Package 'sigmajs'

June 18, 2020

Title Interface to 'Sigma.js' Graph Visualization Library

Date 2020-06-17

Version 0.1.5

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

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0.9000

Depends R (>= 2.10)

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

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

Imports htmlwidgets, dplyr (>= 0.7.0), magrittr, shiny, jsonlite, igraph, htmltools, purrr, scales, crosstalk

Suggests testthat

NeedsCompilation no

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

Maintainer John Coene <jcoenep@gmail.com>

Repository CRAN

Date/Publication 2020-06-18 18:10:02 UTC

R topics documented:

`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.

Value

A modified version of the sg object.

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 *Add forceAtlas2*

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 sigmajsas intatiated by sigmajs.
...	Any parameter, see official documentation.
delay	Milliseconds after which the layout algorithm should stop running.
proxy	An object of class sigmajsProxy as returned by sigmajsProxy.
refresh	Whether to refresh the graph after node is dropped, required to take effect.
data	data.frame 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.

Value

Their first arguments, either sg or proxy.

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 ompletely stops the layout and terminates the assiociated 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 *Edges from co-appearances of characters in "Les Miserables"*

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 *Co-appearances of characters in "Les Miserables" as igraph object*

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 *Nodes from co-appearances of characters in "Les Miserables"*

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 *Read*

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.

Value

The proxy object.

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 sigmajsProxy as returned by sigmajsProxy.
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.

Value

The proxy object.

Examples

```
library(shiny)

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

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

output$sg <- renderSigmajs({
sigmajs()
})

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

`sg`	An object of class `sigmajs`as intatiated by `sigmajs`.
`data`	Data.frame (or list) of nodes or edges.
`...`	Any column name, see details.
`delay`	Column name containing delay in milliseconds.
`refresh`	Whether to refresh the `force` layout.

Value

A modified version of the `sg` object.

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 *Add images to nodes*

Description

Add images to nodes with the Custom Shapes plugin.

Usage

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

Arguments

sg	An object of class sigmajsas intatiated by sigmajs.
data	Data.frame 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 *Add nodes and edges*

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 sigmajsas intatiated by sigmajs.
data	Data.frame (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.

Value

A modified version of the `sg` object.

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 sigmajsProxy as returned by sigmajsProxy.
data	A data.frame of _one_ 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.

Value

The proxy object.

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_add_nodes_p *Add nodes or edges*

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 sigmajsProxy as returned by sigmajsProxy.
data	A data.frame 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 data.frame of nodes is added or at each iteration (row-wise). Only applies if refresh is set to TRUE.

Value

The proxy object.

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 *Add node or edge*

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 sigmajsProxy as returned by sigmajsProxy.
data	A data.frame of _one_ 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.

Value

The `proxy` object.

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` *Animate*

Description

Animate graph components.

Usage

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

Arguments

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

Value

An object of class `htmlwidget` which renders the visualisation on print.

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 sigmajsas intatiated by sigmajs.
event	Event the button triggers, see valid events.
...	Content of the button, complient with htmltools.
position	Position of button, top or bottom.
class	Button CSS class, see note.
tag	A Valid htmltools tags function.
id	A valid CSS id.

Details

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

Value

An object of class `htmlwidget` which renders the visualisation on print.

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)

sigmajs() %>%
  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 sigmajsProxy as returned by sigmajsProxy.
data	data.frame holding delay column.
value	Column containing value.
attribute	Name of attribute to change.
rate	Rate at chich 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 sigmajsProxy 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 sigmajsas intatiated by sigmajs.

Value

The proxy object.

A modified version of the sg object.

---

sg_cluster *Cluster*

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 sigmajsas 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 algo.
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 while sg_cluster returns an object of class htmlwidget which renders the visualisation on print.

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)
```

---

sg_custom_shapes *Custom shapes*

Description

Indicate a graph uses custom shapes

Usage

```
sg_custom_shapes(sg)
```

Arguments

sg	An object of class sigmajsas intatiated by sigmajs.

---

sg_drag_nodes *Drag nodes*

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 sigmajsas intatiated by sigmajs.
proxy	An object of class sigmajsProxy as returned by sigmajsProxy.

Value

sg_drag_nodes An object of class htmlwidget which renders the visualisation on print. While sg_drag_nodes_start_p and sg_drag_nodes_kill_p

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 *Drop*

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 sigmajsas intatiated by sigmajs.
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.

Value

A modified version of the sg object.

