Package 'DiagrammeR'

October 13, 2015

Type Package

Title Create Graph Diagrams and Flowcharts Using R

Version 0.8.1

Date 2015-10-11

Maintainer Richard Iannone <riannone@me.com>

Description Create graph diagrams and flowcharts using R.

License MIT + file LICENSE

Imports htmlwidgets (>= 0.5), rstudioapi (>= 0.3.1), stringr (>= 1.0.0), visNetwork (>= 0.1.1)

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

Suggests knitr, testthat

NeedsCompilation no

Author Knut Sveidqvist [aut, cph] (mermaid.js library in htmlwidgets/lib, http://github.com/knsv/mermaid/),
Mike Bostock [aut, cph] (d3.js library in htmlwidgets/lib, http://d3js.org),
Chris Pettitt [aut, cph] (dagre-d3.js library in htmlwidgets/lib, http://github.com/cpettitt/dagre-d3),
Mike Daines [aut, cph] (viz.js library in htmlwidgets/lib, http://github.com/mdaines/viz.js/),
Andrei Kashcha [aut, cph] (vivagraphjs library in htmlwidgets/lib, https://github.com/anvaka/VivaGraphJS),
Richard Iannone [aut, cre] (R interface)

Repository CRAN

Date/Publication 2015-10-13 09:10:44

R topics documented:

---

add_edges *Add edges to an existing graph object*

Description

With a graph object of class dgr_graph, add one or more edges of specified types to nodes within the graph.

Usage

```
add_edges(graph, edges_df = NULL, from = NULL, to = NULL, rel = NULL)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
edges_df	an edge data frame that is created using create_edges.
from	a vector of the outgoing nodes from which each edge is connected.
to	a vector of the incoming nodes to which each edge is connected.
rel	a string specifying the relationship between the connected nodes.

Value

a graph object of class dgr_graph.

Examples

```
## Not run:
# Create a graph with two nodes
graph <- create_graph(create_nodes(nodes = c("a", "b")))

# Add an edge between those nodes and attach a relationship to the edge
graph <- add_edges(graph, from = "a", to = "b",
                   rel = "to_get")

# Examples of pipeable graph building using 'create_edges' with
# 'add_edges' in order to include values for the 'style' edge attribute
# (it modifies the style of the connecting line)

library(magrittr)

graph <- create_graph() %>%
  add_node("a") %>%
  add_node("b") %>%
  add_edges(create_edges(from = "a", to = "b",
                         style = "solid")) %>%
  add_node("c") %>%
  add_node("d") %>%
```

```
  add_edges(create_edges(from = "c", to = "d",
                         style = "dashed")) %>%
  add_node("e") %>%
  add_node("f") %>%
  add_edges(create_edges(from = "e", to = "f",
                         style = "dotted")) %>%
  add_node("g") %>%
  add_node("h") %>%
  add_edges(create_edges(from = "g", to = "h",
                         style = "bold"))

## End(Not run)
```

---

add_node *Add a node to an existing graph object*

Description

With a graph object of class dgr_graph, add a new node of a specified type to extant nodes within the graph.

Usage

```
add_node(graph, node, from = NULL, to = NULL, label = TRUE, type = NULL,
  ...)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
node	a node ID for the newly connected node.
from	an optional vector containing node IDs from which edges will be directed to the new node.
to	an optional vector containing node IDs to which edges will be directed from the new node.
label	a character object for supplying an optional label to the node. Setting to TRUE ascribes the node ID to the label. Setting to FALSE yields a blank label.
type	an optional string that describes the entity type for the node to be added.
...	one or more vectors pertaining to node attributes.

Value

a graph object of class dgr_graph.

Examples

```
## Not run:
# Create an empty graph
graph <- create_graph()

# Add two nodes
graph <- add_node(graph, node = "a")
graph <- add_node(graph, node = "b")

## End(Not run)
```

`add_to_series` *Add graph object to a graph series object*

Description

Add a graph object to an extant graph series object for storage of multiple graphs across a sequential or temporal one-dimensional array.

Usage

```
add_to_series(graph, graph_series)
```

Arguments

`graph`	a graph object to add to the graph series object.
`graph_series`	a graph series object to which the graph object will be added.

Value

a graph series object of type `dgr_graph_1D`.

Examples

```
## Not run:
# Create three graphs (using \code{pipeR} for speed)
# and create a graph series using those graphs
library(magrittr)

graph_1 <- create_graph() %>%
  add_node("a") %>% add_node("b") %>% add_node("c") %>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_2 <- graph_1 %>%
  add_node("d") %>% add_edges(from = "d", to = "c")

graph_3 <- graph_2 %>%
  add_node("e") %>% add_edges(from = "e", to = "b")
```

```
# Create an empty graph series
series <- create_series(series_type = "sequential")

# Add graphs to the graph series
series <- graph_1 %>% add_to_series(series)
series <- graph_2 %>% add_to_series(series)
series <- graph_3 %>% add_to_series(series)

## End(Not run)
```

`combine_edges` *Combine multiple edge data frames into a single edge data frame*

Description

Combine several edge data frames in the style of `rbind`, except, it works regardless of the number and ordering of the columns.

Usage

```
combine_edges(...)
```

Arguments

`...`	two or more edge data frames, which contain edge IDs and associated attributes.

Value

a combined edge data frame.

Examples

```
## Not run:
# Combine two edge data frames
edges_1 <-
  create_edges(from = c("a", "a", "b", "c"),
               to = c("b", "d", "d", "a"),
               rel = "requires",
               color = "green",
               data = c(2.7, 8.9, 2.6, 0.6))

edges_2 <-
  create_edges(from = c("e", "g", "h", "h"),
               to = c("g", "h", "f", "e"),
               rel = "receives",
               arrowhead = "dot",
               color = "red")

all_edges <- combine_edges(edges_1, edges_2)
```

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

`combine_graphs` *Combine two graphs into a single graph*

Description

Combine two graphs in order to make a new graph, merging nodes and edges in the process. The use of an optional data frame allows for new edges to be formed across the combined graphs.

Usage

```
combine_graphs(x, y, edges_df = NULL)
```

Arguments

`x`	a `DiagrammeR` graph object to which another graph will be joined. This graph should be considered the host graph as the resulting graph will retain only the attributes of this graph.
`y`	a `DiagrammeR` graph object that is to be joined with the graph suppled as `x`.
`edges_df`	an optional edge data frame that allows for connections between nodes across the graphs to be combined.

Value

a graph object of class `dgr_graph`.

`combine_nodes` *Combine multiple node data frames into a single node data frame*

Description

Combine several node data frames in the style of `rbind`, except, it works regardless of the number and ordering of the columns.

Usage

```
combine_nodes(...)
```

Arguments

`...`	two or more node data frames, which contain node IDs and associated attributes.

Value

a combined node data frame.

Examples

```
## Not run:
# Combine two node data frames
nodes_1 <-
  create_nodes(nodes = c("a", "b", "c", "d"),
               label = FALSE,
               type = "lower",
               style = "filled",
               color = "aqua",
               shape = c("circle", "circle",
                         "rectangle", "rectangle"),
               data = c(3.5, 2.6, 9.4, 2.7))

nodes_2 <-
  create_nodes(nodes = c("e", "f", "g", "h"),
               label = FALSE,
               type = "upper",
               style = "filled",
               color = "red",
               shape = "triangle",
               data = c(0.5, 3.9, 3.7, 8.2))

all_nodes <- combine_nodes(nodes_1, nodes_2)

## End(Not run)
```

---

country_graph *Create a graph object that contains the boundaries of a country*

Description

Create a graph object that contains the boundaries of a country.

