

Package ‘sigmajs’

April 9, 2019

Title Interface to 'Sigma.js' Graph Visualization Library

Date 2019-04-08

Version 0.1.3

Description

Interface to 'sigma.js' graph visualization library including animations, plugins and shiny proxies.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 2.10)

URL <http://sigmajs.john-coene.com/>

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

Imports htmlwidgets, dplyr, magrittr, shiny, jsonlite, igraph,
htmltools, purrr, scales, crosstalk

Suggests knitr, rmarkdown, DT, testthat, covr

VignetteBuilder knitr

NeedsCompilation no

Author John Coene [aut, cre] (<<https://orcid.org/0000-0002-6637-4107>>)

Maintainer John Coene <jcoenep@gmail.com>

Repository CRAN

Date/Publication 2019-04-09 11:10:09 UTC

R topics documented:

color-scale	2
force	3
lesmis_edges	5
lesmis_igraph	5
lesmis_nodes	6

read	6
read-batch	8
read-static	9
sg_add_images	11
sg_add_nodes	11
sg_add_nodes_delay_p	13
sg_add_nodes_p	14
sg_add_node_p	15
sg_animate	15
sg_button	16
sg_change_nodes_p	18
sg_clear_p	19
sg_cluster	20
sg_custom_shapes	21
sg_drag_nodes	22
sg_drop_nodes	22
sg_drop_nodes_delay_p	23
sg_drop_nodes_p	25
sg_drop_node_p	25
sg_export_svg	26
sg_filter_gt_p	27
sg_from_gexf	28
sg_from_igraph	29
sg_get_nodes_p	29
sg_layout	30
sg_make_nodes	32
sg_neighbours	33
sg_nodes	33
sg_noverlap	35
sg_progress	36
sg_refresh_p	37
sg_relative_size	37
sg_settings	38
sg_zoom_p	38
sigmajs	39
sigmajs-shiny	40

Index	41
--------------	-----------

color-scale

Color

Description

Scale color by node size.

Usage

```
sg_scale_color(sg, pal)
```

Arguments

`sg` An object of class `sigmajsas` intatiated by `sigmajs`.

`pal` Vector of color.

Examples

```
nodes <- sg_make_nodes()
edges <- sg_make_edges(nodes, 20)

sigmajs() %>%
  sg_nodes(nodes, id, size) %>%
  sg_scale_color(pal = c("red", "blue"))
```

force	<i>Add forceAtlas2</i>
-------	------------------------

Description

Implementation of `forceAtlas2`.

Usage

```
sg_force(sg, ...)

sg_force_start(sg, ...)

sg_force_stop(sg, delay = 5000)

sg_force_restart_p(proxy, ..., refresh = TRUE)

sg_force_restart(sg, data, delay, cumsum = TRUE)

sg_force_start_p(proxy, ..., refresh = TRUE)

sg_force_stop_p(proxy)

sg_force_kill_p(proxy)

sg_force_config_p(proxy, ...)
```

Arguments

sg	An object of class <code>sigmajs</code> as initiated by <code>sigmajs</code> .
...	Any parameter, see official documentation .
delay	Milliseconds after which the layout algorithm should stop running.
proxy	An object of class <code>sigmajsProxy</code> as returned by <code>sigmajsProxy</code> .
refresh	Whether to refresh the graph after node is dropped, required to take effect.
data	<code>data.frame</code> holding delay column.
cumsum	Whether to compute the cumulative sum of the delay.

Details

The delay helps for build dynamic visualisations where nodes and edges do not appear all at the same time. How the delay works depends on the `cumsum` parameter. if TRUE the function computes the cumulative sum of the delay to effectively add each row one after the other: delay is thus applied at each row (number of seconds to wait before the row is added *since the previous row*). If FALSE this is the number of milliseconds to wait before the node or edge is added to the visualisation; delay is used as passed to the function.

Functions

- `sg_force`, `sg_force_start` starts the `forceAtlas2` layout
- `sg_force_stop` stops the `forceAtlas2` layout after a delay milliseconds
- `sg_force_restart_p` proxy to re-starts (kill then start) the `forceAtlas2` layout, the options you pass to this function are applied on restart. If `forceAtlas2` has not started yet it is launched.
- `sg_force_start_p` proxy to start `forceAtlas2`.
- `sg_force_stop_p` proxy to stop `forceAtlas2`.
- `sg_force_kill_p` proxy to completely stops the layout and terminates the associated worker. You can still restart it later, but a new worker will have to initialize.
- `sg_force_config_p` proxy to set configurations of `forceAtlas2`.
- `sg_force_restart` Restarts (kills then starts) `forceAtlas2` at given delay.

See Also

[official documentation](#)

Examples

```
nodes <- sg_make_nodes(50)
edges <- sg_make_edges(nodes, 100)

sigmajs() %>%
  sg_nodes(nodes, id, label, size) %>%
  sg_edges(edges, id, source, target) %>%
  sg_force() %>%
  sg_force_stop() # stop force after 5 seconds
```

lesmis_edges	<i>Edges from co-appearances of characters in "Les Miserables"</i>
--------------	--

Description

A graph where the nodes are characters in "Les Miserables" updated from its first encoding by Professor Donald Knuth, as part of the Stanford Graph Base (SGB)

Usage

```
lesmis_edges
```

Format

An igraph object with 181 nodes and 4 variables

source abbreviation of character name

target abbreviation of character name

id unique edge id

label edge label

Source

<https://github.com/MADStudioNU/lesmiserables-character-network>

lesmis_igraph	<i>Co-appearances of characters in "Les Miserables" as igraph object</i>
---------------	--

Description

A graph where the nodes are characters in "Les Miserables" updated from its first encoding by Professor Donald Knuth, as part of the Stanford Graph Base (SGB)

Usage

```
lesmis_igraph
```

Format

An igraph object with 181 nodes and 1589 edges

id abbreviation of character name

label character name

color random color

Source

<https://github.com/MADStudioNU/lesmiserables-character-network>

lesmis_nodes	<i>Nodes from co-appearances of characters in "Les Miserables"</i>
--------------	--

Description

A graph where the nodes are characters in "Les Miserables" updated from its first encoding by Professor Donald Knuth, as part of the Stanford Graph Base (SGB)

Usage

```
lesmis_nodes
```

Format

An igraph object with 181 nodes and 2 variables

id abbreviation of character name

label character name

Source

<https://github.com/MADStudioNU/lesmiserables-character-network>

read	<i>Read</i>
------	-------------

Description

Read nodes and edges to add to the graph. Other proxy methods to add data to a graph have to add nodes and edges one by one, thereby draining the browser, this method will add multiple nodes and edges more efficiently.