Examples

```
nodes <- sg_make_nodes(75)

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

sigmajs() %>%
  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 sigmajsProxy as returned by sigmajsProxy.
data	A data.frame of _one_ 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.

Value

The proxy object.

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_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 sigmajsProxy as returned by sigmajsProxy.
data	A data.frame 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 data.frame of nodes is added or at each iteration (row-wise). Only applies if refresh is set to TRUE.

Value

The proxy object.

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 sigmajsProxy 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.

Value

The proxy object.

---

sg_events *Events*

Description

Get events server-side.

Usage

```
sg_events(sg, events)
```

Arguments

sg	An object of class sigmajsas intatiated by sigmajs.
events	A vector of valid events (see section below).

Details

Events: Valid events to pass to events.

- clickNode
- clickNodes
- clickEdge
- clickEdges
- clickStage
- doubleClickStage
- rightClickStage
- doubleClickNode
- doubleClickNodes
- doubleClickEdge
- doubleClickEdges
- rightClickNode
- rightClickNodes

- rightClickEdge
- rightClickEdges
- hoverNode
- hoverNodes
- hoverEdge
- hoverEdges
- outNode
- outNodes
- outEdge
- outEdges

Value

An object of class `htmlwidget` which renders the visualisation on print.

See Also

official documentation.

Examples

```
library(shiny)

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

ui <- fluidPage(
  sigmajsOutput("sg"),
  p("Click on a node"),
  verbatimTextOutput("clicked")
)

server <- function(input, output){
  output$sg <- renderSigmajs({
    sigmajs() %>%
      sg_nodes(nodes, id, size, color) %>%
      sg_edges(edges, id, source, target) %>%
      sg_events("clickNode")
  })

# capture node clicked
output$clicked <- renderPrint({
    input$sg_click_node
  })
}

## Not run: shinyApp(ui, server)
```

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 sigmajsas intatiated by sigmajs.
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 sigmajsProxy as returned by sigmajsProxy.

Value

An object of class htmlwidget which renders the visualisation on print. Functions ending in _p return the proxy.

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")
```

---

sg_filter_gt_p *Filter*

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 sigmajsProxy as returned by sigmajsProxy.
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 sg_filter_undo.
node	Node id to filter neighbours.

Value

The proxy object.

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.

---

sg_from_gexf *Graph from GEXF file*

Description

Create a sigmajs graph from a GEXF file.

Usage

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

Arguments

sg	An object of class sigmajsas intatiated by sigmajs.
file	Path to GEXF file.
sd	A SharedData of nodes.

Value

A modified version of the sg object.

Examples

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

sigmajs() %>%
  sg_from_gexf(gexf)

## End(Not run)
```

sg_from_igraph *Create from igraph*

Description

Create a sigmajs from an igraph object.

Usage

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

Arguments

sg	An object of class sigmajsas intatiated by sigmajs.
igraph	An object of class igraph.
layout	A matrix of coordinates.
sd	A SharedData of nodes.

Value

A modified version of the sg object.

Examples

```
## Not run:
data("lesmis_igraph")

layout <- igraph::layout_with_fr(lesmis_igraph)

sigmajs() %>%
sg_from_igraph(lesmis_igraph, layout) %>%
sg_settings(defaultNodeColor = "#000")

## End(Not run)
```

sg_get_nodes_p *Get nodes*

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.

Value

The proxy object.

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 sigmajsas intatiated by sigmajs.
directed	Whether or not to create a directed graph, passed to graph_from_data_frame.
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 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_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` *Generate data*

Description

Generate nodes and edges.

Usage

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

sg_make_edges(nodes, n = NULL)

sg_make_nodes_edges(n, ...)
```

Arguments

`n`	Number of nodes.
`colors`	Color palette to use.
`nodes`	Nodes, as generated by `sg_make_nodes`.
`...`	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 *Highlight neighbours*

Description

Highlight node neighbours on click.

Usage

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

sg_neighbors(sg, nodes = "#eee", edges = "#eee")

sg_neighbours_p(proxy, nodes = "#eee", edges = "#eee")

sg_neighbors_p(proxy, nodes = "#eee", edges = "#eee")
```

Arguments

sg	An object of class sigmajsas intatiated by sigmajs.
nodes, edges	Color of nodes and edges
proxy	An object of class sigmajsProxy as returned by sigmajsProxy.

Value

A modified version of the sg object.

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 *Add nodes and edges*

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 `sigmajs`as intatiated by `sigmajs`.
`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 `id`s. If an edges goes from or to an `id` that is not in node `id`.

Value

A modified version of the `sg` object.

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

Value

The first argument either sg or proxy.

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 *Text*

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 sigmajsas intatiated by sigmajs.
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, complient with htmltools.
position	Position of button, top or bottom.
id	A valid CSS id.
tag	A Valid htmltools tags 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.

Value

A modified version of the sg object.

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 *Refresh instance*

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 *Relative node sizes*

Description

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

Usage

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

Arguments

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

Value

A modified version of the sg object.

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, ...)

sg_settings_p(proxy, ...)
```

Arguments

sg	An object of class sigmajsas intatiated by sigmajs.
...	Any parameter, see official documentation.
proxy	A proxy as returned by sigmajsProxy.

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 *Initialise*

Description

Initialise a graph.

Usage

```
sigmajs(
  type = NULL,
  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 elment.

Value

An object of class htmlwidget which renders the visualisation on print.

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

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

Index