Usage

```
country_graph(iso_a2 = NULL, scaling_factor = 40)
```

Arguments

iso_a2	the ISO 2-letter identifier for a country.
scaling_factor	the factor for which degree coordinates for country boundries will be multiplied.

Value

a graph object of class dgr_graph.

create_edges *Create a data frame with edges and their attributes*

Description

Combine several named vectors for edges and their attributes into a data frame, which can be combined with other similarly-generated data frame, or, added to a graph object.

Usage

```
create_edges(...)
```

Arguments

...	one or more named vectors for edges and associated attributes; the names for the named vectors must include `from` and `to` alongside any named vectors for the edge attributes and ancillary data.

Value

a data frame.

Examples

```
## Not run:
# Create an edge data frame
edges <-
  create_edges(from = c("a", "b", "c"),
               to = c("d", "c", "a"),
               rel = "leading_to")

## End(Not run)
```

create_graph *Create a graph object using data frames representative of nodes and edges*

Description

Generates a graph object using data frames for nodes and/or edges; the graph object can be manipulated by other functions.

Usage

```
create_graph(nodes_df = NULL, edges_df = NULL, graph_attrs = NULL,
  node_attrs = NULL, edge_attrs = NULL, directed = TRUE,
  graph_name = NULL, graph_time = NULL, graph_tz = NULL,
  generate_dot = TRUE)
```

Arguments

nodes_df	an optional data frame containing, at minimum, a column (called nodes) which contains node IDs for the graph. Additional columns (named as Graphviz node attributes) can be included with values for the named node attribute.
edges_df	an optional data frame containing, at minimum, two columns (called from and to) where node IDs are provided. Additional columns (named as Graphviz edge attributes) can be included with values for the named edge attribute.
graph_attrs	an optional vector of graph attribute statements that can serve as defaults for the graph.
node_attrs	an optional vector of node attribute statements that can serve as defaults for nodes.
edge_attrs	an optional vector of edge attribute statements that can serve as defaults for edges.
directed	with TRUE (the default) or FALSE, either directed or undirected edge operations will be generated, respectively.
graph_name	an optional string for labeling the graph object.
graph_time	a date or date-time string (required for insertion of graph into a graph series of the type temporal).
graph_tz	an optional value for the time zone (tz) corresponding to the date or date-time string supplied as a value to graph_time. If no time zone is provided then it will be set to GMT.
generate_dot	an option to generate Graphviz DOT code and place into the graph object.

Value

a graph object of class dgr_graph.

Examples

```
## Not run:
# Create an empty graph
graph <- create_graph()

# Create a graph with nodes but no edges
nodes <- create_nodes(nodes = c("a", "b", "c", "d"))

graph <- create_graph(nodes_df = nodes)

# Create a graph with nodes with values, types, labels
nodes <- create_nodes(nodes = c("a", "b", "c", "d"),
               label = TRUE,
               type = c("type_1", "type_1",
                        "type_5", "type_2"),
               shape = c("circle", "circle",
                         "rectangle", "rectangle"),
               values = c(3.5, 2.6, 9.4, 2.7))
```

```
graph <- create_graph(nodes_df = nodes)

# Create a graph from an edge data frame, the nodes will
edges <- create_edges(from = c("a", "b", "c"),
                      to = c("d", "c", "a"),
                      rel = "leading_to")

graph <- create_graph(edges_df = edges)

# Create a graph with both nodes and nodes defined, and,
# add some default attributes for nodes and edges
graph <- create_graph(nodes_df = nodes,
                      edges_df = edges,
                      node_attrs = "fontname = Helvetica",
                      edge_attrs = c("color = blue",
                                     "arrowsize = 2"))

## End(Not run)
```

`create_nodes` *Create a data frame with nodes and their attributes*

Description

Combine several named vectors for nodes and their attributes into a data frame, which can be combined with other similarly-generated data frame, or, added to a graph object.

Usage

```
create_nodes(...)
```

Arguments

...	one or more named vectors for nodes and associated attributes.

Value

a data frame.

Examples

```
## Not run:
# Create a node data frame
nodes <-
  create_nodes(nodes = c("a", "b", "c", "d"),
               label = TRUE,
               type = c("lower", "lower", "upper", "upper"),
               style = "filled",
               color = "aqua",
               shape = c("circle", "circle",
```

```
                    "rectangle", "rectangle"),
                data = c(3.5, 2.6, 9.4, 2.7))

## End(Not run)
```

`create_random_graph` *Create a randomized graph*

Description

Create a graph of up to n nodes with randomized edge assignments.

Usage

```
create_random_graph(n, m)
```

Arguments

n	the number of nodes to use in the random graph.
m	the number of edges to use in the random graph.

Examples

```
## Not run:
# Create a random graph with 50 nodes
graph_random <- create_random_graph(50)

## End(Not run)
```

`create_series` *Create a graph series object*

Description

Create a graph series object for storage of multiple graphs across a sequential or temporal one-dimensional array.

Usage

```
create_series(graph = NULL, series_name = NULL,
  series_type = "sequential", series_scripts = NULL)
```

Arguments

graph	a graph object to add to the new graph series object.
series_name	an optional name to ascribe to the series.
series_type	either a sequential type (the default) or a temporal type (which requires date-time strings and time zone codes to be supplied).
series_scripts	a vector of R scripts or paths to R scripts.

Value

a graph series object of type dgr_graph_1D.

Examples

```
## Not run:
# Create three graphs (using \code{magrittr} pipes)
# and create a graph series using those graphs
library(magrittr)

graph_1 <- create_graph() %>%
  add_node("a") %>% add_node("b") %>% add_node("c") %>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_2 <- graph_1 %>%
  add_node("d") %>% add_edges(from = "d", to = "c")

graph_3 <- graph_2 %>%
  add_node("e") %>% add_edges(from = "e", to = "b")

# Create an empty graph series
series <- create_series(series_type = "sequential")

# Add graphs to the graph series
series <- graph_1 %>>% add_to_series(series)
series <- graph_2 %>>% add_to_series(series)
series <- graph_3 %>>% add_to_series(series)

## End(Not run)
```

---

create_subgraph *Create a subgraph based on a walk distance from a specified node*

Description

Create a subgraph for a neighborhood of nodes connected a specified distance from the selected node.

Usage

```
create_subgraph(graph, starting_node, distance)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
starting_node	the node from which the subgraph will originate.
distance	the maximum number of steps from the starting_node for inclusion in the subgraph.

Examples

```
## Not run:
# Create a graph, then, create a subgraph of that larger graph
nodes <-
  create_nodes(nodes = LETTERS,
               type = "letter",
               shape = sample(c("circle", "rectangle"),
                              length(LETTERS),
                              replace = TRUE),
               fillcolor = sample(c("aqua", "gray80",
                                    "pink", "lightgreen",
                                    "azure", "yellow"),
                                  length(LETTERS),
                                  replace = TRUE))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               rel = "letter_to_letter")

graph <- create_graph(nodes_df = nodes,
                      edges_df = edges,
                      graph_attrs = "layout = neato",
                      node_attrs = c("fontname = Helvetica",
                                     "style = filled"),
                      edge_attrs = c("color = gray20",
                                     "arrowsize = 0.5"))

# Create a subgraph centered around node "U" and include
# those nodes up to (and including) 2 connections away
subgraph <- create_subgraph(graph = graph,
                            starting_node = "U",
                            distance = 2)

# Render the graph using Graphviz
render_graph(subgraph)

# Render the graph using VivaGraph
render_graph(subgraph, output = "vivagraph")
```

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

---

delete_edge *Delete an edge from an existing graph object*

Description

From a graph object of class dgr_graph, delete an existing edge by specifying a pair of node IDs corresponding to the edge direction.

Usage

```
delete_edge(graph, from, to)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
from	a node ID from which the edge to be removed is outgoing.
to	a node ID to which the edge to be removed is incoming.