Usage

```
sg_read_nodes_p(proxy, data, ...)
```

```
sg_read_edges_p(proxy, data, ...)
```

```
sg_read_exec_p(proxy)
```

Arguments

proxy An object of class sigmajsProxy as returned by [sigmajsProxy](#).

data A data.frame of `_one_` node or edge.

... any column.

Functions

- `sg_read_nodes_p` read nodes.
- `sg_read_edges_p` read edges.
- `sg_read_exec_p` send read nodes and edges to JavaScript front end.

Examples

```

library(shiny)

ui <- fluidPage(
  actionButton("add", "add nodes & edges"),
  sigmajsOutput("sg")
)

server <- function(input, output, session){

  nodes <- sg_make_nodes()
  edges <- sg_make_edges(nodes)

  output$sg <- renderSigmajs({
    sigmajs() %>%
    sg_nodes(nodes, id, label, color, size) %>%
    sg_edges(edges, id, source, target) %>%
    sg_layout()
  })

  i <- 10

  observeEvent(input$add, {
    new_nodes <- sg_make_nodes()
    new_nodes$id <- as.character(as.numeric(new_nodes$id) + i)
    i <<- i + 10
    ids <- 1:(i)
    new_edges <- data.frame(
      id = as.character((i * 2 + 15):(i * 2 + 29)),
      source = as.character(sample(ids, 15)),
      target = as.character(sample(ids, 15))
    )

    sigmajsProxy("sg") %>%
    sg_force_kill_p() %>%
    sg_read_nodes_p(new_nodes, id, label, color, size) %>%
    sg_read_edges_p(new_edges, id, source, target) %>%
    sg_read_exec_p() %>%
    sg_force_start_p() %>%
    sg_refresh_p()
  })

}

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

```

read-batch

Batch read

Description

Read nodes and edges by batch with a delay.

Usage

```
sg_read_delay_nodes_p(proxy, data, ..., delay)
```

```
sg_read_delay_edges_p(proxy, data, ..., delay)
```

```
sg_read_delay_exec_p(proxy, refresh = TRUE)
```

Arguments

proxy	An object of class <code>sigmajProxy</code> as returned by sigmajProxy .
data	A data.frame of nodes or edges to add to the graph.
...	any column.
delay	Column name of containing batch identifier.
refresh	Whether to refresh the graph after each batch (delay) has been added to the graph. Note that this will also automatically restart any running force layout.

Details

Add nodes and edges with `sg_read_delay_nodes_p` and `sg_read_delay_edges_p` then execute (send to JavaScript end) with `sg_read_delay_exec_p`.

Examples

```
library(shiny)

ui <- fluidPage(
  actionButton("add", "add nodes & edges"),
  sigmajOutput("sg")
)

server <- function(input, output, session){

  output$sg <- renderSigmaj({
    sigmaj()
  })

  observeEvent(input$add, {
    nodes <- sg_make_nodes(50)
  })
}
```



```

nodes$batch <- c(
  rep(1000, 25),
  rep(3000, 25)
)

edges <- data.frame(
  id = 1:80,
  source = c(
    sample(1:25, 40, replace = TRUE),
    sample(1:50, 40, replace = TRUE)
  ),
  target = c(
    sample(1:25, 40, replace = TRUE),
    sample(1:50, 40, replace = TRUE)
  ),
  batch = c(
    rep(1000, 40),
    rep(3000, 40)
  )
) %>%
dplyr::mutate_all(as.character)

sigmajsProxy("sg") %>%
  sg_force_start_p() %>%
  sg_read_delay_nodes_p(nodes, id, color, label, size, delay = batch) %>%
  sg_read_delay_edges_p(edges, id, source, target, delay = batch) %>%
  sg_read_delay_exec_p() %>%
  sg_force_stop_p()
})

}

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

```

read-static

Read

Description

Read nodes and edges into your graph, with or without a delay.

Usage

```
sg_read_nodes(sg, data, ..., delay)
```

```
sg_read_edges(sg, data, ..., delay)
```

```
sg_read_exec(sg, refresh = TRUE)
```

Arguments

<code>sg</code>	An object of class <code>sigmajsas</code> initiated by <code>sigmajs</code> .
<code>data</code>	<code>Data.frame</code> (or list) of nodes or edges.
<code>...</code>	Any column name, see details.
<code>delay</code>	Column name containing delay in milliseconds.
<code>refresh</code>	Whether to refresh the <code>force</code> layout.

Functions

- `sg_read_nodes` read nodes.
- `sg_read_edges` read edges.
- `sg_read_exec` send read nodes and edges to JavaScript front end.

Examples

```

nodes <- sg_make_nodes(50)
nodes$batch <- c(
  rep(1000, 25),
  rep(3000, 25)
)

edges <- data.frame(
  id = 1:80,
  source = c(
    sample(1:25, 40, replace = TRUE),
    sample(1:50, 40, replace = TRUE)
  ),
  target = c(
    sample(1:25, 40, replace = TRUE),
    sample(1:50, 40, replace = TRUE)
  ),
  batch = c(
    rep(1000, 40),
    rep(3000, 40)
  )
) %>%
dplyr::mutate_all(as.character)

sigmajs() %>%
  sg_force_start() %>%
  sg_read_nodes(nodes, id, label, color, size, delay = batch) %>%
  sg_read_edges(edges, id, source, target, delay = batch) %>%
  sg_force_stop(4000) %>%
  sg_read_exec() %>%
  sg_button("read_exec", "Add nodes & edges")

```

sg_add_images	<i>Add images to nodes</i>
---------------	----------------------------

Description

Add images to nodes with the [Custom Shapes plugin](#).

Usage

```
sg_add_images(sg, data, url, ...)
```

Arguments

sg	An object of class <code>sigmajsas</code> intatiated by sigmajs .
data	<code>Data.frame</code> containing columns.
url	URL of image.
...	Any other column.

See Also

[Official documentation](#)

Examples

```
## Not run:  
demo("custom-shapes", package = "sigmajs")  
  
## End(Not run)
```

sg_add_nodes	<i>Add nodes and edges</i>
--------------	----------------------------

Description

Add nodes or edges.

