Package 'g6R'

May 8, 2026

Title Graph Visualisation Engine Widget for R and 'shiny' Apps

Version 0.6.0

Description Create stunning network experiences powered by the 'G6' graph visualisation engine 'JavaScript' library <https://g6.antv.antgroup.com/en>. In 'shiny' mode, modify your graph directly from the server function to dynamically interact with nodes and edges. Select your favorite layout among 20 choices. 15 behaviors are available such as interactive edge creation, collapse-expand and brush select.
17 plugins designed to improve the user experience such as a mini-map, toolbars and grid lines. Customise the look and feel of your graph with comprehensive options for nodes, edges and more.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/cynkra/g6R, https://cynkra.github.io/g6R/

BugReports https://github.com/cynkra/g6R/issues

Imports htmlwidgets, shiny

Suggests knitr, rmarkdown, quarto, igraph, roxy.shinylive, testthat (>= 3.0.0), withr, stringr, htmltools, bslib

VignetteBuilder quarto, knitr

Depends R (>= 4.1.0)

LazyData true

Config/testthat/edition 3

NeedsCompilation no

Author David Granjon [aut, cre],
David Schoch [aut],
cynkra GmbH [fnd],
Bristol Myers Squibb [fnd]

Maintainer David Granjon <dgranjon@ymail.com>

Repository CRAN

Date/Publication 2026-04-27 15:40:02 UTC

Contents

`animation_config` *Create Animation Configuration for G6 Graphs*

Description

Configures animation settings for G6 graph elements. These settings control how graph elements animate when changes occur.

Usage

```
animation_config(
  delay = NULL,
 direction = c("forward", "alternate", "alternate-reverse", "normal", "reverse"),
  duration = NULL,
  easing = NULL,
  fill = c("none", "auto", "backwards", "both", "forwards"),
  iterations = NULL
)
```

Arguments

`delay`	Animation delay time in milliseconds. The time to wait before the animation begins. Must be a non-negative numeric value.
`direction`	Animation playback direction. Options: • `"forward"`: Plays normally (default) • `"alternate"`: Plays forward, then in reverse • `"alternate-reverse"`: Plays in reverse, then forward • `"normal"`: Same as forward • `"reverse"`: Plays in reverse direction
`duration`	Animation duration in milliseconds. The length of time the animation will take to complete one cycle. Must be a non-negative numeric value.
`easing`	Animation easing function. Controls the rate of change during the animation. Common values include "linear", "ease", "ease-in", "ease-out", "ease-in-out", or cubic-bezier values.
`fill`	Fill mode after animation ends. Options: • `"none"`: Element returns to its initial state when animation ends (default)

- "auto": Follows the rules of the animation effect
- "backwards": Element retains first keyframe values during delay period
- "both": Combines forwards and backwards behavior
- "forwards": Element retains final keyframe values after animation ends

iterations Number of times the animation should repeat. A value of Inf will cause the animation to repeat indefinitely. Must be a non-negative numeric value.

Details

Animation configuration allows fine-tuning the timing and behavior of animations in G6 graphs. This includes controlling the duration, delay, easing function, direction, and other aspects of how graph elements animate.

Value

A list containing animation configuration that can be passed to g6_options().

Examples

```
# Basic animation with duration
config <- animation_config(
  duration = 500
)

# Complex animation configuration
config <- animation_config(
  delay = 100,
  duration = 800,
  easing = "ease-in-out",
  direction = "alternate",
  fill = "forwards",
  iterations = 2
)

# Infinite animation
config <- animation_config(
  duration = 1000,
  easing = "linear",
  iterations = Inf
)
```

---

antv_dagre_layout *Generate G6 AntV Dagre layout configuration*

Description

This function creates a configuration list for G6 AntV Dagre layout with all available options as parameters.