Value

a graph object of class dgr_graph.

Examples

```
## Not run:
# Create an empty graph
graph <- create_graph()

# Add two nodes
graph <- add_node(graph, node = "a")
graph <- add_node(graph, node = "b")

# Add an edge
graph <- add_edges(graph, from = "a", to = "b")

# Delete the edge
graph <- delete_edge(graph, from = "a", to = "b")

## End(Not run)
```

delete_node *Delete a node from an existing graph object*

Description

From a graph object of class dgr_graph, delete an existing node by specifying its node ID.

Usage

```
delete_node(graph, node)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
node	a node ID for the node to be deleted from the graph.

Value

a graph object of class dgr_graph.

Examples

```
## Not run:
# Create an empty graph
graph <- create_graph()

# Add two nodes
graph <- add_node(graph, node = "a")
graph <- add_node(graph, node = "b")

# Delete a node
graph <- delete_node(graph, node = "a")

## End(Not run)
```

DiagrammeR *R + mermaid.js*

Description

Make diagrams in R using viz.js or mermaid.js with infrastructure provided by htmlwidgets.

Usage

```
DiagrammeR(diagram = "", type = "mermaid", ...)
```

Arguments

diagram	diagram in graphviz or mermaid format or a file (as a connection or file name) containing a diagram specification. The recommended filename extensions are .gv and .mmd for the Graphviz and the mermaid diagram specifications, respectively. If no diagram is provided (diagram = "") then the function will assume that a diagram will be provided by tags and DiagrammeR is just being used for dependency injection.
type	string - either mermaid (default) or grViz indicating the type of diagram spec and the desired parser/renderer
...	any other parameters to pass to grViz or mermaid

Value

An object of class htmlwidget that will intelligently print itself into HTML in a variety of contexts including the R console, within R Markdown documents, and within Shiny output bindings.

Examples

```
## Not run:
# note the whitespace is not important
DiagrammeR("
  graph LR
    A-->B
    A-->C
    C-->E
    B-->D
    C-->D
    D-->F
    E-->F
")

DiagrammeR("
   graph TB
   A-->B
   A-->C
   C-->E
   B-->D
   C-->D
   D-->F
   E-->F
")

DiagrammeR("graph LR;A(Rounded)-->B[Squared];B-->C{A Decision};
 C-->D[Square One];C-->E[Square Two];
 style A fill:#E5E25F;  style B fill:#87AB51; style C fill:#3C8937;
 style D fill:#23772C;  style E fill:#B6E6E6;"
)

# Load in the 'mtcars' dataset
data(mtcars)
connections <- sapply(
```

```
 1:ncol(mtcars)
  ,function(i){
     paste0(
        i
      ,"(",colnames(mtcars)[i],")---"
      ,i,"-stats("
      ,paste0(
        names(summary(mtcars[,i]))
        ,": "
        ,unname(summary(mtcars[,i]))
        ,collapse="<br/>"
      )
      ,")"
   )
 }
)

DiagrammeR(
   paste0(
     "graph TD;", "\n",
     paste(connections, collapse = "\n"),"\n",
     "classDef column fill:#0001CC, stroke:#0D3FF3, stroke-width:1px;" ,"\n",
     "class ", paste0(1:length(connections), collapse = ","), " column;"
   )
 )

# also with DiagrammeR() you can use tags from htmltools
# just make sure to use class = "mermaid"
library(htmltools)
diagramSpec = "
graph LR;
  id1(Start)-->id2(Stop);
  style id1 fill:#f9f,stroke:#333,stroke-width:4px;
  style id2 fill:#ccf,stroke:#f66,stroke-width:2px,stroke-dasharray: 5, 5;
"
html_print(tagList(
  tags$h1("R + mermaid.js = Something Special")
  ,tags$pre(diagramSpec)
  ,tags$div(class="mermaid",diagramSpec)
  ,DiagrammeR()
))

# sequence diagrams
# Using this "How to Draw a Sequence Diagram"
#   http://www.cs.uku.fi/research/publications/reports/A-2003-1/page91.pdf
# draw some sequence diagrams with DiagrammeR

library(DiagrammeR)

DiagrammeR("
sequenceDiagram;
   customer->>ticket seller: ask ticket;
   ticket seller->>database: seats;
```

```
   alt tickets available
     database->>ticket seller: ok;
     ticket seller->>customer: confirm;
     customer->>ticket seller: ok;
     ticket seller->>database: book a seat;
     ticket seller->>printer: print ticket;
   else sold out
     database->>ticket seller: none left;
     ticket seller->>customer:  sorry;
   end
")

## End(Not run)
```

DiagrammeROutput *Widget output function for use in Shiny*

Description

Widget output function for use in Shiny

Usage

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

Arguments

outputId	output variable to read from
width	a valid CSS unit for the width or a number, which will be coerced to a string and have px appended.
height	a valid CSS unit for the height or a number, which will be coerced to a string and have px appended.

---

display_graph_object *Update and display graph object*

Description

Using a dgr_graph object, update values of counts for nodes, edges, attributes, directed state, and display the schematic in the RStudio Viewer.

Usage

```
display_graph_object(graph, width = 400)
```

Arguments

graph	a dgr_graph object, created using the create_graph function.
width	the width of the graph representation in pixels.

Examples

```
## Not run:
Create a simple graph and display a visual summary
of simple graph properties
nodes <-
  create_nodes(nodes = c("a", "b", "c", "d"),
               label = FALSE,
               type = "letter")

edges <-
  create_edges(from = c("a", "b", "c"),
               to = c("d", "c", "a"),
               rel = "connected_to")

graph <- create_graph(nodes_df = nodes,
                      edges_df = edges)

display_graph_object(graph, width = 640)

## End(Not run)
```

---

edge_count *Get count of all edges or edges with distinct relationship types*

Description

From a graph object of class dgr_graph, get a count of edges in the graph and optionally obtain a count of edges by their relationship type.

Usage

```
edge_count(graph, rel = FALSE)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
rel	either a logical value, where TRUE provides a named vector of edge count by type and FALSE (the default) provides a total count of edges, or, a string corresponding to one or more edge relationship types.

Value

a numeric vector of single length.

Examples

```
## Not run:
# Before getting counts of edges, create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

# Get a total count of edges with no grouping
edge_count(graph, rel = FALSE)
#> [1] 26

## End(Not run)
```

---

edge_info *Get detailed information on edges*

Description

Obtain a data frame with detailed information on edges and their interrelationships within a graph.

Usage

```
edge_info(graph)
```

Arguments

graph	a graph object of class dgr_graph.

Value

a data frame containing information specific to each edge within the graph.

Examples

```
## Not run:
# Create a simple graph and get edge information from it
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

edge_info(graph)
#>    from   to              rel label
#> 1     A    Z letter_to_letter  edge
#> 2     H    U letter_to_letter  edge
#> 3     W    O letter_to_letter  edge
#> 4     U    K letter_to_letter  edge
#> 5     I    V letter_to_letter  edge
#>..   ...  ...              ...   ...

## End(Not run)
```

`edge_present` *Determine whether a specified edge is present in an existing graph object*

Description

From a graph object of class `dgr_graph`, determine whether a directed edge (defined by a pair of node IDs extant in the graph) is present.