Usage

```
sg_add_nodes(sg, data, delay, ..., cumsum = TRUE)
```

```
sg_add_edges(sg, data, delay, ..., cumsum = TRUE, refresh = FALSE)
```

Arguments

sg	An object of class <code>sigmajsas</code> intatiated by <code>sigmajs</code> .
data	<code>Data.frame</code> (or list) of nodes or edges.
delay	Column name containing delay in milliseconds.
...	Any column name, see details.
cumsum	Whether to compute the cumulative sum of the delay.
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed and restarted at every iteration.

Details

The delay helps for build dynamic visualisations where nodes and edges do not appear all at the same time. How the delay works depends on the `cumsum` parameter. if `TRUE` the function computes the cumulative sum of the delay to effectively add each row one after the other: delay is thus applied at each row (number of seconds to wait before the row is added *since the previous row*). If `FALSE` this is the number of milliseconds to wait before the node or edge is added to the visualisation; delay is used as passed to the function.

Examples

```
# initial nodes
nodes <- sg_make_nodes()

# additional nodes
nodes2 <- sg_make_nodes()
nodes2$id <- as.character(seq(11, 20))

# add delay
nodes2$delay <- runif(nrow(nodes2), 500, 1000)

sigmajs() %>%
  sg_nodes(nodes, id, label, size, color) %>%
  sg_add_nodes(nodes2, delay, id, label, size, color)

edges <- sg_make_edges(nodes, 25)
edges$delay <- runif(nrow(edges), 100, 2000)

sigmajs() %>%
  sg_force_start() %>%
  sg_nodes(nodes, id, label, size, color) %>%
  sg_add_edges(edges, delay, id, source, target, cumsum = FALSE) %>%
  sg_force_stop(2300) # stop after all edges added
```

sg_add_nodes_delay_p *Add nodes or edges with a delay*

Description

Proxies to dynamically add multiple nodes or edges to an already existing graph with a **delay** between each addition.

Usage

```
sg_add_nodes_delay_p(proxy, data, delay, ..., refresh = TRUE,  
  cumsum = TRUE)
```

```
sg_add_edges_delay_p(proxy, data, delay, ..., refresh = TRUE,  
  cumsum = TRUE)
```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by <code>sigmajsProxy</code> .
data	A <code>data.frame</code> of <code>_one_</code> node or edge.
delay	Column name containing delay in milliseconds.
...	any column.
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed and restarted at every iteration.
cumsum	Whether to compute the cumulative sum of the delay.

Details

The delay helps for build dynamic visualisations where nodes and edges do not appear all at the same time. How the delay works depends on the `cumsum` parameter. if `TRUE` the function computes the cumulative sum of the delay to effectively add each row one after the other: delay is thus applied at each row (number of seconds to wait before the row is added **since the previous row**). If `FALSE` this is the number of milliseconds to wait before the node or edge is added to the visualisation; delay is used as passed to the function.

Note

Have the parameters from your initial graph match that of the node you add, i.e.: if you pass `size` in your initial chart, make sure you also have it in your proxy.

Examples

```
## Not run:  
demo("add-nodes-delay", package = "sigmajs") # add nodes with a delay  
demo("add-edges-delay", package = "sigmajs") # add edges with a delay  
demo("add-delay", package = "sigmajs") # add nodes and edges with a delay
```

```
## End(Not run)
```

sg_add_nodes_p	<i>Add nodes or edges</i>
----------------	---------------------------

Description

Proxies to dynamically add *multiple* nodes or edges to an already existing graph.

Usage

```
sg_add_nodes_p(proxy, data, ..., refresh = TRUE, rate = "once")
```

```
sg_add_edges_p(proxy, data, ..., refresh = TRUE, rate = "once")
```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by sigmajsProxy .
data	A <code>data.frame</code> of nodes or edges.
...	any column.
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed and restarted at every iteration..
rate	Refresh rate, either once, the graph is refreshed after <code>data.frame</code> of nodes is added or at each iteration (row-wise). Only applies if <code>refresh</code> is set to <code>TRUE</code> .

Note

Have the parameters from your initial graph match that of the node you add, i.e.: if you pass `size` in your initial chart, make sure you also have it in your proxy.

Examples

```
## Not run:
demo("add-nodes", package = "sigmajs")
demo("add-edges", package = "sigmajs")

## End(Not run)
```

sg_add_node_p	<i>Add node or edge</i>
---------------	-------------------------

Description

Proxies to dynamically add a node or an edge to an already existing graph.

Usage

```
sg_add_node_p(proxy, data, ..., refresh = TRUE)
```

```
sg_add_edge_p(proxy, data, ..., refresh = TRUE)
```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by sigmajsProxy .
data	A <code>data.frame</code> of <code>_one_</code> node or edge.
...	any column.
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed.

Note

Have the parameters from your initial graph match that of the node you add, i.e.: if you pass `size` in your initial chart, make sure you also have it in your proxy.

Examples

```
## Not run:
demo("add-node", package = "sigmajs")
demo("add-edge", package = "sigmajs")
demo("add-node-edge", package = "sigmajs")

## End(Not run)
```

sg_animate	<i>Animate</i>
------------	----------------

Description

Animate graph components.

Usage

```
sg_animate(sg, mapping, options = list(easing = "cubicInOut"),
  delay = 5000)
```

Arguments

sg	An object of class <code>sigmajs</code> as initiated by <code>sigmajs</code> .
mapping	Variables to map animation to.
options	Animations options.
delay	Delay in milliseconds before animation is triggered.

Details

You can animate, x, y, size and color.

See Also

[official documentation](#)

Examples

```
# generate graph
nodes <- sg_make_nodes(20)
edges <- sg_make_edges(nodes, 30)

# add transition
n <- nrow(nodes)
nodes$to_x <- runif(n, 5, 10)
nodes$to_y <- runif(n, 5, 10)
nodes$to_size <- runif(n, 5, 10)

sigmajs() %>%
  sg_nodes(nodes, id, label, size, color, to_x, to_y, to_size) %>%
  sg_edges(edges, id, source, target) %>%
  sg_animate(mapping = list(x = "to_x", y = "to_y", size = "to_size"))
```

sg_button

Buttons

Description

Add buttons to your graph.

Usage

```
sg_button(sg, event, ..., position = "top", class = "btn btn-default",
  tag = htmltools::tags$button, id = NULL)
```


Arguments

sg	An object of class <code>sigmajs</code> as initiated by <code>sigmajs</code> .
event	Event the button triggers, see valid events.
...	Content of the button, compliant with <code>htmltools</code> .
position	Position of button, top or bottom.
class	Button CSS class, see note.
tag	A Valid <code>tags</code> function.
id	A valid CSS id.