Usage

```
antv_dagre_layout(
  rankdir = c("TB", "BT", "LR", "RL"),
  align = c("UL", "UR", "DL", "DR"),
  nodesep = 50,
  nodesepFunc = NULL,
  ranksep = 100,
  ranksepFunc = NULL,
  ranker = c("network-simplex", "tight-tree", "longest-path"),
  nodeSize = NULL,
  controlPoints = FALSE,
  begin = NULL,
  sortByCombo = FALSE,
  edgeLabelSpace = TRUE,
  nodeOrder = NULL,
  radial = FALSE,
  focusNode = NULL,
  preset = NULL,
  ...
)
```

Arguments

rankdir	Layout direction: "TB" (top to bottom), "BT" (bottom to top), "LR" (left to right), or "RL" (right to left).
align	Node alignment: "UL" (upper left), "UR" (upper right), "DL" (down left), or "DR" (down right).
nodesep	Node spacing (px). When rankdir is "TB" or "BT", it's the horizontal. spacing of nodes; when rankdir is "LR" or "RL", it's the vertical spacing of nodes.
nodesepFunc	Function to customize node spacing for different nodes, in the form of function(node) that returns a number. Has higher priority than nodesep.
ranksep	Layer spacing (px). When rankdir is "TB" or "BT", it's the vertical spacing between adjacent layers; when rankdir is "LR" or "RL", it's the horizontal spacing.
ranksepFunc	Function to customize layer spacing, in the form of function(node) that returns a number. Has higher priority than ranksep.
ranker	Algorithm for assigning ranks to nodes: "network-simplex", "tight-tree", or "longest-path".
nodeSize	Node size for collision detection. Can be a single number (same width/height), an array [width, height], or a function that returns either.
controlPoints	Whether to retain edge control points.
begin	Alignment position of the upper left corner of the layout. Can be [x, y] or [x, y, z].
sortByCombo	Whether to sort nodes on the same layer by parentId to prevent combo overlap.
edgeLabelSpace	Whether to leave space for edge labels.
nodeOrder	Reference array of node order on the same layer, containing node id values.

radial	Whether to perform a radial layout based on dagre.
focusNode	Focused node (only used when radial=TRUE). Can be a node ID or node object.
preset	Node positions to reference during layout calculation.
...	Additional parameters to pass to the layout.

Value

A list containing the configuration for G6 AntV Dagre layout.

Examples

```
# Basic dagre layout
dagre_config <- antv_dagre_layout()

# Horizontal layout with custom spacing
dagre_config <- antv_dagre_layout(
  rankdir = "LR",
  align = "UL",
  nodesep = 80,
  ranksep = 150
)

# Radial layout with focus node
dagre_config <- antv_dagre_layout(
  radial = TRUE,
  focusNode = "node1",
  ranker = "tight-tree"
)
```

---

as_g6_node *Coerce to a g6 element object*

Description

Coerce to a g6 element object

Usage

```
as_g6_node(x, ...)

## S3 method for class 'g6_node'
as_g6_node(x, ...)

## S3 method for class 'list'
as_g6_node(x, ...)

as_g6_edge(x, ...)
```

```
## S3 method for class 'g6_edge'
as_g6_edge(x, ...)

## S3 method for class 'list'
as_g6_edge(x, ...)

as_g6_combo(x, ...)

## S3 method for class 'g6_combo'
as_g6_combo(x, ...)

## S3 method for class 'list'
as_g6_combo(x, ...)
```

Arguments

x	An object to be coerced. List or g6 element are supported.
...	Additional arguments (unused).

Value

An element of class g6_node, g6_edge, or g6_combo.

---

as_g6_nodes *Coerce to a list of g6_elements objects*

Description

Coerce to a list of g6_elements objects

Usage

```
as_g6_nodes(x, ...)

as_g6_edges(x, ...)

## S3 method for class 'g6_edges'
as_g6_edges(x, ...)

## S3 method for class 'data.frame'
as_g6_edges(x, ...)

## S3 method for class 'list'
as_g6_edges(x, ...)

as_g6_combos(x, ...)
```

```
## S3 method for class 'g6_combos'
as_g6_combos(x, ...)

## S3 method for class 'data.frame'
as_g6_combos(x, ...)

## S3 method for class 'list'
as_g6_combos(x, ...)
```

Arguments

x	An object to be coerced. Data frame, list or g6 element are supported.
...	Additional arguments (unused).

Value

An object of class g6_nodes, g6_edges, or g6_combos.

---

as_g6_port *Coerce to a g6_port object*

Description

Coerce to a g6_port object

Usage

```
as_g6_port(x, ...)

## S3 method for class 'g6_port'
as_g6_port(x, ...)

## S3 method for class 'list'
as_g6_port(x, ...)
```

Arguments

x	An object to coerce.
...	Additional arguments (unused).

