
SVGElement$feFuncB(..., type = NULL, tableValues = NULL)
```

Method `feFuncA()`**:**

Usage:

```
SVGElement$feFuncA(..., type = NULL, tableValues = NULL)
```

Method `feGaussianBlur()`**:**

Usage:

```
SVGElement$feGaussianBlur(
  ...,
  in_ = NULL,
  stdDeviation = NULL,
  edgeMode = NULL,
  result = NULL
)
```

Method `feImage()`**:**

Usage:

```
SVGElement$feImage(
  ...,
  preserveAspectRatio = NULL,
  xlink_href = NULL,
  result = NULL
)
```

Method `feMerge()`**:**

Usage:

```
SVGElement$feMerge(..., result = NULL)
```

Method `feMergeNode()`**:**

Usage:

```
SVGElement$feMergeNode(..., in_ = NULL)
```

Method `feMorphology()`**:**

Usage:

```
SVGElement$feMorphology(
  ...,
  in_ = NULL,
  operator = NULL,
  radius = NULL,
  result = NULL
)
```

Method `feOffset()`**:**

Usage:

```
SVGElement$feOffset(..., in_ = NULL, dx = NULL, dy = NULL, result = NULL)
```

Method `feSpecularLighting()`**:**

Usage:

```
SVGElement$feSpecularLighting(
  ...,
  in_ = NULL,
  surfaceScale = NULL,
  specularConstant = NULL,
  specularExponent = NULL,
  kernelUnitLength = NULL,
  result = NULL
)
```

Method `feTile()`**:**

Usage:

```
SVGElement$feTile(..., in_ = NULL, result = NULL)
```

Method `feTurbulence()`**:**

Usage:

```
SVGElement$feTurbulence(
  ...,
  type = NULL,
  baseFrequency = NULL,
  numOctaves = NULL,
  seed = NULL,
  stitchTiles = NULL,
  result = NULL
)
```

Method `filter()`**:**

Usage:

```
SVGElement$filter(
  ...,
  id = NULL,
  x = NULL,
  y = NULL,
  width = NULL,
  height = NULL,
  filterRes = NULL,
  filterUnits = NULL,
  primitiveUnits = NULL,
  xlink_href = NULL
)
```

Method `foreignObject()`**:**

Usage:

```
SVGElement$foreignObject(..., x = NULL, y = NULL, width = NULL, height = NULL)
```

Method `g()`**:**

Usage:

```
SVGElement$g(..., id = NULL)
```

Method `image()`**:**

Usage:

```
SVGElement$image(
  ...,
  x = NULL,
  y = NULL,
  width = NULL,
  height = NULL,
  xlink_href = NULL,
  preserveAspectRatio = NULL
)
```

Method `line()`**:**

Usage:

```
SVGElement$line(..., x1 = NULL, y1 = NULL, x2 = NULL, y2 = NULL)
```

Method `linearGradient()`**:**

Usage:

```
SVGElement$linearGradient(
  ...,
  x1 = NULL,
  y1 = NULL,
  x2 = NULL,
  y2 = NULL,
  href = NULL,
  gradientTransform = NULL,
  gradientUnits = NULL,
  spreadMethod = NULL
)
```

Method `marker()`**:**

Usage:

```
SVGElement$marker(
  ...,
  refX = NULL,
  refY = NULL,
  markerWidth = NULL,
  markerHeight = NULL,
  markerUnits = NULL,
  orient = NULL,
  preserveAspectRatio = NULL,
  viewBox = NULL
)
```

Method `mask()`**:**

Usage:

```
SVGElement$mask(
  ...,
  x = NULL,
  y = NULL,
  width = NULL,
  height = NULL,
  maskUnits = NULL,
  maskContentUnits = NULL
)
```

Method `mpath()`**:**

Usage:

```
SVGElement$mpath(..., xlink_href = NULL)
```

Method `path()`**:**

Usage:

```
SVGElement$path(..., d = NULL, pathLength = NULL)
```

Method `pattern()`**:**

Usage:

```
SVGElement$pattern(
  ...,
  id = NULL,
  x = NULL,
  y = NULL,
  width = NULL,
  height = NULL,
  href = NULL,
  patternUnits = NULL,
  patternTransform = NULL,
  preserveAspectRatio = NULL
)
```

Method `radialGradient()`**:**

Usage:

```
SVGElement$radialGradient(
  ...,
  id = NULL,
  cx = NULL,
  cy = NULL,
  r = NULL,
  fx = NULL,
  fy = NULL,
  fr = NULL,
  href = NULL,
  gradientUnits = NULL,
  gradientTransform = NULL,
  spreadMethod = NULL
)
```

Method `rect()`**:**

Usage:

```
SVGElement$rect(
  ...,
  x = NULL,
  y = NULL,
  width = NULL,
  height = NULL,
  rx = NULL,
  ry = NULL
)
```

Method `script()`**:**

Usage:

```
SVGElement$script(..., type = NULL, href = NULL)
```

Method `set()`**:**

Usage:

```
SVGElement$set(..., to = NULL)
```

Method `stop()`**:**

Usage:

```
SVGElement$stop(..., offset = NULL, stop_color = NULL, stop_opacity = NULL)
```

Method `style()`**:**

Usage:

```
SVGElement$style(...)
```

Method `switch()`**:**

Usage:

```
SVGElement$switch(...)
```

Method `symbol()`**:**

Usage:

```
SVGElement$symbol(
  ...,
  id = NULL,
  x = NULL,
  y = NULL,
  width = NULL,
  height = NULL,
  refX = NULL,
  refY = NULL,
  preserveAspectRatio = NULL
)
```

Method `text()`**:**

Usage:

```
SVGElement$text(
  ...,
  x = NULL,
  y = NULL,
  dx = NULL,
  dy = NULL,
  rotate = NULL,
  textWidth = NULL,
  lengthAdjust = NULL
)
```

Method `textPath()`**:**

Usage:

```
SVGElement$textPath(
  ...,
  href = NULL,
  path = NULL,
  method = NULL,
  side = NULL,
  spacing = NULL,
  startOffset = NULL,
  textLength = NULL,
  lengthAdjust = NULL
)
```

Method `title()`**:**

Usage:

```
SVGElement$title(...)
```

Method `use()`**:**

Usage:

```
SVGElement$use(
  ...,
  x = NULL,
  y = NULL,
  width = NULL,
  height = NULL,
  href = NULL
)
```

Method `matrix()`**:**

Usage:

```
SVGElement$matrix(a, b, c, d, e, f)
```

Method `translate()`**:**

Usage:

```
SVGElement$translate(x, y = NULL)
```

Method `scale()`**:**

Usage:

```
SVGElement$scale(x, y = NULL)
```

Method `rotate()`**:**

Usage:

```
SVGElement$rotate(a, x = NULL, y = NULL)
```

Method `skewX()`**:**

Usage:

```
SVGElement$skewX(a)
```

Method `skewY()`**:**

Usage:

```
SVGElement$skewY(a)
```

Method `polygon()`**:**

Usage:

```
SVGElement$polygon(xs = NULL, ys = NULL, points = NULL, ...)
```

Method `polyline()`**:**

Usage:

```
SVGElement$polyline(xs = NULL, ys = NULL, points = NULL, ...)
```

Method `new()`**:** Initialize an SVGElement

Usage:

```
SVGElement$new(name, ...)
```

Arguments:

`name` node name e.g. "circle"

`...` further arguments. Named arguments treated as attributes, unnamed arguments treated as child nodes

Method `update()`**:** Update the SVG Element.

Usage:

```
SVGElement$update(...)
```

Arguments:

`...` attributes and children to set on this node

Details: Named arguments are considered attributes and will overwrite existing attributes with the same name. Set to NULL to delete the attribute
Unnamed arguments are appended to the list of child nodes. These should be text, other SVGElements or any ojbect that can be represented as a single text string using "as.character()"
To print just the attribute name, but without a value, set to NA

Method `append()`**:** Append child nodes at the specified position.

Usage:

```
SVGElement$append(..., position = NULL)
```

Arguments:

`...` child nodes

`position` by default at the end of the list of children nodes but 'position' argument can be used to set location by index

Method `update_child_list()`**:** Update the list of child nodes by tag name

Usage:

```
SVGElement$update_child_list(new_elem)
```

Arguments:

`new_elem` the element being added

Method `rebuild_child_list()`**:** URebuild the list of child nodes by tag name

Usage:

```
SVGElement$rebuild_child_list()
```

Method `add()`**:** Simultaneous create an SVG element and add it as a child node

Usage:

```
SVGElement$add(name, ...)
```

Arguments:

`name` name of node to create

`...` attributes and children of this newly created node

Returns: In contrast to most other methods, `$add()` returns the newly created element, *not* the document

Method `remove()`**:** Remove child objects at the given indices

Usage:

```
SVGElement$remove(indices)
```

Arguments:

`indices` indices of the children to remove

Method `reset_transform()`**:** Remove any transform attributes from this node

Usage:

```
SVGElement$reset_transform()
```

Method `as_character_inner()`**:** Recursively convert this SVGElement and children to text

Usage:

```
SVGElement$as_character_inner(..., depth = 0)
```

Arguments:

`...` ignored

`depth` recursion depth

Returns: single character string

Method `as_character()`**:** Recursively convert this SVGElement and children to text

Usage:

```
SVGElement$as_character(..., depth = 0, include_declaration = FALSE)
```

Arguments:

`...` ignored

`depth` recursion depth. default: 0

`include_declaration` Include the leading XML declaration? default: FALSE

Returns: single character string

Method `save()`**:** Save the text representation of this node and its children

Usage:

```
SVGElement$save(filename, include_declaration = FALSE, ...)
```

Arguments:

`filename` filename

`include_declaration` Include the leading XML declaration? default: FALSE

`...` Extra arguments passed to `SVGElement$as_character()`

Method `has_attribs()`**:** test if this element has all of the named attributes

Usage:

```
SVGElement$has_attribs(attribs)
```

Arguments:

`named` list of attributes

Returns: logical. Note: if `length(attribs) == 0`, this method returns TRUE

Method `find()`**:** Find elements which match the given tag names and attributes.

Usage:

```
SVGElement$find(tag = c(), attribs = list())
```

Arguments:

`tag` character vector of tags to find. default: c()

`attribs` named list of attributes to match. default: list(). Note that attribute matching is matched using `in`

Returns: List of R6 reference objects

Examples:

```
\dontrun{
doc$find(tag = c('rect', 'circle'), attribs = list(fill = c('red', 'black')))
}
```

Method `copy()`**:** Make a deep copy of this node and its children

Usage:

```
SVGElement$copy()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SVGElement$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SVGElement$find`
## ------------------------------------------------

## Not run:
doc$find(tag = c('rect', 'circle'), attribs = list(fill = c('red', 'black')))

## End(Not run)
```

SVGFilter *SVGFilter Class*

Description

SVGFilter Class

SVGFilter Class

Details

This is a slightly specialized subclass of `SVGElement` which has methods to specifically handle SVG `<filter>` nodes

Super classes

`minisvg::SVGNode` -> `minisvg::SVGElement` -> `SVGFilter`

Methods

Public methods:

- `SVGFilter$new()`
- `SVGFilter$as_full_svg()`
- `SVGFilter$save_full_svg()`
- `SVGFilter$show()`
- `SVGFilter$clone()`

Method `new()`**:** Initialise an SVGFilter object

Usage:

```
SVGFilter$new(..., name = "filter")
```

Arguments:

`...` Further arguments passed to `SVGElement$new()`

`name` defaults to 'filter'

Method `as_full_svg()`**:** Wrap the SVG for this filter in a full SVG document and return the text

Usage:

```
SVGFilter$as_full_svg(width = 400, height = width)
```

Arguments:

`height, width` dimensions of SVG wrapper around this filter

Method `save_full_svg()`**:** Save the SVG for this filter in a full SVG document

Usage:

```
SVGFilter$save_full_svg(filename, include_declaration = TRUE, ...)
```

Arguments:

`filename` filename for output

`include_declaration` Include leading XML declaration. default: TRUE

`...` Further arguments passed to `SVGFilter$as_full_svg()`

Method `show()`**:** Render this filter in the context of a complete SVG document

Usage:

```
SVGFilter$show(..., viewer = getOption("viewer", utils::browseURL))
```

Arguments:

`...` Further arguments passed to `SVGFilter$save_full_svg()`

`viewer` viewer.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SVGFilter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
f <- SVGFilter$new(id = "turbulence-filter", stag$feTurbulence(...))

## End(Not run)
```

`SVGLiteral` *A class representing a literal SVG element.*

Description

A class representing a literal SVG element.

A class representing a literal SVG element.

Super class

`minisvg::SVGNode` -> `SVGLiteral`

Public fields

`x` literal contents. Must be coercible to string via 'as.character(x)'

`name` kept only for compatibility with other SVGNode objects. This should be set to 'literal'

Methods

Public methods:

- `SVGLiteral$new()`
- `SVGLiteral$update()`
- `SVGLiteral$as_character_inner()`
- `SVGLiteral$as_character()`
- `SVGLiteral$clone()`

Method `new()`**:** Initialize an SVGElement

Usage:

```
SVGLiteral$new(x, ...)
```

Arguments:

`x` the literal text to include

`...` further arguments. Named arguments treated as attributes, unnamed arguments treated as child nodes

Method `update()`**:** Update the SVG Element.

Usage:

```
SVGLiteral$update(x, ...)
```

Arguments:

`x` the literal text to include

`...` attributes and children to set on this node

Details: Named arguments are considered attributes and will overwrite existing attributes with the same name. Set to NULL to delete the attribute
Unnamed arguments are appended to the list of child nodes. These should be text, other SVGElements or any ojbect that can be represented as a single text string using "as.character()"
To print just the attribute name, but without a value, set to NA

Method `as_character_inner()`**:** Convert this SVGLiteral

Usage:

```
SVGLiteral$as_character_inner(...)
```

Arguments:

`...` ignored

Returns: single character string

Method `as_character()`**:** Convert this SVGLiteral

Usage:

```
SVGLiteral$as_character(...)
```

Arguments:

`...` ignored

Returns: single character string

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SVGLiteral$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`SVGNode` *A class representing a single SVG element.*

Description

A class representing a single SVG element.

A class representing a single SVG element.

Public fields

`css_decls` character vector of css declaration text for this node

`css_urls` character vector of css urls for this node

`js_code` character vector of javascript code for this node

`js_urls` character vector of javascript urls for this node

Methods

Public methods:

- `SVGNode$new()`
- `SVGNode$update()`
- `SVGNode$add_css_url()`
- `SVGNode$add_css()`
- `SVGNode$add_js_code()`
- `SVGNode$add_js_url()`
- `SVGNode$get_css_decls()`
- `SVGNode$get_css_urls()`
- `SVGNode$get_js_code()`
- `SVGNode$get_js_urls()`
- `SVGNode$get_css_style()`

- SVGNode$get_js_style()
- SVGNode$as_character_inner()
- SVGNode$as_character()
- SVGNode$print()
- SVGNode$save()
- SVGNode$copy()
- SVGNode$find()
- SVGNode$clone()

Method `new()`**:** Initialize an SVGNode

Usage:

```
SVGNode$new()
```

Method `update()`**:** Update the SVG Element.

