

Package ‘linne’

July 22, 2025

Title Convenient 'CSS'

Version 0.0.2

Description Conveniently generate 'CSS' using R code.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1.9000

Imports R6, rlang, purrr, cli, shiny, magrittr

Suggests htmltools, knitr, rmarkdown, testthat, covr

VignetteBuilder knitr

URL <https://linne.john-coene.com/>

BugReports <https://github.com/JohnCoene/linne/issues>

NeedsCompilation no

Author John Coene [aut, cre]

Maintainer John Coene <jcoenep@gmail.com>

Repository CRAN

Date/Publication 2020-10-26 09:20:10 UTC

Contents

convenience	2
Linne	3
pipes	10
selectors	11
useLinne	12
when	12
Index	13

convenience

Convenience Functions

Description

Convenience functions for common operations.

Usage

`important_(value)`

`url_(value)`

`rgb_(r, g, b)`

`rgba_(r, g, b, a = 0.5)`

Arguments

`value` Value to use.

`r, g, b, a` Red, green, blue, and alpha values.

Functions

- `important_()` - Makes it such that the rule cannot be overwritten by other rules (other selections).
- `rgb_()`, `rgba_()` - Functions for red, green, blue and alpha for transparency.
- `url_()` - Wrap in a `url` CSS function call.

Examples

```
Linne$
new()$
rule(
  sel_id("id"),
  color = rgba_(255, 255, 255, .6),
  fontSize = important_(20)
)
```

Linne

Linne

Description

Generate CSS from R code. Initialise a new CSS environment with `new`, use `rule` to define CSS rules.

Attributes

There are hundreds of attributes to pass to the three-dot construct (`...`), a comprehensive list of them can be found on [w3schools](http://www.w3schools.com).

Note that Linne accepts camelCase for convenience, e.g.: `font-size` or `fontSize`.

Methods

Public methods:

- `Linne$define()`
- `Linne$rule()`
- `Linne$build()`
- `Linne$get_css()`
- `Linne$show_css()`
- `Linne$import()`
- `Linne$include()`
- `Linne$write()`
- `Linne$print()`
- `Linne$inject()`
- `Linne$clone()`

Method `define()`:

Usage:

```
Linne$define(...)
```

Arguments:

... Named variables to define.

Details: Define variables.

Returns: Self: the Linne object.

Examples:

```
Linne$new()$define(baseColor = "blue")
```

Method `rule()`:

Usage:

```
Linne$rule(selector, ...)
```

Arguments:

selector An object of class selector as returned by the `sel_*` family of functions.

... *Declarations:* properties and their values. This accepts camelcase, e.g.: `font-style` or `fontStyle`.

Details: Rule

Define a CSS rule.

Returns: Self: the Linne object.

Examples:

```
Linne$new()$rule(sel_id("myButton"), color = "blue", fontSize = 50)
```

Method `build()`:*Usage:*

```
Linne$build()
```

Details: Builds CSS

Builds the CSS from definitions and rules.

Examples:

```
Linne$
new()$
define(primary_color = 'red')$
rule(
  sel_id("myButton"),
  color = primary_color,
  fontSize = 50
)$
rule(
  sel_class("container"),
  backgroundColor = primary_color
)$
build()
```

Method `get_css()`:*Usage:*

```
Linne$get_css(build = TRUE)
```

Arguments:

build Whether to build the CSS with the build method.

Details: Retrieve the CSS

Returns: A string.

Examples:

```
Linne$new()$rule(sel_id("myId"), fontSize = 20)$get_css()
```

Method `show_css()`:*Usage:*

```
Linne$show_css(build = TRUE)
```

Arguments:

build Whether to build the CSS with the build method.

Details: Prints Generated CSS

Examples:

```
Linne$new()$rule(sel_id("myButton"), color = "blue")$show_css()
```

Method import():*Usage:*

```
Linne$import(url)
```

Arguments:

url URL to import.

Details: Import

Import from a url or path.

Examples:

```
Linne$new()$import('https://fonts.googleapis.com/css2?family=Roboto')
```

Method include():*Usage:*

```
Linne$include(build = TRUE)
```

Arguments:

build Whether to build the CSS with the build method.