Value

An object of class g6_port.

Examples

```
as_g6_port(list(key = "input-1", type = "input", placement = "left"))
as_g6_port(g6_port("input-1", type = "input"))
```

as_g6_ports *Coerce to a list of g6_port objects (g6_ports)*

Description

Coerce to a list of g6_port objects (g6_ports)

Usage

```
as_g6_ports(x, ...)

## S3 method for class 'g6_ports'
as_g6_ports(x, ...)

## S3 method for class 'list'
as_g6_ports(x, ...)
```

Arguments

x	An object to coerce.
...	Additional arguments (unused).

Value

An object of class g6_ports.

Examples

```
as_g6_ports(list(
  list(key = "input-1", type = "input", placement = "left"),
  list(key = "output-1", type = "output", placement = "right")
))
as_g6_ports(g6_ports(
  g6_port("input-1", type = "input"),
  g6_port("output-1", type = "output")
))
```

auto_adapt_label *Configure Auto Adapt Label Behavior*

Description

Creates a configuration object for the auto-adapt-label behavior in G6. This behavior automatically adjusts label positions to reduce overlapping and improve readability in the graph visualization.

Usage

```
auto_adapt_label(
  key = "auto-adapt-label",
  enable = TRUE,
  throttle = 100,
  padding = 0,
  sort = NULL,
  sortNode = list(type = "degree"),
  sortEdge = NULL,
  sortCombo = NULL,
  ...
)
```

Arguments

key	Unique identifier for the behavior (string, default: "auto-adapt-label").
enable	Whether to enable this behavior (JS function, default: returns TRUE for all events).
throttle	Throttle time in milliseconds to optimize performance (numeric, default: 100).
padding	Padding space around labels in pixels (numeric, default: 0).
sort	Global sorting rule for all element types (list or JS function, default: NULL).
sortNode	Sorting rule specifically for node labels (list, default: list(type = "degree")).
sortEdge	Sorting rule specifically for edge labels (list, default: NULL).
sortCombo	Sorting rule specifically for combo labels (list, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/auto-adapt-label.

Sorting parameters determine which labels take priority when space is limited:

- When `sort` is provided, it applies to all element types and overrides type-specific settings
- Type-specific sorting (`sortNode`, `sortEdge`, `sortCombo`) only applies when `sort` is NULL
- The default sorting for nodes is by degree (higher degree nodes' labels are shown first)

Value

A list with the configuration settings for the auto-adapt-label behavior.

Examples

```
# Basic configuration with defaults
config <- auto_adapt_label()

# Custom configuration with more padding and custom throttle
config <- auto_adapt_label(
  key = "my-label-adapter",
```

```
  throttle = 200,
  padding = 5
)


# Using a custom enable function
config <- auto_adapt_label(
  enable = JS("(e) => e.targetType === 'node'")
)
```

---

auto_fit_config *Create Auto-Fit Configuration for G6 Graphs*

Description

Configures the auto-fit behavior for a G6 graph. Auto-fit automatically adjusts the view to fit all elements or centers them within the canvas.

Usage

```
auto_fit_config(
  type = c("view", "center"),
  when = c("overflow", "always"),
  direction = c("x", "y", "both"),
  duration = 1000,
 easing = c("ease-in-out", "ease", "ease-in", "ease-out", "linear", "cubic-bezier",
    "step-start", "step-end")
)
```

Arguments

type	The auto-fit mode to use. Options: • "view": Scale and translate the graph to fit all elements within the view (default) • "center": Only translate the graph to center elements without scaling
when	When the auto-fit should be triggered. Options: • "overflow": Trigger auto-fit only when elements overflow the canvas (default) • "always": Always perform auto-fit when the graph data changes
direction	The direction for auto-fit adjustment. Options: • "x": Adjust only along the x-axis • "y": Adjust only along the y-axis • "both": Adjust in both x and y directions (default)
duration	The duration of the auto-fit animation in milliseconds (default: 1000)
easing	The animation easing function to use. Options:

- "ease-in-out": Slow at the beginning and end of the animation (default)
- "ease": Standard easing
- "ease-in": Slow at the beginning
- "ease-out": Slow at the end
- "linear": Constant speed throughout
- "cubic-bezier": Custom cubic-bezier curve
- "step-start": Jump immediately to the end state
- "step-end": Jump at the end to the end state

Details

The auto-fit feature helps ensure that graph elements remain visible within the canvas. It can be configured to either fit all elements to view or center them, and can be triggered under different conditions.