Details

You can pass multiple events as a vector, see examples. You can also pass multiple buttons.

Events

- force_start
- force_stop
- noverlap
- drag_nodes
- relative_size
- add_nodes
- add_edges
- drop_nodes
- drop_edges
- animate
- export_svg
- export_img
- progress
- read_exec

Note

The default class (`btn btn-default`) works with Bootstrap 3 (the default framework for Shiny and R markdown).

Examples

```
nodes <- sg_make_nodes()
edges <- sg_make_edges(nodes)

# Button starts the layout and stops it after 3 seconds
sigmajs() %>%
  sg_nodes(nodes, id, size) %>%
  sg_edges(edges, id, source, target) %>%
```

```

sg_force_start() %>%
sg_force_stop(3000) %>%
sg_button(c("force_start", "force_stop"), "start layout")

# additional nodes
nodes2 <- sg_make_nodes()
nodes2$id <- as.character(seq(11, 20))

# add delay
nodes2$delay <- runif(nrow(nodes2), 500, 1000)

sigmaj() %>%
  sg_nodes(nodes, id, label, size, color) %>%
  sg_add_nodes(nodes2, delay, id, label, size, color) %>%
  sg_force_start() %>%
  sg_force_stop(3000) %>%
  sg_button(c("force_start", "force_stop"), "start layout") %>%
  sg_button("add_nodes", "add nodes")

```

sg_change_nodes_p *Change*

Description

Change nodes and edges attributes on the fly

Usage

```
sg_change_nodes_p(proxy, data, value, attribute, rate = c("once",
  "iteration"), refresh = TRUE)
```

```
sg_change_edges_p(proxy, data, value, attribute, rate = c("once",
  "iteration"), refresh = TRUE)
```

Arguments

proxy	An object of class <code>sigmajProxy</code> as returned by sigmajProxy .
data	<code>data.frame</code> holding delay column.
value	Column containing value.
attribute	Name of attribute to change.
rate	Rate at which to refresh takes once refreshes once after all values have been changed, and iteration which refreshes at every iteration.
refresh	Whether to refresh the graph after the change is made.

Examples

```
library(shiny)

nodes <- sg_make_nodes()
nodes$new_color <- "red"
edges <- sg_make_edges(nodes)

ui <- fluidPage(
  actionButton("start", "Change color"),
  sigmajsOutput("sg")
)

server <- function(input, output){

  output$sg <- renderSigmajs({
    sigmajs() %>%
      sg_nodes(nodes, id, size, color) %>%
      sg_edges(edges, id, source, target)
  })

  observeEvent(input$start, {
    sigmajsProxy("sg") %>% # use sigmajsProxy!
      sg_change_nodes_p(nodes, new_color, "color")
  })

}

if(interactive()) shinyApp(ui, server) # run
```

sg_clear_p

Clear or kill the graph

Description

Clear all nodes and edges from the graph or kills the graph.

Kill the graph to ensure new data is redrawn, useful in Shiny when graph is not updated by [sigmajsProxy](#).

Usage

```
sg_clear_p(proxy, refresh = TRUE)
```

```
sg_kill_p(proxy, refresh = TRUE)
```

```
sg_kill(sg)
```

```
sg_clear(sg)
```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by sigmajsProxy .
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed and restarted.
sg	An object of class <code>sigmajsas</code> intatiated by sigmajs .

Examples

```
## Not run:
demo("clear-graph", package = "sigmajs")

## End(Not run)
```

sg_cluster	<i>Cluster</i>
------------	----------------

Description

Color nodes by cluster.

Usage

```
sg_cluster(sg, colors = c("#B1E2A3", "#98D3A5", "#328983", "#1C5C70",
  "#24C96B"), directed = TRUE, algo = igraph::cluster_walktrap,
  quiet = !interactive(), save_igraph = TRUE, ...)

sg_get_cluster(nodes, edges, colors = c("#B1E2A3", "#98D3A5", "#328983",
  "#1C5C70", "#24C96B"), directed = TRUE,
  algo = igraph::cluster_walktrap, quiet = !interactive(),
  save_igraph = TRUE, ...)
```

Arguments

sg	An object of class <code>sigmajsas</code> intatiated by sigmajs .
colors	Palette to color the nodes.
directed	Whether or not to create a directed graph, passed to graph_from_data_frame .
algo	An igraph clustering function.
quiet	Set to TRUE to print the number of clusters to the console.
save_igraph	Whether to save the igraph object used internally.
...	Any parameter to pass to <code>algo</code> .
nodes, edges	Nodes and edges as prepared for sigmajs .

Details

The package uses `igraph` internally for a lot of computations the `save_igraph` allows saving the object to speed up subsequent computations.

Value

`sg_get_cluster` returns nodes with color variable.

Functions

- `sg_cluster` Color nodes by cluster.
- `sg_get_cluster` helper to get graph's nodes color by cluster.

Examples

```
nodes <- sg_make_nodes()
edges <- sg_make_edges(nodes, 15)

sigmajs() %>%
  sg_nodes(nodes, id, size) %>%
  sg_edges(edges, id, source, target) %>%
  sg_layout() %>%
  sg_cluster()

clustered <- sg_get_cluster(nodes, edges)
```

<code>sg_custom_shapes</code>	<i>Custom shapes</i>
-------------------------------	----------------------

Description

Indicate a graph uses custom shapes

Usage

```
sg_custom_shapes(sg)
```

Arguments

`sg` An object of class `sigmajsas` intiated by `sigmajs`.

sg_drag_nodes	<i>Drag nodes</i>
---------------	-------------------

Description

Allow user to drag and drop nodes.

Usage

```
sg_drag_nodes(sg)

sg_drag_nodes_start_p(proxy)

sg_drag_nodes_kill_p(proxy)
```

Arguments

sg	An object of class <code>sigmajs</code> as initiated by <code>sigmajs</code> .
proxy	An object of class <code>sigmajsProxy</code> as returned by <code>sigmajsProxy</code> .

Examples

```
# generate graph
nodes <- sg_make_nodes(20)
edges <- sg_make_edges(nodes, 35)

sigmajs() %>%
  sg_nodes(nodes, id, label, size) %>%
  sg_edges(edges, id, source, target) %>%
  sg_drag_nodes()

## Not run:
# proxies
demo("drag-nodes", package = "sigmajs")

## End(Not run)
```

sg_drop_nodes	<i>Drop</i>
---------------	-------------

Description

Drop nodes or edges.