Usage

```
edge_present(graph, from, to)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
from	a node ID from which the edge to be queried is outgoing.
to	a node ID to which the edge to be queried is incoming.

Value

a logical value.

Examples

```
## Not run:
# Before finding out whether a particular edge is present,
# create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

# Is there any edge between nodes with IDs 'A' and 'B'?
edge_present(graph, from = "A", to = "B")
#> FALSE

# Verify that there is an edge between nodes 'K' and 'V'
edge_present(graph, from = "K", to = "V")
#> TRUE

## End(Not run)
```

---

edge_rel *Create, read, update, delete, or report status of an edge relationship*

Description

From a graph object of class dgr_graph, query an edge in the graph (defined by a pair of node IDs extant in the graph) and perform operations on the relationship for that edge.

Usage

```
edge_rel(graph, from, to, action = "read", value = NULL)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
from	a node ID from which the edge to be queried is outgoing.
to	a node ID to which the edge to be queried is incoming.
action	the operation to perform on the edge's relationship attribute. To remove a relationship from an edge, use either delete, remove, or drop. To add a relationship to an edge with no set relationship, use add or create. To update an edge relationship, use update. To return the value of an edge relationship, use read. To determine whether there is a set relationship, use check.
value	a string denoting the relationship, to be supplied when either adding or updating an edge relationship.

Value

a graph object of class dgr_graph.

---

get_edges *Get node IDs associated with edges*

Description

Provides information on the node IDs associated with edges from one or more edge data frames, or, a graph object.

Usage

```
get_edges(..., return_type = "list")
```

Arguments

...	a collection of edge data frames or graph objects.
return_type	using list (the default) will provide a list object containing vectors of outgoing and incoming node IDs associated with edges. With df, a data frame containing outgoing and incoming node IDs associated with edges. With vector or string, a vector of character objects representing the edges is provided.

Value

a list, data frame, or a vector object, depending on the value given to `return_type`.

Examples

```
## Not run:
# Before getting node ID values, create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

# Can get the 'outgoing' and 'incoming' node ID values
# in a list object
get_edges(graph, return_type = "list") # the default
#> [[1]]
#>  [1] "A" "H" "W" "U" "I" "M" "U" "T" "I" "R" "O"
#> [12] "G" "O" "A" "V" "I" "M" "K" "R" "T" "Y" "R"
#> [23] "M" "L" "H" "V"

#> [[2]]
#>  [1] "Z" "U" "O" "K" "V" "M" "N" "C" "D" "Z" "B"
#> [12] "G" "U" "Y" "H" "V" "R" "V" "Z" "S" "Q" "I"
#> [23] "P" "S" "E" "P"

# Similarly, you can specify that a data frame is given
get_edges(graph, return_type = "df")
#>    from to
#> 1     A  Z
#> 2     H  U
#> 3     W  O
#> 4     U  K
#> 5     I  V
#>..   ... ..

# A character string with node IDs can instead be gotten
```

```
get_edges(graph, return_type = "vector")
#>  [1] "A -> Z" "H -> U" "W -> O" "U -> K" "I -> V"
#>  [6] "M -> M" "U -> N" "T -> C" "I -> D" "R -> Z"
#> [11] "O -> B" "G -> G" "O -> U" "A -> Y" "V -> H"
#> [16] "I -> V" "M -> R" "K -> V" "R -> Z" "T -> S"
#> [21] "Y -> Q" "R -> I" "M -> P" "L -> S" "H -> E"
#> [26] "V -> P"

## End(Not run)
```

`get_nodes` *Get vector of node IDs*

Description

Provides information on the node IDs from one or several node data frames, edge data frames, or graph objects.

Usage

```
get_nodes(...)
```

Arguments

`...`	a collection of node data frames, edge data frames, or a single graph object.

Value

a vector of node ID values.

Examples

```
## Not run:
# Before getting node ID values, create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
```

```
                edges_df = edges,
                graph_attrs = "layout = neato",
                node_attrs = c("fontname = Helvetica",
                               "shape = circle"))

# Get a vector of all nodes in a graph
get_nodes(graph)
#> [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
#> [13] "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X"
#> [25] "Y" "Z"

# Get a vector of node ID values from a node data frame
get_nodes(nodes)
#> [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
#> [13] "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X"
#> [25] "Y" "Z"

# Get a vector of node ID values from an edge data frame
get_nodes(edges)
#> [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
#> [13] "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X"
#> [25] "Y" "Z"

## End(Not run)
```

`get_predecessors` *Get node IDs for predecessor nodes to the specified node*

Description

Provides a vector of node IDs for all nodes that have a connection to the given node.

Usage

```
get_predecessors(graph, node)
```

Arguments

`graph`	a graph object of class `dgr_graph`.
`node`	a node ID for the selected node.

Value

a vector of node ID values.

Examples

```
## Not run:
# Before getting node ID values for predecessors of
# a specified node, create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

# Get predecessors for node "Z" in the graph
get_predecessors(graph, node = "Z")
#> [1] "A" "R" "R"

# If there are no predecessors, NA is returned
get_predecessors(graph, node = "A")
#> [1] NA

## End(Not run)
```

`get_successors` *Get node IDs for successor nodes to the specified node*

Description

Provides a vector of node IDs for all nodes that have a connection from the given node.

Usage

```
get_successors(graph, node)
```

Arguments

`graph`	a graph object of class `dgr_graph`.
`node`	a node ID for the selected node.

Value

a vector of node ID values.

Examples

```
## Not run:
# Before getting node ID values for successors of
# a specified node, create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

# Get sucessors for node "A" in the graph
get_successors(graph, node = "A")
#> [1] "Z" "Y"

# If there are no successors, NA is returned
get_successors(graph, node = "Z")
#> [1] NA

## End(Not run)
```

---

graph_count *Count graphs in a graph series object*

Description

Counts the total number of graphs in a graph series object.

Usage

```
graph_count(graph_series)
```

Arguments

graph_series	a graph series object of type dgr_graph_1D

Value

a numeric vector representing a count of graphs in a graph series object.

Examples

```
## Not run:
# Create three graphs (using \code{magrittr} pipes)
# and create a graph series using those graphs
library(magrittr)

graph_1 <- create_graph() %>%
  add_node("a") %>% add_node("b") %>% add_node("c") %>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_2 <- graph_1 %>%
  add_node("d") %>% add_edges(from = "d", to = "c")

graph_3 <- graph_2 %>%
  add_node("e") %>% add_edges(from = "e", to = "b")

# Create an empty graph series
series <- create_series(series_type = "sequential")

# Add graphs to the graph series
series <- graph_1 %>% add_to_series(series)
series <- graph_2 %>% add_to_series(series)
series <- graph_3 %>% add_to_series(series)

# Count the number of graphs in the graph series
graph_count(series)
#> [1] 3

## End(Not run)
```

---

grViz *R + viz.js*

Description

Make diagrams in R using viz.js with infrastructure provided by htmlwidgets.