Value

A list containing the auto-fit configuration that can be passed to g6_options().

Examples

```
# Basic auto-fit configuration with default settings
config <- auto_fit_config()

# Auto-fit with only centering (no scaling)
config <- auto_fit_config(type = "center")

# Auto-fit that always triggers when graph data changes
config <- auto_fit_config(when = "always")

# Auto-fit only in the x direction
config <- auto_fit_config(direction = "x")

# Auto-fit with a fast animation
config <- auto_fit_config(duration = 300, easing = "ease-out")
```

---

background *Configure Background Plugin for G6*

Description

Creates a configuration object for the background plugin in G6. This plugin adds a customizable background to the graph canvas.

Usage

```
background(
  key = NULL,
  width = "100%",
  height = "100%",
  backgroundColor = NULL,
  backgroundImage = NULL,
  backgroundSize = "cover",
  backgroundPosition = NULL,
  backgroundRepeat = NULL,
  opacity = NULL,
  transition = "background 0.5s",
  zIndex = "-1",
  ...
)
```

Arguments

key	Unique identifier for updates (string, default: NULL).
width	Background width (string, default: "100%").
height	Background height (string, default: "100%").
backgroundColor	Background color (string, default: NULL).
backgroundImage	Background image URL (string, default: NULL).
backgroundSize	Background size (string, default: "cover").
backgroundPosition	Background position (string, default: NULL).
backgroundRepeat	Background repeat (string, default: NULL).
opacity	Background opacity (string, default: NULL).
transition	Transition animation (string, default: "background 0.5s").
zIndex	Stacking order (string, default: "-1").
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/background.

Value

A list with the configuration settings for the background plugin.

Examples

```
# Basic background color
bg <- background(backgroundColor = "#f0f0f0")

# Background with image
```

```
bg <- background(
  backgroundImage = "https://example.com/background.jpg",
  backgroundSize = "contain",
  backgroundRepeat = "no-repeat",
  backgroundPosition = "center"
)

# Semi-transparent background with transition
bg <- background(
  backgroundColor = "#000000",
  opacity = "0.3",
  transition = "all 1s ease-in-out"
)
```

`brush_select` *Configure Brush Selection Interaction*

Description

Creates a configuration object for brush selection interaction in graph visualizations. This function configures how elements are selected when using a brush selection tool.

Usage

```
brush_select(
  key = "brush-select",
  animation = FALSE,
  enable = JS("(e) => {\n      return true;\n    }"),
  enableElements = "node",
  immediately = FALSE,
  mode = c("default", "union", "intersect", "diff"),
  onSelect = NULL,
  state = c("selected", "active", "inactive", "disabled", "highlight"),
  style = NULL,
  trigger = "shift",
  outputId = NULL,
  ...
)
```

Arguments

`key`	Behavior unique identifier. Useful to modify this behavior from JS side.
`animation`	Whether to enable animation (boolean, default: FALSE).
`enable`	Whether to enable brush select functionality (boolean or function, default: TRUE).
`enableElements`	Types of elements that can be selected (character vector, default: "node"). Can be `c("node", "edge", "combo")`.
`immediately`	Whether to select immediately in default mode (boolean, default: FALSE).

mode	Selection mode: "union", "intersect", "diff", or "default" (string, default: "default").
onSelect	Callback for selected element state (JS function).
state	State to switch to when selected (string, default: "selected").
style	Style specification for the selection box (list). See https://g6.antv.antgroup.com/en/manual/behavior/brush-select#style.
trigger	Shortcut keys for selection (character vector).
outputId	Manually pass the Shiny output ID. This is useful when the graph is initialised outside the shiny render function and the ID cannot be automatically inferred. This allows to set input values from the callback function with the right namespace and graph ID. You must typically pass `session$ns("graphid")` to ensure this also works in modules.
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/brush-select.

Value

A list with the configuration settings for the brush select behavior.