Usage

```
sg_drop_nodes(sg, data, ids, delay, cumsum = TRUE)

sg_drop_edges(sg, data, ids, delay, cumsum = TRUE, refresh = FALSE)
```

Arguments

sg	An object of class <code>sigmaajs</code> as initiated by <code>sigmaajs</code> .
data	Data.frame (or list) of nodes or edges.
ids	Ids of elements to drop.
delay	Column name containing delay in milliseconds.
cumsum	Whether to compute the cumulative sum of the delay.
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed and restarted at every iteration.

Details

The delay helps for build dynamic visualisations where nodes and edges do not disappear all at the same time. How the delay works depends on the `cumsum` parameter. if `TRUE` the function computes the cumulative sum of the delay to effectively drop each row one after the other: delay is thus applied at each row (number of seconds to wait before the row is dropped *since the previous row*). If `FALSE` this is the number of milliseconds to wait before the node or edge is dropped to the visualisation; delay is used as passed to the function.

Examples

```
nodes <- sg_make_nodes(75)

# nodes to drop
nodes2 <- nodes[sample(nrow(nodes), 50), ]
nodes2$delay <- runif(nrow(nodes2), 1000, 3000)

sigmaajs() %>%
  sg_nodes(nodes, id, size, color) %>%
  sg_drop_nodes(nodes2, id, delay, cumsum = FALSE)
```

sg_drop_nodes_delay_p *Drop nodes or edges with a delay*

Description

Proxies to dynamically drop multiple nodes or edges to an already existing graph with a *delay* between each removal.

Usage

```
sg_drop_nodes_delay_p(proxy, data, ids, delay, refresh = TRUE,
  cumsum = TRUE)
```

```
sg_drop_edges_delay_p(proxy, data, ids, delay, refresh = TRUE,
  cumsum = TRUE)
```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by <code>sigmajsProxy</code> .
data	A <code>data.frame</code> of <code>_one_</code> node or edge.
ids	Ids of elements to drop.
delay	Column name containing delay in milliseconds.
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed and restarted at every iteration.
cumsum	Whether to compute the cumulative sum of the delay.

Details

The delay helps for build dynamic visualisations where nodes and edges do not disappear all at the same time. How the delay works depends on the `cumsum` parameter. if `TRUE` the function computes the cumulative sum of the delay to effectively drop each row one after the other: delay is thus applied at each row (number of seconds to wait before the row is dropped *since the previous row*). If `FALSE` this is the number of milliseconds to wait before the node or edge is added to the visualisation; delay is used as passed to the function.

Note

Have the parameters from your initial graph match that of the node you add, i.e.: if you pass `size` in your initial chart, make sure you also have it in your proxy.

Examples

```
## Not run:
demo("drop-nodes-delay", package = "sigmajs") # add nodes with a delay
demo("drop-edges-delay", package = "sigmajs") # add edges with a delay
demo("drop-delay", package = "sigmajs") # add nodes and edges with a delay

## End(Not run)
```

sg_drop_nodes_p *Drop nodes or edges*

Description

Proxies to dynamically drop *multiple* nodes or edges from an already existing graph.

Usage

```
sg_drop_nodes_p(proxy, data, ids, refresh = TRUE, rate = "once")
```

```
sg_drop_edges_p(proxy, data, ids, refresh = TRUE, rate = "once")
```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by sigmajsProxy .
data	A <code>data.frame</code> of nodes or edges.
ids	Column containing ids to drop from the graph.
refresh	Whether to refresh the graph after node is dropped, required to take effect.
rate	Refresh rate, either once, the graph is refreshed after <code>data.frame</code> of nodes is added or at each iteration (row-wise). Only applies if <code>refresh</code> is set to <code>TRUE</code> .

Note

Have the parameters from your initial graph match that of the node you add, i.e.: if you pass `size` in your initial chart, make sure you also have it in your proxy.

sg_drop_node_p *Remove node or edge*

Description

Proxies to dynamically remove a node or an edge to an already existing graph.

Usage

```
sg_drop_node_p(proxy, id, refresh = TRUE)
```

```
sg_drop_edge_p(proxy, id, refresh = TRUE)
```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by sigmajsProxy .
id	Id of edge or node to delete.
refresh	Whether to refresh the graph after node is dropped, required to take effect, if you are running force the algorithm is killed and restarted.

Examples

```
## Not run:
demo("drop-node", package = "sigmajs")

## End(Not run)
```

sg_export_svg

Export

Description

Export graph to SVG.

Usage

```
sg_export_svg(sg, download = TRUE, file = "graph.svg", size = 1000,
  width = 1000, height = 1000, labels = FALSE, data = FALSE)
```

```
sg_export_img(sg, download = TRUE, file = "graph.png",
  background = "white", format = "png", labels = FALSE)
```

```
sg_export_img_p(proxy, download = TRUE, file = "graph.png",
  background = "white", format = "png", labels = FALSE)
```

```
sg_export_svg_p(proxy, download = TRUE, file = "graph.svg",
  size = 1000, width = 1000, height = 1000, labels = FALSE,
  data = FALSE)
```

Arguments

sg	An object of class <code>sigmajs</code> as instantiated by <code>sigmajs</code> .
download	set to TRUE to download.
file	Name of file.
size	Size of the SVG in pixels.
width, height	Width and height of the SVG in pixels.
labels	Whether the labels should be included in the svg file.
data	Whether additional data (node ids for instance) should be included in the svg file.
background	Background color of image.
format	Format of image, takes png, jpg, gif or tiff.
proxy	An object of class <code>sigmajsProxy</code> as returned by <code>sigmajsProxy</code> .

Examples

```

nodes <- sg_make_nodes()
edges <- sg_make_edges(nodes, 17)

sigmajs() %>%
  sg_nodes(nodes, id, size) %>%
  sg_edges(edges, id, source, target) %>%
  sg_export_svg() %>%
  sg_button("export_svg", "download")

# demo("export-graph", package = "sigmajs")

```

sg_filter_gt_p	<i>Filter</i>
----------------	---------------

Description

Filter nodes and/or edges.