Usage

```
grViz(diagram = "", engine = "dot", allow_subst = TRUE, options = NULL,
  width = NULL, height = NULL)
```

Arguments

diagram	spec for a diagram as either text, filename string, or file connection.
engine	string for the Graphviz layout engine; can be dot (default), neato, circo, or twopi. For more information see https://github.com/mdaines/viz.js#usage.
allow_subst	a boolean that enables/disables subsitution functionality.
options	parameters supplied to the htmlwidgets framework.
width	an optional parameter for specifying the width of the resulting graphic in pixels.
height	an optional parameter for specifying the height of the resulting graphic in pixels.

Value

An object of class htmlwidget that will intelligently print itself into HTML in a variety of contexts including the R console, within R Markdown documents, and within Shiny output bindings.

---

grVizOutput *Widget output function for use in Shiny*

Description

Widget output function for use in Shiny

Usage

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

Arguments

outputId	output variable to read from
width	a valid CSS unit for the width or a number, which will be coerced to a string and have px appended.
height	a valid CSS unit for the height or a number, which will be coerced to a string and have px appended.

Examples

```
## Not run:
library(shiny)
library(shinyAce)

ui = shinyUI(fluidPage(fluidRow(
  column(
    width=4
    , aceEditor("ace", selectionId = "selection", value="digraph {A;}")
  ),
  column(
    width = 6
```

```
    , grVizOutput('diagram' )
  )
)))

server = function(input, output){
  output$diagram <- renderGrViz({
    grViz(
      input$ace
    )
  })

}

shinyApp(ui = ui, server = server)

## End(Not run)
```

---

image_icon *Icons and their download locations*

Description

Create a data frame with image icons and their web addresses.

Usage

```
image_icon(icon_name)
```

Arguments

icon_name	the name of the FontAwesome icon.

---

import_graph *Import a graph from various graph formats*

Description

Import a variety of graphs from different graph formats and create a graph object.

Usage

```
import_graph(graph_file, file_type = NULL, graph_name = NULL,
  graph_time = NULL, graph_tz = NULL)
```

Arguments

graph_file	a connection to a graph file.
file_type	the type of file to be imported. Options are: graphml (GraphML) and sif (SIF). If not supplied, the function will infer the type by its file extension.
graph_name	an optional string for labeling the graph object.
graph_time	a date or date-time string (required for insertion of graph into a graph series of the type temporal).
graph_tz	an optional value for the time zone (tz) corresponding to the date or date-time string supplied as a value to graph_time. If no time zone is provided then it will be set to GMT.

Value

a graph object of class dgr_graph.

---

is_graph_directed *Is the graph a directed graph?*

Description

Determines whether a graph is set to be directed or not and returns a logical value to that effect.

Usage

```
is_graph_directed(graph)
```

Arguments

graph	a graph object of class dgr_graph.

Value

a logical value.

Examples

```
## Not run:
# Create a graph with nodes and edges
nodes <-
  create_nodes(nodes = c("a", "b", "c", "d"))

edges <-
  create_edges(from = c("a", "b", "c"),
               to = c("d", "c", "a"))

graph <-
  create_graph(nodes_df = nodes,
```

```
                edges_df = edges)

# Determine whether the graph is directed
is_graph_directed(graph)
#> [1] TRUE

## End(Not run)
```

`is_graph_empty` *Is the graph empty?*

Description

Provides a logical value on whether the graph is empty (i.e., contains no nodes).

Usage

```
is_graph_empty(graph)
```

Arguments

`graph`	a graph object of class `dgr_graph`.

Value

a logical value.

Examples

```
## Not run:
# Create an empty graph
graph <- create_graph()

# Determine whether the graph is empty
is_graph_empty(graph)
#> [1] TRUE

# Create a graph with nodes and edges
nodes <-
  create_nodes(nodes = c("a", "b", "c", "d"))

edges <-
  create_edges(from = c("a", "b", "c"),
               to = c("d", "c", "a"))

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges)

# Determine whether the graph is empty
```

```
is_graph_empty(graph)
#> [1] FALSE

## End(Not run)
```

`mermaid` *R + mermaid.js*

Description

Make diagrams in R using mermaid.js with infrastructure provided by htmlwidgets.

Usage

```
mermaid(diagram = "", ..., width = NULL, height = NULL)
```

Arguments

`diagram`	diagram in mermaid markdown-like language or file (as a connection or file name) containing a diagram specification. If no diagram is provided `diagram = ""` then the function will assume that a diagram will be provided by `tags` and `DiagrammeR` is just being used for dependency injection.
`...`	other arguments and parameters you would like to send to Javascript
`width`	the width of the resulting graphic in pixels.
`height`	the height of the resulting graphic in pixels.

Value

An object of class `htmlwidget` that will intelligently print itself into HTML in a variety of contexts including the R console, within R Markdown documents, and within Shiny output bindings.

Examples

```
## Not run:
# Create a simple graph running left to right (note
# that the whitespace is not important)
DiagrammeR("
  graph LR
    A-->B
    A-->C
    C-->E
    B-->D
    C-->D
    D-->F
    E-->F
")
# Create the equivalent graph but have it running
# from top to bottom
```

```
DiagrammeR("
   graph TB
   A-->B
   A-->C
   C-->E
   B-->D
   C-->D
   D-->F
   E-->F
")

# Create a graph with different node shapes and
# provide fill styles for each node
DiagrammeR("graph LR;A(Rounded)-->B[Squared];B-->C{A Decision};
 C-->D[Square One];C-->E[Square Two];
 style A fill:#E5E25F;  style B fill:#87AB51; style C fill:#3C8937;
 style D fill:#23772C;  style E fill:#B6E6E6;"
)

# Load in the 'mtcars' dataset
data(mtcars)
connections <- sapply(
 1:ncol(mtcars)
  ,function(i){
     paste0(
        i
      ,"(",colnames(mtcars)[i],")---"
      ,i,"-stats("
      ,paste0(
        names(summary(mtcars[,i]))
        ,": "
        ,unname(summary(mtcars[,i]))
        ,collapse="<br/>"
      )
      ,")"
   )
 }
)

# Create a diagram using the 'connections' object
DiagrammeR(
   paste0(
     "graph TD;", "\n",
     paste(connections, collapse = "\n"),"\n",
     "classDef column fill:#0001CC, stroke:#0D3FF3, stroke-width:1px;" ,"\n",
     "class ", paste0(1:length(connections), collapse = ","), " column;"
   )
 )

# Also with \code{DiagrammeR()}, you can use tags
# from \code{htmltools} (just make sure to use
# \code{class = "mermaid"})
library(htmltools)
```

```
diagramSpec = "
graph LR;
  id1(Start)-->id2(Stop);
  style id1 fill:#f9f,stroke:#333,stroke-width:4px;
  style id2 fill:#ccf,stroke:#f66,stroke-width:2px,stroke-dasharray: 5, 5;
"
html_print(tagList(
  tags$h1("R + mermaid.js = Something Special")
  ,tags$pre(diagramSpec)
  ,tags$div(class="mermaid",diagramSpec)
  ,DiagrammeR()
))

# Create a sequence diagram
DiagrammeR("
sequenceDiagram;
   customer->>ticket seller: ask for a ticket;
   ticket seller->>database: seats;
   alt tickets available
     database->>ticket seller: ok;
     ticket seller->>customer: confirm;
     customer->>ticket seller: ok;
     ticket seller->>database: book a seat;
     ticket seller->>printer: print a ticket;
   else sold out
     database->>ticket seller: none left;
     ticket seller->>customer: sorry;
   end
")

## End(Not run)
```

---

node_count *Get count of all nodes or certain types of nodes*

Description

From a graph object of class dgr_graph, get a count of nodes in the graph and optionally obtain a count of nodes by their type.