Usage:

```
SVGNode$update(...)
```

Arguments:

`...` attributes and children to set on this node

Details: Named arguments are considered attributes and will overwrite existing attributes with the same name. Set to NULL to delete the attribute
Unnamed arguments are appended to the list of child nodes. These should be text, other SVGElements or any ojbect that can be represented as a single text string using "as.character()"
To print just the attribute name, but without a value, set to NA

Method `add_css_url()`**:** Add a URL to a CSS style sheet

Usage:

```
SVGNode$add_css_url(css_url)
```

Arguments:

`css_url` URL to style sheet. e.g. `$add_css_url("css/local.css")`

Method `add_css()`**:** Add a CSS declaration for this element.

Usage:

```
SVGNode$add_css(css_decl)
```

Arguments:

`css_decl` CSS string, or object which can be coerced to character. e.g. `$add_dec("#thing {font-size: 27px}")`

Method `add_js_code()`**:** Add javascript code for this element

Usage:

```
SVGNode$add_js_code(js_code)
```

Arguments:

`js_code` character string containing javascript code.

Method `add_js_url()`**:** Add a javaxcript URL to load within the SVG

Usage:

```
SVGNode$add_js_url(js_url)
```

Arguments:

`js_url` URL to javascript code. e.g. `$add_js_url("example.org/eg.js")`

Method `get_css_decls()`**:** Create a CSS declaration string for inclusion in the character output for this element.

Usage:

```
SVGNode$get_css_decls()
```

Details: this includes all css for all child elements

Method `get_css_urls()`**:** Create a vector or urls for CSS inclustion

Usage:

```
SVGNode$get_css_urls()
```

Details: this includes all CSS URLs for all child elements

Method `get_js_code()`**:** Create a character vector of JS code for this node and all child nodes.

Usage:

```
SVGNode$get_js_code()
```

Method `get_js_urls()`**:** Create a vector of external JS urls

Usage:

```
SVGNode$get_js_urls()
```

Details: this includes all CSS URLs for all child elements

Method `get_css_style()`**:** Create a complete CSS <style> tag using the declarations and URLs of the current element, and all child elements.

Usage:

```
SVGNode$get_css_style()
```

Returns: character string

Method `get_js_style()`**:** Create a complete CSS style tag using the declarations and URLs of the current element, and all child elements.

Usage:

```
SVGNode$get_js_style()
```

Returns: character string

Method `as_character_inner()`**:** Recursively convert this SVGElement and children to text

Usage:

```
SVGNode$as_character_inner()
```

Returns: single character string

Method `as_character()`**:** Recursively convert this SVGElement and children to text

Usage:

```
SVGNode$as_character()
```

Returns: single character string

Method `print()`**:** Print the SVG string to the terminal

Usage:

```
SVGNode$print(include_declaration = FALSE, ...)
```

Arguments:

`include_declaration` Include the leading XML declaration? default: FALSE

`...` Extra arguments passed to `SVGElement$as_character()`

Method `save()`**:** Save the text representation of this node and its children

Usage:

```
SVGNode$save(filename, include_declaration = FALSE, ...)
```

Arguments:

`filename` filename

`include_declaration` Include the leading XML declaration? default: FALSE

`...` Extra arguments passed to `SVGElement$as_character()`

Method `copy()`**:** Make a deep copy of this node and its children

Usage:

```
SVGNode$copy()
```

Method `find()`**:** Find elements which match the given tags

Usage:

```
SVGNode$find(tags)
```

Arguments:

`tags` character vector of tags to accept

Returns: minisvg objects which inherit from SVGNode will return NULL unless this method is overridden.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SVGNode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

SVGPattern *SVGPattern Class*

Description

SVGPattern Class

SVGPattern Class

Details

This is a slightly specialized subclass of `SVGElement` which has methods to specifically handle SVG `<pattern>` nodes

SVGPattern objects may also have their own 'filter_def' filter definition.