Usage

```

sg_filter_gt_p(proxy, input, var, target = c("nodes", "edges", "both"),
  name = NULL)

sg_filter_lt_p(proxy, input, var, target = c("nodes", "edges", "both"),
  name = NULL)

sg_filter_eq_p(proxy, input, var, target = c("nodes", "edges", "both"),
  name = NULL)

sg_filter_not_eq_p(proxy, input, var, target = c("nodes", "edges",
  "both"), name = NULL)

sg_filter_undo_p(proxy, name)

sg_filter_neighbours_p(proxy, node, name = NULL)

```

Arguments

proxy	An object of class <code>sigmajsProxy</code> as returned by <code>sigmajsProxy</code> .
input	A Shiny input.
var	Variable to filter.
target	Target of filter, nodes, edges, or both.
name	Name of the filter, useful to undo the filter later on with <code>sg_filter_undo</code> .
node	Node id to filter neighbours.

Functions

- `sg_filter_gt_p` Filter greater than var.
- `sg_filter_lt_p` Filter less than var.
- `sg_filter_eq_p` Filter equal to var.
- `sg_filter_not_eq_p` Filter not equal to var.
- `sg_filter_undo_p` Undo filters, accepts vector of names.

Examples

```
# demo("filter-nodes", package = "sigmajs")
```

<code>sg_from_gexf</code>	<i>Graph from GEXF file</i>
---------------------------	-----------------------------

Description

Create a sigmajs graph from a GEXF file.

Usage

```
sg_from_gexf(sg, file, sd = NULL)
```

Arguments

<code>sg</code>	An object of class <code>sigmajs</code> as initiated by <code>sigmajs</code> .
<code>file</code>	Path to GEXF file.
<code>sd</code>	A <code>SharedData</code> of nodes.

Examples

```
## Not run:
gexf <- "https://gephi.org/gexf/data/yeast.gexf"

sigmajs() %>%
  sg_from_gexf(gexf)

## End(Not run)
```

sg_from_igraph	<i>Create from igraph</i>
----------------	---------------------------

Description

Create a `sigmaj`s from an `igraph` object.

Usage

```
sg_from_igraph(sg, igraph, layout = NULL, sd = NULL)
```

Arguments

<code>sg</code>	An object of class <code>sigmaj</code> s as initiated by <code>sigmaj</code> s.
<code>igraph</code>	An object of class <code>igraph</code> .
<code>layout</code>	A matrix of coordinates.
<code>sd</code>	A <code>SharedData</code> of nodes.

Examples

```
## Not run:  
data("lesmis_igraph")  
  
layout <- igraph::layout_with_fr(lesmis_igraph)  
  
sigmaj() %>%  
sg_from_igraph(lesmis_igraph, layout) %>%  
sg_settings(defaultNodeColor = "#000")  
  
## End(Not run)
```

sg_get_nodes_p	<i>Get nodes</i>
----------------	------------------

Description

Retrieve nodes and edges from the widget.

Usage

```
sg_get_nodes_p(proxy)  
  
sg_get_edges_p(proxy)
```

Arguments

proxy An object of class sigmajsProxy as returned by [sigmajsProxy](#).

Examples

```
library(shiny)

nodes <- sg_make_nodes()
edges <- sg_make_edges(nodes)

ui <- fluidPage(
  actionButton("start", "Trigger layout"), # add the button
  sigmajsOutput("sg"),
  verbatimTextOutput("txt")
)

server <- function(input, output){

  output$sg <- renderSigmajs({
    sigmajs() %>%
      sg_nodes(nodes, id, size, color) %>%
      sg_edges(edges, id, source, target)
  })

  observeEvent(input$start, {
    sigmajsProxy("sg") %>% # use sigmajsProxy!
      sg_get_nodes_p()
  })

  output$txt <- renderPrint({
    input$sg_nodes
  })

}
if(interactive()) shinyApp(ui, server) # run
```

sg_layout

Layouts

Description

Layout your graph.

Usage

```
sg_layout(sg, directed = TRUE, layout = igraph::layout_nicely,
  save_igraph = TRUE, ...)
```

```
sg_get_layout(nodes, edges, directed = TRUE,
  layout = igraph::layout_nicely, save_igraph = TRUE, ...)
```

Arguments

sg	An object of class <code>sigmajsas</code> initiated by <code>sigmajs</code> .
directed	Whether or not to create a directed graph, passed to <code>graph_from_data_frame</code> .
layout	An igraph layout function.
save_igraph	Whether to save the igraph object used internally.
...	Any other parameter to pass to layout function.
nodes, edges	Nodes and edges as prepared for <code>sigmajs</code> .

Details

The package uses `igraph` internally for a lot of computations the `save_igraph` allows saving the object to speed up subsequent computations.

Value

`sg_get_layout` returns nodes with x and y coordinates.

Functions

- `sg_layout` layout your graph.
- `sg_get_layout` helper to get graph's x and y positions.

Examples

```
nodes <- sg_make_nodes(250) # 250 nodes
edges <- sg_make_edges(nodes, n = 500)

sigmajs() %>%
  sg_nodes(nodes, id, size, color) %>%
  sg_edges(edges, id, source, target) %>%
  sg_layout()

nodes_coords <- sg_get_layout(nodes, edges)
```

sg_make_nodes	<i>Generate data</i>
---------------	----------------------

Description

Generate nodes and edges.

Usage

```
sg_make_nodes(n = 10, colors = c("#B1E2A3", "#98D3A5", "#328983",  
  "#1C5C70", "#24C96B"))
```

```
sg_make_edges(nodes, n = nrow(nodes) * 1.5)
```

```
sg_make_nodes_edges(n, ...)
```

Arguments

n	Number of nodes.
colors	Color palette to use.
nodes	Nodes, as generated by <code>sg_make_nodes</code> .
...	Any other argument to pass to sample_pa .

Value

tibble of nodes or edges or a list of the latter.

Functions

- `sg_make_nodes` generate data.frame nodes.
- `sg_make_edges` generate data.frame edges.
- `sg_make_nodes_edges` generate list of nodes and edges.

Examples

```
nodes <- sg_make_nodes()  
edges <- sg_make_edges(nodes)  
  
sigmajs() %>%  
  sg_nodes(nodes, id, label, size, color) %>%  
  sg_edges(edges, id, source, target) %>%  
  sg_settings(defaultNodeColor = "#0011ff")
```

sg_neighbours	<i>Highlight neighbours</i>
---------------	-----------------------------

Description

Highlight node neighbours on click.

Usage

```
sg_neighbours(sg, nodes = "#eee", edges = "#eee")
sg_neighbours(sg, nodes = "#eee", edges = "#eee")
sg_neighbours_p(proxy, nodes = "#eee", edges = "#eee")
sg_neighbours_p(proxy, nodes = "#eee", edges = "#eee")
```

Arguments

sg	An object of class <code>sigmajs</code> as initiated by sigmajs .
nodes, edges	Color of nodes and edges
proxy	An object of class <code>sigmajsProxy</code> as returned by sigmajsProxy .