Usage

```
node_count(graph, type = FALSE)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
type	a logical value, where TRUE provides a named vector of node count by type and FALSE (the default) provides a total count.

Value

a numeric vector of single length.

Examples

```
## Not run:
# Before getting counts of nodes, create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

# Get counts of nodes grouped by the "type" attribute
node_count(graph, type = TRUE) # the default
#> a_to_g  h_to_p  q_to_x y_and_z
#>      7       9       8       2

# Get a total count of nodes with no grouping
node_count(graph, type = FALSE)
#> [1] 26

## End(Not run)
```

---

node_info *Get detailed information on nodes*

Description

Obtain a data frame with detailed information on nodes and their interrelationships within a graph.

Usage

```
node_info(graph)
```

Arguments

graph	a graph object of class dgr_graph.

Value

a data frame containing information specific to each node within the graph.

Examples

```
## Not run:
# Create a simple graph and get node information from it
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))

node_info(graph)
#>    node_ID label    type degree indegree outdegree loops
#> 1        A     A  a_to_g      2        0         2     0
#> 2        W     W  q_to_x      1        0         1     0
#> 3        T     T  q_to_x      2        0         2     0
#> 4        L     L  h_to_p      1        0         1     0
#> 5        F     F  a_to_g      0        0         0     0
#>..      ...   ...     ...    ...      ...       ...   ...

# Import a large graph
power_grid <-
import_graph(system.file("examples/power_grid.graphml",
                         package = "DiagrammeR"))

# Use dplyr::filter to determine which nodes are highly
# connected in this graph
library(dplyr)

high_connect_nodes <-
  filter(node_info(power_grid), degree > 10)$node_ID
```

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

---

node_present *Determine whether a specified node is present in an existing graph object*

Description

From a graph object of class dgr_graph, determine whether a specified node is present.

Usage

```
node_present(graph, node)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
node	a value that may or may not match a node ID in the graph.

Value

a logical value.

Examples

```
## Not run:
# Before finding out whether a particular node is present,
# create a simple graph
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = c(rep("a_to_g", 7),
                        rep("h_to_p", 9),
                        rep("q_to_x", 8),
                        rep("y_and_z",2)))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               label = "edge",
               rel = "letter_to_letter")

graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "shape = circle"))
```

```
# Verify that node with ID 'a' is not in graph
node_present(graph, "a")
#> FALSE

# Is node with ID 'A' in the graph?
node_present(graph, "A")
#> TRUE

# Are all node ID values from the LETTERS vector
in the graph?
all(sapply(LETTERS, function(x) node_present(graph, x)))
#> TRUE

## End(Not run)
```

---

node_type *Create, read, update, delete, or report status of a node type definition*

Description

From a graph object of class dgr_graph, query a node in the graph (using the node ID) and perform operations on the type definition for that node.

Usage

```
node_type(graph, node, action = "read", value = NULL)
```

Arguments

graph	a graph object of class dgr_graph that is created using create_graph.
node	a node ID corresponding to the node to be selected.
action	the operation to perform on the node's type attribute. To remove the type definition from a node, use either delete, remove, or drop. To add a type definition to a node with no type set, use add or create. To update a node's type definition, use update. To return the value of a node type, use read. To determine whether there is a type set for the selected node, use check.
value	a string denoting the node type, to be supplied when either adding or updating a node type definition.

Value

a graph object of class dgr_graph.

`remove_from_series` *Remove graph object from a graph series object*

Description

Remove a single graph object from an array of graph objects in a graph series object.

Usage

```
remove_from_series(graph_series, index = "last")
```

Arguments

`graph_series`	a graph series object from which the graph object will be removed.
`index`	the index of the graph object to be removed from the graph series object.

Value

a graph series object of type `dgr_graph_1D`.

Examples

```
## Not run:
# Create three graphs (using \code{magrittr} pipes)
# and create a graph series using those graphs
library(magrittr)

graph_1 <- create_graph() %>%
  add_node("a") %>% add_node("b") %>% add_node("c") %>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_2 <- graph_1 %>%
  add_node("d") %>% add_edges(from = "d", to = "c")

graph_3 <- graph_2 %>%
  add_node("e") %>% add_edges(from = "e", to = "b")

# Create an empty graph series
series <- create_series(series_type = "sequential")

# Add graphs to the graph series
series <- graph_1 %>% add_to_series(series)
series <- graph_2 %>% add_to_series(series)
series <- graph_3 %>% add_to_series(series)

# Remove the second graph from the graph series
series <- remove_from_series(graph_series = series, index = 2)

## End(Not run)
```

renderDiagrammeR *Widget render function for use in Shiny*

Description

Widget render function for use in Shiny

Usage

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

Arguments

expr	an expression that generates a DiagrammeR graph
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.

renderGrViz *Widget render function for use in Shiny*

Description

Widget render function for use in Shiny

Usage

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

Arguments

expr	an expression that generates a DiagrammeR graph
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.

See Also

grVizOutput for an example in Shiny

render_graph *Render the graph or output in various formats*

Description

Using a dgr_graph object, either render graph in the Viewer or output in various formats.

Usage

```
render_graph(graph, output = "graph", layout = NULL, width = NULL,
  height = NULL)
```

Arguments

graph	a dgr_graph object, created using the create_graph function.
output	a string specifying the output type; graph (the default) renders the graph using the grViz function, vivagraph renders the graph using the vivagraph function, visNetwork renders the graph using the visnetwork function, and DOT outputs DOT code for the graph.
layout	a string specifying a layout type for a vivagraph rendering of the graph, either forceDirected or constant.
width	an optional parameter for specifying the width of the resulting graphic in pixels.
height	an optional parameter for specifying the height of the resulting graphic in pixels.

Examples

```
## Not run:
# Create a graph and then view it in the RStudio Viewer
nodes <-
  create_nodes(nodes = LETTERS,
               label = TRUE,
               type = "letter",
               shape = sample(c("circle", "square"),
                              length(LETTERS),
                              replace = TRUE),
               fillcolor = sample(c("aqua", "orange",
                                    "pink", "lightgreen",
                                    "black", "yellow"),
                                  length(LETTERS),
                                  replace = TRUE))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               rel = "letter_to_letter")


graph <-
```

```
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = twopi",
               node_attrs = c("fontname = Helvetica",
                              "style = filled"),
               edge_attrs = c("color = gray20",
                              "arrowsize = 0.5"))

# Render the graph using Graphviz
render_graph(graph)

# Render the graph using VivaGraph
render_graph(graph, output = "vivagraph")

# Render the graph using visNetwork
render_graph(graph, output = "visNetwork")

## End(Not run)
```

`render_graph_from_series`

Render a graph available in a series

Description

Using a graph series object of type `dgr_graph_1D`, either render graph in the Viewer or output in various formats.

Usage

```
render_graph_from_series(graph_series, graph_no, output = "graph",
  width = NULL, height = NULL)
```

Arguments

`graph_series`	a graph series object of type `dgr_graph_1D`.
`graph_no`	the index of the graph in the graph series.
`output`	a string specifying the output type; `graph` (the default) renders the graph using the `grViz` function, `DOT` outputs DOT code for the graph, and `SVG` provides SVG code for the rendered graph.
`width`	an optional parameter for specifying the width of the resulting graphic in pixels.
`height`	an optional parameter for specifying the height of the resulting graphic in pixels.