Examples

```
# Basic configuration
config <- brush_select()

# Custom configuration
config <- brush_select(
  animation = TRUE,
  enableElements = c("node", "edge"),
  mode = "union",
  state = "highlight",
  style = list(
    fill = "rgba(0, 0, 255, 0.1)",
    stroke = "blue",
    lineWidth = 2
  ),
  trigger = c("Shift")
)
```

---

bubble_sets *Configure Bubble Sets Plugin for G6*

Description

Creates a configuration object for the bubble-sets plugin in G6. This plugin creates bubble-like contours around groups of specified elements.

Usage

```
bubble_sets(
  members,
  key = "bubble-sets",
  avoidMembers = NULL,
  label = TRUE,
  labelPlacement = c("bottom", "left", "right", "top", "center"),
  labelBackground = FALSE,
  labelPadding = 0,
  labelCloseToPath = TRUE,
  labelAutoRotate = TRUE,
  labelOffsetX = 0,
  labelOffsetY = 0,
  labelMaxWidth = NULL,
  maxRoutingIterations = 100,
  maxMarchingIterations = 20,
  pixelGroup = 4,
  edgeR0 = NULL,
  edgeR1 = NULL,
  nodeR0 = NULL,
  nodeR1 = NULL,
  morphBuffer = NULL,
  threshold = NULL,
  memberInfluenceFactor = NULL,
  edgeInfluenceFactor = NULL,
  nonMemberInfluenceFactor = NULL,
  virtualEdges = NULL,
  ...
)
```

Arguments

members	Member elements, including nodes and edges (character/numeric vector, required).
key	Unique identifier for updates (string, default: NULL).
avoidMembers	Elements to avoid when drawing contours (character/numeric vector, default: NULL).
label	Whether to display labels (boolean, default: TRUE).
labelPlacement	Label position (string, default: "bottom").
labelBackground	Whether to display background (boolean, default: FALSE).
labelPadding	Label padding (numeric or numeric vector, default: 0).
labelCloseToPath	Whether the label is close to the contour (boolean, default: TRUE).
labelAutoRotate	Whether the label rotates with the contour (boolean, default: TRUE).

`labelOffsetX`	Label x-axis offset (numeric, default: 0).
`labelOffsetY`	Label y-axis offset (numeric, default: 0).
`labelMaxWidth`	Maximum width of the text (numeric, default: NULL).
`maxRoutingIterations`	Maximum iterations for path calculation (numeric, default: 100).
`maxMarchingIterations`	Maximum iterations for contour calculation (numeric, default: 20).
`pixelGroup`	Number of pixels per potential area group (numeric, default: 4).
`edgeR0`	Edge radius parameter R0 (numeric, default: NULL).
`edgeR1`	Edge radius parameter R1 (numeric, default: NULL).
`nodeR0`	Node radius parameter R0 (numeric, default: NULL).
`nodeR1`	Node radius parameter R1 (numeric, default: NULL).
`morphBuffer`	Morph buffer size (numeric, default: NULL).
`threshold`	Threshold (numeric, default: NULL).
`memberInfluenceFactor`	Member influence factor (numeric, default: NULL).
`edgeInfluenceFactor`	Edge influence factor (numeric, default: NULL).
`nonMemberInfluenceFactor`	Non-member influence factor (numeric, default: NULL).
`virtualEdges`	Whether to use virtual edges (boolean, default: NULL).
`...`	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/bubble-sets.

Value

A list with the configuration settings for the bubble-sets plugin.

Examples

```
# Basic bubble set around specific nodes
bubble <- bubble_sets(
  members = c("node1", "node2", "node3"),
  label = TRUE
)

# More customized bubble set
bubble <- bubble_sets(
  key = "team-a",
  members = c("node1", "node2", "node3", "edge1", "edge2"),
  avoidMembers = c("node4", "node5"),
  labelPlacement = "top",
  labelBackground = TRUE,
  labelPadding = c(4, 2),
  maxRoutingIterations = 150
)
```

```
# Bubble set with advanced parameters
bubble <- bubble_sets(
  members = c("node1", "node2", "node3"),
  pixelGroup = 6,
  edgeR0 = 10,
  nodeR0 = 5,
  memberInfluenceFactor = 0.8,
  edgeInfluenceFactor = 0.5,
  nonMemberInfluenceFactor = 0.3,
  virtualEdges = TRUE
)
```

`canvas_config` *Create Canvas Configuration for G6 Graphs*

Description

Configures the canvas settings for a G6 graph. The canvas is the rendering surface where the graph is drawn.