Super classes

`minisvg::SVGNode` -> `minisvg::SVGElement` -> `SVGPattern`

Public fields

`filter_def` A filter definition to accompany this pattern

Methods

Public methods:

- `SVGPattern$new()`
- `SVGPattern$as_full_svg()`
- `SVGPattern$save_full_svg()`
- `SVGPattern$as_character()`
- `SVGPattern$show()`
- `SVGPattern$clone()`

Method `new()`**:** Initialise an SVGPattern object

Usage:

```
SVGPattern$new(..., name = "pattern")
```

Arguments:

`...` Further arguments passed to `SVGElement$new()`

`name` defaults to 'pattern', but some gradients may also be used here

Method `as_full_svg()`**:** Wrap the SVG for this pattern in a full SVG document and return the text

Usage:

```
SVGPattern$as_full_svg(width = 400, height = width)
```

Arguments:

`height, width` dimensions of SVG wrapper around this pattern

Method `save_full_svg()`**:** Save the SVG for this pattern in a full SVG document

Usage:

```
SVGPattern$save_full_svg(filename, include_declaration = TRUE, ...)
```

Arguments:

`filename` filename for output

`include_declaration` Include leading XML declaration. default: TRUE

`...` Further arguments passed to `SVGPattern$as_full_svg()`

Method `as_character()`**:** Recursively convert this SVGElement and children to text

Usage:

```
SVGPattern$as_character(..., depth = 0, include_declaration = FALSE)
```

Arguments:

`...` ignored

`depth` recursion depth. default: 0

`include_declaration` Include the leading XML declaration? default: FALSE

Returns: single character string

Method `show()`**:** Render this pattern in the context of a complete SVG document

Usage:

```
SVGPattern$show(..., viewer = getOption("viewer", utils::browseURL))
```

Arguments:

`...` Further arguments passed to `SVGPattern$save_full_svg()`

`viewer` viewer.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SVGPattern$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
a <- SVGPattern$new()
f <- stag$filter(id = "turbulence-filter", stag$feTurbulence(...))
a$filter_def <- f

## End(Not run)
```

SVGPatternList_to_svg *Create a complete SVG object from an svg pattern*

Description

Create a complete SVG object from an svg pattern

Usage

```
SVGPatternList_to_svg(pattern_list, width = 400, height = 400, ncol = 2, ...)
```

Arguments

pattern_list	svg_pattern_list object
width, height	the display width of the surrounding SVG wrapper. defualt: 400x400
ncol	number of columns. if NULL, then will use an auto-layout
...	other arguments ignored

Index

∗ **datasets**
svg_prop, 11

add_method, 2
as.character.svg_property, 3
as.character.SVGDocument, 3
as.character.SVGElement, 4
as.character.SVGLiteral, 4
as.character.SVGNode, 5

knit_print.SVGDocument, 5
knit_print.SVGFilter, 6
knit_print.SVGPattern, 6

minisvg::SVGElement, *12*, *29*, *36*
minisvg::SVGNode, *12, 13*, *29*, *31*, *36*

parse_inner, 7
parse_svg_doc, 7
parse_svg_elem *(*parse_svg_doc*)*, 7
pres, 8
print.svg_property, 10

show_svg, 10
stag, 11
svg_doc *(*SVGDocument*)*, 11
svg_elem *(*SVGElement*)*, 13
svg_filter *(*SVGFilter*)*, 29
svg_literal *(*SVGLiteral*)*, 30
svg_pattern *(*SVGPattern*)*, 36
svg_prop, 11
SVGDocument, 11
SVGElement, 13
SVGFilter, 29
SVGLiteral, 30
SVGNode, 32
SVGPattern, 36
SVGPatternList_to_svg, 38