Details: Include in Shiny

Includes the CSS in shiny, place the call to this method anywhere in the shiny UI.

Returns: [htmltools::tags](#)

Examples:

```
# generate CSS
css <- Linne$
  new()$
  define(grey = '#c4c4c4')$
  rule(
    sel_id("myButton"),
    backgroundColor = 'red',
    fontSize = 20,
    color = grey
  )$
  rule(
    sel_class("aClass"),
    color = grey
  )

# include in an app
library(shiny)
```

```

ui <- fluidPage(
  css$include(),
  h1("Some text", class = "aClass"),
  actionButton("myButton", "Am I red?", class = "aClass")
)

server <- function(input, output){
  output$myPlot <- renderPlot(plot(cars))
}

if(interactive())
  shinyApp(ui, server)

```

Method write():*Usage:*

```
Linne$write(path = "style.css", pretty = FALSE, build = TRUE)
```

Arguments:

path Path to file.

pretty Whether to keep tabs and newlines.

build Whether to build the CSS with the build method.

Details: Save

Write the CSS to file.

Examples:

```

if(interactive())
  Linne$new()$rule(sel_id("id"), fontStyle = "italic")$write("styles.css")

```

Method print():*Usage:*

```
Linne$print()
```

Details: Print

Prints information on the Linne object.

Method inject():*Usage:*

```
Linne$inject(build = TRUE, session = shiny::getDefaultReactiveDomain())
```

Arguments:

build Whether to build the CSS with the build method.

session A valid shiny session.

Details: Inject CSS

Dynamically inject CSS from the server of a shiny application.

Examples:

```

library(shiny)

ui <- fluidPage(
  useLinne(),
  actionButton("change", "Change me!")
)

server <- function(input, output){

  linne <- Linne$
    new()$
    rule(
      sel_id("change"),
      color = "white",
      backgroundColor = "black"
    )

  observeEvent(input$change, {
    linne$inject()
  })

}

if(interactive())
  shinyApp(ui, server)

```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```

## -----
## Method `Linne$define`
## -----

Linne$new()$define(baseColor = "blue")

## -----
## Method `Linne$rule`
## -----

Linne$new()$rule(sel_id("myButton"), color = "blue", fontSize = 50)

## -----
## Method `Linne$build`
## -----

```

```

Linne$
  new()$
  define(primary_color = 'red')$
  rule(
    sel_id("myButton"),
    color = primary_color,
    fontSize = 50
  )$
  rule(
    sel_class("container"),
    backgroundColor = primary_color
  )$
  build()

## -----
## Method `Linne$get_css`
## -----

Linne$new()$rule(sel_id("myId"), fontSize = 20)$get_css()

## -----
## Method `Linne$show_css`
## -----

Linne$new()$rule(sel_id("myButton"), color = "blue")$show_css()

## -----
## Method `Linne$import`
## -----

Linne$new()$import('https://fonts.googleapis.com/css2?family=Roboto')

## -----
## Method `Linne$include`
## -----

# generate CSS
css <- Linne$
  new()$
  define(grey = '#c4c4c4')$
  rule(
    sel_id("myButton"),
    backgroundColor = 'red',
    fontSize = 20,
    color = grey
  )$
  rule(
    sel_class("aClass"),
    color = grey
  )
)

# include in an app

```



```

library(shiny)

ui <- fluidPage(
  css$include(),
  h1("Some text", class = "aClass"),
  actionButton("myButton", "Am I red?", class = "aClass")
)

server <- function(input, output){
  output$myPlot <- renderPlot(plot(cars))
}

if(interactive())
  shinyApp(ui, server)

## -----
## Method `Linne$write`
## -----

if(interactive())
  Linne$new()$rule(sel_id("id"), fontStyle = "italic")$write("styles.css")

## -----
## Method `Linne$inject`
## -----

library(shiny)

ui <- fluidPage(
  useLinne(),
  actionButton("change", "Change me!")
)

server <- function(input, output){

  linne <- Linne$
    new()$
    rule(
      sel_id("change"),
      color = "white",
      backgroundColor = "black"
    )

  observeEvent(input$change, {
    linne$inject()
  })