Usage

```
canvas_config(
  container = NULL,
  devicePixelRatio = NULL,
  width = NULL,
  height = NULL,
  cursor = NULL,
  background = NULL,
  renderer = NULL,
  enableMultiLayer = NULL
)
```

Arguments

`container`	The container element for the canvas. Can be a CSS selector string or an HTML element reference.
`devicePixelRatio`	The device pixel ratio to use for rendering. Higher values provide sharper rendering on high-DPI displays but may impact performance. If not specified, the device's pixel ratio will be used.
`width`	The width of the canvas in pixels.
`height`	The height of the canvas in pixels.
`cursor`	The CSS cursor style to use when hovering over the canvas. Common values include "default", "pointer", "move", etc.

background The background color of the canvas. Can be any valid CSS color value (hex, rgb, rgba, named colors).

renderer A function that returns a renderer for different layers. The function takes a layer parameter which can be 'background', 'main', 'label', or 'transient'.

enableMultiLayer Whether to enable multi-layer rendering. This is a non-dynamic parameter and is only effective during initialization. Multi-layer rendering can improve performance for complex graphs by separating elements into different rendering layers.

Details

Canvas configuration controls how the graph is rendered, including its size, scaling, background, and rendering layer settings. This function provides a structured way to configure all canvas-related options.

Note that many of these settings (container, width, height, devicePixelRatio, background, cursor) can also be set directly in the main graph configuration, which will be automatically converted to canvas configuration items.

Value

A list containing the canvas configuration that can be passed to g6_options().

Examples

```
# Basic canvas configuration
config <- canvas_config(
  container = "#graph-container",
  width = 800,
  height = 600
)

# Canvas with multi-layer rendering enabled
config <- canvas_config(
  container = "#graph-container",
  width = 1000,
  height = 700,
  enableMultiLayer = TRUE,
  cursor = "grab"
)
```

---

circular_layout *Generate G6 AntV circular layout configuration*

Description

This function creates a configuration list for G6 AntV circular layout with all available options as parameters.

Usage

```
circular_layout(
  angleRatio = 1,
  center = NULL,
  clockwise = TRUE,
  divisions = 1,
  nodeSize = 10,
  nodeSpacing = 10,
  ordering = NULL,
  radius = NULL,
  startAngle = 0,
  endAngle = 2 * pi,
  startRadius = NULL,
  endRadius = NULL,
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

angleRatio	How many 2*PI are there between the first node and the last node.
center	Center of layout as vector c(x, y) or c(x, y, z).
clockwise	Is it arranged clockwise?
divisions	Number of segments that nodes are placed on the ring.
nodeSize	Node size (diameter) for collision detection.
nodeSpacing	Minimum distance between rings.
ordering	Basis for sorting nodes ("topology", "topology-directed", or "degree").
radius	Radius of the circle (overrides startRadius and endRadius).
startAngle	Starting angle of the layout.
endAngle	End angle of the layout.
startRadius	Starting radius of the spiral layout.
endRadius	End radius of the spiral layout.
width	Width of layout.
height	Height of layout.
...	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/circular-layout.

Value

A list containing the configuration for G6 AntV circular layout.

Examples

```
circular_config <- circular_layout(
  radius = 200,
  startAngle = 0,
  endAngle = pi,
  clockwise = FALSE
)
```

`click_select` *Configure Click Select Behavior*

Description

Creates a configuration object for the click-select behavior in G6. This allows users to select graph elements by clicking.

Usage