Examples

```
## Not run:
# Create three graphs (using \code{magrittr} for piping)
# and create a graph series using those graphs
library(magrittr)

graph_1 <- create_graph() %>%
  add_node("a") %>% add_node("b") %>% add_node("c") %>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_2 <- graph_1 %>%
  add_node("d") %>% add_edges(from = "d", to = "c")

graph_3 <- graph_2 %>%
  add_node("e") %>% add_edges(from = "e", to = "b")

# Create an empty graph series
series <- create_series(series_type = "sequential")

# Add graphs to the graph series
series <- graph_1 %>% add_to_series(series)
series <- graph_2 %>% add_to_series(series)
series <- graph_3 %>% add_to_series(series)

# View the second graph in the series in the Viewer
render_graph_from_series(graph_series = series,
                         graph_no = 2)

## End(Not run)
```

---

replace_in_spec *Razor-like template for diagram specification*

Description

Use Razor-like syntax to define a template for use in a grViz diagram.

Usage

```
replace_in_spec(spec)
```

Arguments

spec	string spec to be parsed and evaluated

Examples

```
## Not run:

# a simple example to use a LETTER as a node label
spec <- "
  digraph { '@1' }

 [1]: LETTERS[1]
"
grViz(replace_in_spec(spec))


spec <- "
digraph a_nice_graph {
node [fontname = Helvetica]
a [label = '@1']
b [label = '@2-1']
c [label = '@2-2']
d [label = '@2-3']
e [label = '@2-4']
f [label = '@2-5']
g [label = '@2-6']
h [label = '@2-7']
i [label = '@2-8']
j [label = '@2-9']
a -> { b c d e f g h i j}
}

[1]: 'top'
[2]: 10:20
"
grViz(replace_in_spec(spec))

## End(Not run)
```

---

select_graph_from_series

Select a graph available in a series

Description

Using a graph series object of type `dgr_graph_1D`, select a graph object.

Usage

```
select_graph_from_series(graph_series, graph_no)
```

Arguments

graph_series	a graph series object of type dgr_graph_1D.
graph_no	the index of the graph in the graph series.

Examples

```
## Not run:
# Create three graphs (using \code{magrittr} for piping)
# and create a graph series using those graphs
library(magrittr)

graph_1 <- create_graph() %>%
  add_node("a") %>% add_node("b") %>% add_node("c") %>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_2 <- graph_1 %>%
  add_node("d") %>% add_edges(from = "d", to = "c")

graph_3 <- graph_2 %>%
  add_node("e") %>% add_edges(from = "e", to = "b")

# Create an empty graph series
series <- create_series(series_type = "sequential")

# Add graphs to the graph series
series <- graph_1 %>% add_to_series(series)
series <- graph_2 %>% add_to_series(series)
series <- graph_3 %>% add_to_series(series)

# Select the second graph in the series
extracted_graph <-
  select_graph_from_series(graph_series = series,
                           graph_no = 2)

## End(Not run)
```

---

series_info *Get detailed information on a graph series*

Description

Obtain a data frame with detailed information on the graphs within a graph series.

Usage

```
series_info(graph_series)
```

Arguments

graph_series a graph series object of type dgr_graph_1D.

Value

a data frame containing information on the graphs within the supplied graph series.

Examples

```
## Not run:
# Create three graphs (using \code{pipeR} for speed)
# and create a graph series using those graphs
library(pipeR)

graph_1 <- create_graph() %>>%
  add_node("a") %>>% add_node("b") %>>% add_node("c") %>>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_2 <- graph_1 %>>%
  add_node("d") %>>% add_edges(from = "d", to = "c")

graph_3 <- graph_2 %>>%
  add_node("e") %>>% add_edges(from = "e", to = "b")

# Create an empty graph series
series <- create_series(series_type = "sequential")

# Add graphs to the graph series
series <- graph_1 %>>% add_to_series(series)
series <- graph_2 %>>% add_to_series(series)
series <- graph_3 %>>% add_to_series(series)

# Get information on the graphs in the series
series_info(series)
#>   graph name date_time   tz nodes edges directed
#> 1     1 <NA>      <NA> <NA>     3     3     TRUE
#> 2     2 <NA>      <NA> <NA>     4     4     TRUE
#> 3     3 <NA>      <NA> <NA>     5     5     TRUE

## End(Not run)
```

---

subset_series *Subset a graph series object*

Description

Subsetting a graph series by the graphs' index positions in the graph series or through selection via graphs' date-time attributes.

Usage

```
subset_series(graph_series, by = "number", values, tz = NULL)
```

Arguments

graph_series	a graph series object of type dgr_graph_1D.
by	either number, which allows for subsetting of the graph series by graph indices, or time which for graph series objects of type temporal allows for a subsetting of graphs by a date-time or time range.
values	where the subsetting of the graph series by to occur via graph indices (where by = number), provide a vector of those indices; when subsetting by time (where by = time), a range of times can be provided as a vector.
tz	the time zone (tz) corresponding to dates or date-time string provided in values (if by = "date").

Value

a graph series object of type dgr_graph_1D.

Examples

```
## Not run:
# Create three graphs (using \code{pipeR} for speed)
# and create a graph series using those graphs
library(magrittr)

graph_time_1 <-
  create_graph(graph_name = "graph_with_time_1",
               graph_time = "2015-03-25 03:00",
               graph_tz = "GMT") %>%
  add_node("a") %>% add_node("b") %>% add_node("c") %>%
  add_edges(from = c("a", "a", "b"),
            to =   c("c", "b", "c"))

graph_time_2 <-
  create_graph(graph_name = "graph_with_time_2",
               graph_time = "2015-03-26 03:00",
               graph_tz = "GMT") %>%
  add_node("d") %>% add_node("e") %>% add_node("f") %>%
  add_edges(from = c("d", "d", "e"),
            to =   c("f", "e", "f"))

graph_time_3 <-
  create_graph(graph_name = "graph_with_time_3",
               graph_time = "2015-03-27 15:00",
               graph_tz = "GMT") %>%
  add_node("x") %>% add_node("y") %>% add_node("z") %>%
  add_edges(from = c("x", "x", "y"),
            to =   c("z", "y", "z"))
```

```
# Create an empty graph series
series_temporal <- create_series(series_type = "temporal")

# Add graphs to the graph series
series_temporal <- graph_time_1 %>% add_to_series(series_temporal)
series_temporal <- graph_time_2 %>% add_to_series(series_temporal)
series_temporal <- graph_time_3 %>% add_to_series(series_temporal)

# Subset graph series by sequence
series_sequence_subset <-
  subset_series(graph_series = series_temporal,
                by = "number",
                values = 2)

graph_count(series_sequence_subset)
#> [1] 1

# Subset graph series by date-time
series_time_subset <-
  subset_series(graph_series = series_temporal,
                by = "time",
                values = c("2015-03-25 12:00",
                           "2015-03-26 12:00"),
                tz = "GMT")

graph_count(series_time_subset)
#> [1] 1

## End(Not run)
```

`trigger_script` *Trigger a script embedded in a graph series object*

Description

Run an R script located inside or referenced from the graph series object in order to migrate the state of one or more contained graphs.

Usage

```
trigger_script(graph_series, script = 1)
```

Arguments

`graph_series`	a graph series object of type `dgr_graph_1D`.
`script`	the index of the script character string or path reference held in in the graph series.

Value

a graph series object of type dgr_graph_1D.