}

if(interactive())
  shinyApp(ui, server)

```

pipes

Infixes

Description

Convenient pipes for more sophisticated selectors.

Usage

lhs %child% rhs

lhs %or% rhs

lhs %with% rhs

Arguments

lhs, rhs Selectors as returned by sel_* family of functions.

Operators

- **%child%** - Selects elements where right hand is child of left hand, e.g.: `sel_tag('div') %child% sel_class('aClass')` selects elements with aClass who are direct children of div tags.
- **%or%** - Select left hand or right hand, e.g.: `sel_id('myId') %or% sel_class('myClass')` will select both the element with the id and elements with the class. Ideal to select and apply rules multiple elements at once.
- **%with%** - Left hand selector with right hand selector, e.g.: `sel_tag('div') %with% sel_class('aClass')` selects a div with a class of aClass. Ideal to narrow down the selection.

Examples

```
# select all paragraph 'p' with "red" class
sel_tag("p") %with% sel_class("red")

# the other way around works equally well
sel_class("red") %with% sel_tag("p")

# select multiple elements
# where id = "x" or class = "center"
sel_id("x") %or% sel_class("center")

# select element with id = "x" and parent's id = "y"
sel_id("y") %child% sel_id("x")
```

selectors

Selectors

Description

Create selectors to select particular elements.

Usage

`sel_id(value)`

`sel_input(value)`

`sel_all()`

`sel_class(value)`

`sel_tag(value)`

`sel_attr(attribute, value = NULL, tag = NULL)`

Arguments

<code>value</code>	Value of selector.
<code>attribute</code>	Name of attribute.
<code>tag</code>	Name of tag.

Details

The functions will print in the console the CSS selector they compose.

Functions

- `sel_id()` - Select an object by its id, e.g.: `sel_id('btn')` selects `shiny::actionButton('btn', 'Button')`.
- `sel_all()` - Selects everything.
- `sel_input()` - Selects an input by its id, e.g.: `sel_id('txt')` selects `shiny::textInput('txt', 'Text')`.
- `sel_class()` - Select all elements bearing a specific class, e.g.: `sel_class('cls')`, selects `shiny::h1('hello', class = 'cls')`.
- `sel_tag()` - Select all tags, e.g.: `sel_tag('p')` selects `p('hello')`.
- `sel_attr()` - Select all tags with a specific attribute.

See Also

`%with%`, `%or%`, and `%child%` as well as `when_active()`, `when_hover()`, and `when_focus()` for more sophisticated element selection.

Examples

```
# select element where id = x
sel_id("x")

# select all elements with class = "container"
sel_class("container")
```

 useLinne

Dependency

Description

Imports dependencies necessary for the `inject` method to work. Place this function in your shiny UI.

Usage

```
useLinne()
```

 when

State

Description

Narrows selection to a specific state, e.g.: when it is hovered.

Usage

```
when_active(selector)
```

```
when_hover(selector)
```

```
when_focus(selector)
```

Arguments

`selector` as returned by [selectors](#).

Index

`%child%` (pipes), 10
`%or%` (pipes), 10
`%with%` (pipes), 10
`%child%`, 10, 11
`%or%`, 10, 11
`%with%`, 10, 11

convenience, 2

`htmltools::tags`, 5

`important_` (convenience), 2
`important_()`, 2

Linne, 3

pipes, 10

`rgb_` (convenience), 2
`rgb_()`, 2
`rgba_` (convenience), 2
`rgba_()`, 2

`sel_all` (selectors), 11
`sel_all()`, 11
`sel_attr` (selectors), 11
`sel_attr()`, 11
`sel_class` (selectors), 11
`sel_class()`, 11
`sel_id` (selectors), 11
`sel_id()`, 11
`sel_input` (selectors), 11
`sel_input()`, 11
`sel_tag` (selectors), 11
`sel_tag()`, 11
selectors, 11, 12

`url_` (convenience), 2
`url_()`, 2
`useLinne`, 12

when, 12

`when_active` (when), 12
`when_active()`, 11
`when_focus` (when), 12
`when_focus()`, 11
`when_hover` (when), 12
`when_hover()`, 11