Examples

```
nodes <- sg_make_nodes()
edges <- sg_make_edges(nodes, 20)

sigmajs() %>%
  sg_nodes(nodes, id, size, color) %>%
  sg_edges(edges, id, source, target) %>%
  sg_layout() %>%
  sg_neighbours()
```

sg_nodes	<i>Add nodes and edges</i>
----------	----------------------------

Description

Add nodes and edges to a `sigmajs` graph.

Usage

```
sg_nodes(sg, data, ...)
```

```
sg_edges(sg, data, ...)
```

```
sg_edges2(sg, data)
```

```
sg_nodes2(sg, data)
```

Arguments

sg	An object of class <code>sigmajs</code> as initiated by <code>sigmajs</code> .
data	Data.frame (or list) of nodes or edges.
...	Any column name, see details.

Details

nodes: Must pass `id` (*unique*), `size` and `color`. If `color` is omitted than specify `defaultNodeColor` in `sg_settings` otherwise nodes will be transparent. Ideally nodes also include `x` and `y`, if they are not passed then they are randomly generated, you can either get these coordinates with `sg_get_layout` or `sg_layout`.

edges: Each edge also must include a unique `id` as well as two columns named `source` and `target` which correspond to node `ids`. If an edge goes from or to an `id` that is not in node `id`.

Functions

- Functions ending in 2 take a list like the original `sigma.js` JSON.
- Other functions take the arguments described above.

Note

node also takes a `SharedData`.

Examples

```
nodes <- sg_make_nodes()
edges <- sg_make_edges(nodes)

sg <- sigmajs() %>%
  sg_nodes(nodes, id, label, size, color) %>%
  sg_edges(edges, id, source, target)

sg # no layout

# layout
sg %>%
  sg_layout()

# directed graph
```

```

edges$type <- "arrow" # directed

# omit color
sigmajs() %>%
  sg_nodes(nodes, id, label, size) %>%
  sg_edges(edges, id, source, target, type) %>%
  sg_settings(defaultNodeColor = "#141414")

# all source and target are present in node ids
all(c(edges$source, edges$target) %in% nodes$id)

```

sg_noverlap

No overlap

Description

This plugin runs an algorithm which distributes nodes in the network, ensuring that they do not overlap and providing a margin where specified.

Usage

```

sg_noverlap(sg, ...)

sg_noverlap_p(proxy, nodeMargin = 5, ...)

```

Arguments

sg	An object of class <code>sigmajs</code> as initiated by <code>sigmajs</code> .
...	any option to pass to the plugin, see official documentation .
proxy	An object of class <code>sigmajsProxy</code> as returned by <code>sigmajsProxy</code> .
nodeMargin	The additional minimum space to apply around each and every node.

Examples

```

nodes <- sg_make_nodes(500)
edges <- sg_make_edges(nodes)

sigmajs() %>%
  sg_nodes(nodes, id, size, color) %>%
  sg_edges(edges, id, source, target) %>%
  sg_layout() %>%
  sg_noverlap()

```

sg_progress	<i>Text</i>
-------------	-------------

Description

Add text to your graph.

Usage

```
sg_progress(sg, data, delay, text, ..., position = "top", id = NULL,
           tag = htmltools::span, cumsum = TRUE)
```

Arguments

sg	An object of class <code>sigmajs</code> as intatiated by <code>sigmajs</code> .
data	Data.frame holding delay and text.
delay	Delay, in milliseconds at which text should appear.
text	Text to appear on graph.
...	Content of the button, compliant with <code>htmltools</code> .
position	Position of button, top or bottom.
id	A valid CSS id.
tag	A Valid <code>tags</code> function.
cumsum	Whether to compute the cumulative sum on the delay.

Details

The element is passed to `Document.createElement()` and therefore takes any valid `tagName`, including, but not limited to; `p`, `h1`, `div`.

Examples

```
# initial nodes
nodes <- sg_make_nodes()

# additional nodes
nodes2 <- sg_make_nodes()
nodes2$id <- as.character(seq(11, 20))

# add delay
nodes2$delay <- runif(nrow(nodes2), 500, 1000)
nodes2$text <- seq.Date(Sys.Date(), Sys.Date() + 9, "days")

sigmajs() %>%
  sg_nodes(nodes, id, label, size, color) %>%
  sg_add_nodes(nodes2, delay, id, label, size, color) %>%
  sg_progress(nodes2, delay, text, element = "h3") %>%
  sg_button(c("add_nodes", "progress"), "add")
```

sg_refresh_p	<i>Refresh instance</i>
--------------	-------------------------

Description

Refresh your instance.

Usage

```
sg_refresh_p(proxy)
```

Arguments

proxy An object of class `sigmajsProxy` as returned by `sigmajsProxy`.

Details

It is often required to refresh the instance when using proxies.

sg_relative_size	<i>Relative node sizes</i>
------------------	----------------------------

Description

Change nodes size depending to their degree (number of relationships)

Usage

```
sg_relative_size(sg, initial = 1)
```

Arguments

sg An object of class `sigmajs` as intatiated by `sigmajs`.
initial Initial node size.

Examples

```
nodes <- sg_make_nodes(50)
edges <- sg_make_edges(nodes, 100)

sigmajs() %>%
  sg_nodes(nodes, id, label) %>% # no need to pass size
  sg_edges(edges, id, source, target) %>%
  sg_relative_size()
```

 sg_settings

Settings

Description

Graph settings.

Usage

```
sg_settings(sg, ...)
```

Arguments

sg An object of class `sigmajsas` intatiated by `sigmajs`.
 ... Any parameter, see [official documentation](#).

Examples

```
nodes <- sg_make_nodes()

edges <- sg_make_edges(nodes, 50)

sigmajs() %>%
  sg_nodes(nodes, id, label, size) %>%
  sg_edges(edges, id, source, target) %>%
  sg_force() %>%
  sg_settings(
    defaultNodeColor = "#0011ff"
  )
```

 sg_zoom_p

Zoom

Description

Dynamically Zoom a node.

Usage

```
sg_zoom_p(proxy, id, ratio = 0.5, duration = 1000)
```

Arguments

proxy An object of class `sigmajsProxy` as returned by `sigmajsProxy`.
 id Node id to zoom to.
 ratio The zoom ratio of the graph and its items.
 duration Duration of animation.

sigmajs	<i>Initialise</i>
---------	-------------------

Description

Initialise a graph.