Examples

```
## Not run:
# So, here's a script that essentially takes an empty graph series, and
# creates a new graph on each new day it is triggered. It will create
# random nodes each time it's triggered and add those nodes to the graph
# belonging to the current day. Throughout the script, '_SELF_' refers
# to the graph series in which the script is contained.
sample_node_script <-
'
graph_attrs <-
  c("layout = twopi",
    "overlap = FALSE",
    "outputorder = edgesfirst")

node_attrs <-
  c("shape = circle",
    "fixedsize = TRUE",
    "width = 1",
    "penwidth = 1",
    "color = DodgerBlue",
    "style = filled",
    "fillcolor = Aqua",
    "alpha_fillcolor = 0.5",
    "fontname = Helvetica",
    "fontcolor = Grey25")

edge_attrs <-
  c("arrowhead = dot",
    "minlen = 1.5",
    "color = Green",
    "penwidth = 2")

# If there is no graph available in the series, then, make one!
if (graph_count(graph_series = _SELF_) == 0){

_SELF_ <-
  add_to_series(graph = create_graph(graph_attrs = graph_attrs,
                node_attrs = node_attrs,
                edge_attrs = edge_attrs,
                graph_name = paste0("data_", Sys.Date()),
                graph_time = as.character(Sys.Date()),
                graph_tz = Sys.timezone()),
                graph_series = _SELF_)

}

# Determine the index of the last graph in the series
last_graph_in_series <- graph_count(graph_series = _SELF_)
```

```
# If it is a new day, create a new graph in the series to populate with data
if (Sys.Date() > as.Date(_SELF_$graphs[[last_graph_in_series]]$graph_time,
                         tz = _SELF_$graphs[[last_graph_in_series]]$graph_tz)){

_SELF_ <-
  add_to_series(graph = create_graph(graph_attrs = graph_attrs,
                node_attrs = node_attrs,
                edge_attrs = edge_attrs,
                graph_name = paste0("data_", Sys.Date()),
                graph_time = as.character(Sys.Date()),
                graph_tz = Sys.timezone()),
                graph_series = _SELF_)

last_graph_in_series <-
  graph_count(graph_series = _SELF_)
}

# Create a node to place into the graph
letters <- paste(sample(LETTERS, 5), collapse = "")

# Add node to the most recent graph and attach it to
# another randomly picked node available in the graph.
# Note that adding an edge only works in the case that
# there is at least one node available in the graph.
# For convenience, the relevant graph is extracted from
# the series, then placed back in the series.
if (!is.na(sample(get_nodes(_SELF_$graphs[[last_graph_in_series]]), 1))){

graph <- _SELF_$graphs[[last_graph_in_series]]

graph <- add_node(graph = graph,
                  node = letters)

graph <- add_edges(graph = graph,
                   from = letters,
                   to = sample(get_nodes(graph = graph), 1))

} else {

graph <- _SELF_$graphs[[last_graph_in_series]]

graph <- add_node(graph = graph,
                  node = letters)
}

# Remove old graph from series
_SELF_ <- remove_from_series(graph_series = _SELF_,
                             index = "last")

# Add new graph to correct position in series
# The "add_to_series" function always adds a graph to the
# end of the graph series.
```

```
_SELF_ <- add_to_series(graph = graph,
                        graph_series = _SELF_)

return(_SELF_)
'

# Create an empty graph series of the 'temporal' type and add
# that script as one of the graph series' 'series scripts'
series_temporal <- create_series(series_type = "temporal",
                                 series_scripts = sample_node_script)

# Call the function 60 times, this will generate 60 random nodes
# with 59 edges
for (i in seq(1, 60)){

  series_temporal <-
    trigger_script(graph_series = series_temporal,
                   script = 1)
  if (i == 60) break
}

# Display the results in the RStudio Viewer
render_graph_from_series(graph_series = series_temporal,
                         graph_no = graph_count(series_temporal))

# Get some basic information about the graphs in the graph series object
series_info(series_temporal)

# Write the script to a file
cat(sample_node_script, file = "~/Desktop/sample_node_script.R")

# Create a reference to the file instead of including text directly
# in the 'series_temporal' object
series_temporal <-
  create_series(series_type = "temporal",
                series_scripts = "~/Desktop/sample_node_script.R")

# Call the function 60 times, this will generate 60 random nodes
# with 59 edges
for (i in seq(1, 60)){

  series_temporal <-
    trigger_script(graph_series = series_temporal,
                   script = 1)
  if (i == 60) break
}

# Display the results in the RStudio Viewer
render_graph_from_series(graph_series = series_temporal,
                         graph_no = graph_count(series_temporal))

## End(Not run)
```

visnetwork *Render graph with visNetwork*

Description

Render a graph object with the visNetwork R package.

Usage

```
visnetwork(graph)
```

Arguments

graph	a dgr_graph object, created using the create_graph function.

Examples

```
## Not run:
# Create a node data frame
nodes <-
  create_nodes(nodes = c("a", "b", "c", "d", "e", "f"),
               label = TRUE,
               fillcolor = c("lightgrey", "red", "orange", "pink",
                         "aqua", "yellow"),
               shape = "circle",
               value = c(2, 1, 0.5, 1, 1.8, 1),
               type = c("1", "1", "1", "2", "2", "2"),
               x = c(1, 2, 3, 4, 5, 6),
               y = c(-2, -1, 0, 6, 4, 1))

# Create an edge data frame
edges <-
  create_edges(from = c("a", "b", "c", "d", "f", "e"),
               to = c("d", "c", "a", "c", "a", "d"),
               color = c("green", "green", "grey", "grey",
                         "blue", "blue"),
               rel = "leading_to")

# Create a graph object
graph <- create_graph(nodes_df = nodes,
                      edges_df = edges)

visnetwork(graph)

## End(Not run)
```

vivagraph *Render graph with VivaGraphJS*

Description

Render a graph object with the VivaGraphJS library.

Usage

```
vivagraph(graph = NULL, layout = "forceDirected", positions = NULL,
  config = NULL, height = NULL, width = NULL, elementId = NULL)
```

Arguments

graph	a dgr_graph object, created using the create_graph function.
layout	a string where "forceDirected" is the default whereas "constant" is another layout option.
positions	data.frame of two columns x and y with fixed positions if you intend to provide preset positions for nodes.
config	list of other config options. While currently this does nothing, we expect to add additional configuration options here.
height	string or integer with a valid CSS height for the container for our htmlwidget.
width	string or integer with a valid CSS width for the container for our htmlwidget.
elementId	string with a valid CSS id.

Examples

```
## Not run:
# Create a graph using the \code{create_nodes}, \code{create_edges},
# and \code{create_graph} functions
nodes <-
  create_nodes(nodes = LETTERS,
               type = "letter",
               shape = sample(c("circle", "rectangle"),
                              length(LETTERS),
                              replace = TRUE),
               fillcolor = sample(c("aqua", "gray80",
                                    "pink", "lightgreen",
                                    "azure", "yellow"),
                                  length(LETTERS),
                                  replace = TRUE))

edges <-
  create_edges(from = sample(LETTERS, replace = TRUE),
               to = sample(LETTERS, replace = TRUE),
               rel = "letter_to_letter")
```

```
graph <-
  create_graph(nodes_df = nodes,
               edges_df = edges,
               graph_attrs = "layout = neato",
               node_attrs = c("fontname = Helvetica",
                              "style = filled"),
               edge_attrs = c("color = gray20",
                              "arrowsize = 0.5"))

vivagraph(graph = graph)

## End(Not run)
```

---

x11_hex *X11 colors and hexadecimal color values*

Description

Create a data frame containing information on X11 colors and their corresponding hexadecimal color values.

Usage

```
x11_hex()
```

Index