Usage

```
sigmajs(type = "canvas", width = "100%", kill = FALSE,  
        height = NULL, elementId = NULL)
```

Arguments

type	Renderer type, one of canvas, webgl or svg.
width, height	Dimensions of graph.
kill	Whether to kill the graph, set to FALSE if using sigmajsProxy , else set to TRUE. Only useful in Shiny.
elementId	Id of element.

Note

Keep width at 100% for a responsive visualisation.

See Also

[sg_kill](#).

Examples

```
nodes <- sg_make_nodes()  
edges <- sg_make_edges(nodes)  
  
sigmajs("svg") %>%  
  sg_nodes(nodes, id, label, size, color) %>%  
  sg_edges(edges, id, source, target)
```

`sigmajs-shiny`*Shiny bindings for sigmajs*

Description

Output and render functions for using sigmajs within Shiny applications and interactive Rmd documents.

Usage

```
sigmajsOutput(outputId, width = "100%", height = "400px")
```

```
renderSigmajs(expr, env = parent.frame(), quoted = FALSE)
```

```
sigmajsProxy(id, session = shiny::getDefaultReactiveDomain())
```

Arguments

<code>outputId, id</code>	output variable to read from
<code>width, height</code>	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
<code>expr</code>	An expression that generates a sigmajs
<code>env</code>	The environment in which to evaluate <code>expr</code> .
<code>quoted</code>	Is <code>expr</code> a quoted expression (with <code>quote()</code>)? This is useful if you want to save an expression in a variable.
<code>session</code>	A valid shiny session.

Index

*Topic **datasets**

- lesmis_edges, 5
 - lesmis_igraph, 5
 - lesmis_nodes, 6
- color-scale, 2
- force, 3, 10
- graph_from_data_frame, 20, 31
- lesmis_edges, 5
- lesmis_igraph, 5
- lesmis_nodes, 6
- read, 6
- read-batch, 8
- read-static, 9
- renderSigmaajs (sigmaajs-shiny), 40
- sample_pa, 32
- sg_add_edge_p (sg_add_node_p), 15
- sg_add_edges (sg_add_nodes), 11
- sg_add_edges_delay_p
(sg_add_nodes_delay_p), 13
- sg_add_edges_p (sg_add_nodes_p), 14
- sg_add_images, 11
- sg_add_node_p, 15
- sg_add_nodes, 11
- sg_add_nodes_delay_p, 13
- sg_add_nodes_p, 14
- sg_animate, 15
- sg_button, 16
- sg_change_edges_p (sg_change_nodes_p),
18
- sg_change_nodes_p, 18
- sg_clear (sg_clear_p), 19
- sg_clear_p, 19
- sg_cluster, 20
- sg_custom_shapes, 21
- sg_drag_nodes, 22
- sg_drag_nodes_kill_p (sg_drag_nodes), 22
- sg_drag_nodes_start_p (sg_drag_nodes),
22
- sg_drop_edge_p (sg_drop_node_p), 25
- sg_drop_edges (sg_drop_nodes), 22
- sg_drop_edges_delay_p
(sg_drop_nodes_delay_p), 23
- sg_drop_edges_p (sg_drop_nodes_p), 25
- sg_drop_node_p, 25
- sg_drop_nodes, 22
- sg_drop_nodes_delay_p, 23
- sg_drop_nodes_p, 25
- sg_edges (sg_nodes), 33
- sg_edges2 (sg_nodes), 33
- sg_export_img (sg_export_svg), 26
- sg_export_img_p (sg_export_svg), 26
- sg_export_svg, 26
- sg_export_svg_p (sg_export_svg), 26
- sg_filter_eq_p (sg_filter_gt_p), 27
- sg_filter_gt_p, 27
- sg_filter_lt_p (sg_filter_gt_p), 27
- sg_filter_neighbours_p
(sg_filter_gt_p), 27
- sg_filter_not_eq_p (sg_filter_gt_p), 27
- sg_filter_undo_p (sg_filter_gt_p), 27
- sg_force (force), 3
- sg_force_config_p (force), 3
- sg_force_kill_p (force), 3
- sg_force_restart (force), 3
- sg_force_restart_p (force), 3
- sg_force_start (force), 3
- sg_force_start_p (force), 3
- sg_force_stop (force), 3
- sg_force_stop_p (force), 3
- sg_from_gexf, 28
- sg_from_igraph, 29
- sg_get_cluster (sg_cluster), 20
- sg_get_edges_p (sg_get_nodes_p), 29
- sg_get_layout, 34

`sg_get_layout` (`sg_layout`), 30
`sg_get_nodes_p`, 29
`sg_kill`, 39
`sg_kill` (`sg_clear_p`), 19
`sg_kill_p` (`sg_clear_p`), 19
`sg_layout`, 30, 34
`sg_make_edges` (`sg_make_nodes`), 32
`sg_make_nodes`, 32
`sg_make_nodes_edges` (`sg_make_nodes`), 32
`sg_neighbors` (`sg_neighbours`), 33
`sg_neighbors_p` (`sg_neighbours`), 33
`sg_neighbours`, 33
`sg_neighbours_p` (`sg_neighbours`), 33
`sg_nodes`, 33
`sg_nodes2` (`sg_nodes`), 33
`sg_noverlap`, 35
`sg_noverlap_p` (`sg_noverlap`), 35
`sg_progress`, 36
`sg_read_delay_edges_p` (`read-batch`), 8
`sg_read_delay_exec_p` (`read-batch`), 8
`sg_read_delay_nodes_p` (`read-batch`), 8
`sg_read_edges` (`read-static`), 9
`sg_read_edges_p` (`read`), 6
`sg_read_exec` (`read-static`), 9
`sg_read_exec_p` (`read`), 6
`sg_read_nodes` (`read-static`), 9
`sg_read_nodes_p` (`read`), 6
`sg_refresh_p`, 37
`sg_relative_size`, 37
`sg_scale_color` (`color-scale`), 2
`sg_settings`, 34, 38
`sg_zoom_p`, 38
`SharedData`, 28, 29, 34
`sigmajs`, 3, 4, 10–12, 16, 17, 20–23, 26, 28, 29, 31, 33–38, 39
`sigmajs-shiny`, 40
`sigmajsOutput` (`sigmajs-shiny`), 40
`sigmajsProxy`, 4, 6, 8, 13–15, 18–20, 22, 24–27, 30, 33, 35, 37–39
`sigmajsProxy` (`sigmajs-shiny`), 40
`tags`, 17, 36