```
click_select(
  key = "click-select",
  animation = TRUE,
  degree = 0,
  enable = TRUE,
  multiple = FALSE,
  state = c("selected", "active", "inactive", "disabled", "highlight"),
  neighborState = c("selected", "active", "inactive", "disabled", "highlight"),
  unselectedState = NULL,
  onClick = NULL,
  trigger = "shift",
  ...
)
```

Arguments

`key`	Behavior unique identifier. Useful to modify this behavior from JS side.
`animation`	Whether to enable animation effects when switching element states (boolean, default: TRUE).
`degree`	Controls the highlight spread range (number or function, default: 0).
`enable`	Whether to enable the click element function (boolean or function, default: TRUE).
`multiple`	Whether to allow multiple selections (boolean, default: FALSE).
`state`	The state applied when an element is selected (string, default: "selected").
`neighborState`	The state applied to elements with n-degree relationships (string, default: "selected").

unselectedState	The state applied to all other elements (string, default: NULL).
onClick	Callback when an element is clicked (function, default: NULL).
trigger	Keys for multi-selection (character vector, default: c("shift")).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/click-select.

Value

A list with the configuration settings for the click select behavior.

Examples

```
# Basic configuration
config <- click_select()

# Custom configuration
config <- click_select(
  animation = FALSE,
  degree = 1,
  multiple = TRUE,
  state = "active",
  neighborState = "highlight",
  unselectedState = "inactive",
  trigger = c("Control")
)

# Example leveraging the input[["<GRAPH_ID>-selected_<ELEMENT_TYPE>"]]
if (interactive()) {
  library(shiny)
  library(g6R)
  library(bslib)

  nodes <- data.frame(id = c("node1", "node2"))
  edges <- data.frame(source = "node1", target = "node2")
  combos <- data.frame(id = "combo1", type = "rect")

  ui <- page_fluid(
    g6_output("graph"),
    verbatimTextOutput("selected_elements")
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(
        nodes = nodes,
        edges = edges,
        combos = combos
      ) |>
        g6_layout() |>
        g6_behaviors(
```

```
          click_select(multiple = TRUE),
          brush_select(
            enableElements = c("node", "edge", "combo"),
            immediately = TRUE
          )
        )
    })

    output$selected_elements <- renderPrint({
      list(
        selected_nodes = input[["graph-selected_node"]],
        selected_edges = input[["graph-selected_edge"]],
        selected_combos = input[["graph-selected_combo"]]
      )
    })
  }

  shinyApp(ui, server)
}
```

---

collapse_expand *Configure Collapse Expand Behavior*

Description

Creates a configuration object for the collapse-expand behavior in G6. This allows users to collapse or expand nodes/combos with child elements.

Usage

```
collapse_expand(
  key = "collapse-expand",
  animation = TRUE,
  enable = TRUE,
  trigger = "dblclick",
  onCollapse = NULL,
  onExpand = NULL,
  align = TRUE,
  ...
)
```

Arguments

key	Behavior unique identifier. Useful to modify this behavior from JS side.
animation	Enable expand/collapse animation effects (boolean, default: TRUE).
enable	Enable expand/collapse functionality (boolean or function, default: TRUE).
trigger	Trigger method: "click" or "dblclick" (string, default: "dblclick").
onCollapse	Callback function when collapse is completed (function, default: NULL).

onExpand	Callback function when expand is completed (function, default: NULL).
align	Align with the target element to avoid view offset (boolean, default: TRUE).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/collapse-expand.

Value

A list with the configuration settings for the collapse-expand behavior.

Examples

```
# Basic configuration
config <- collapse_expand()
```

---

combo_combined_layout *Create an AntV Combo Combined Layout*

Description

Creates a combo combined layout configuration for G6 graphs. This layout algorithm combines different layout strategies for elements inside combos and the outermost layer, providing hierarchical organization of graph elements.

Usage

```
combo_combined_layout(
  center = NULL,
  comboPadding = 10,
  innerLayout = NULL,
  nodeSize = 10,
  outerLayout = NULL,
  spacing = NULL,
  treeKey = NULL,
  ...
)
```

Arguments

center	Layout center coordinates. A numeric vector of length 2 [x, y]. If NULL, uses the graph center. Default is NULL.
comboPadding	Padding value inside the combo, used only for force calculation, not for rendering. It is recommended to set the same value as the visual padding. Can be a number, numeric vector, function, or JS function. Default is 10.
innerLayout	Layout algorithm for elements inside the combo. Should be a Layout object or function. If NULL, uses ConcentricLayout as default.

nodeSize	Node size (diameter), used for collision detection. If not specified, it is calculated from the node's size property. Can be a number, numeric vector, function, or JS function. Default is 10.
outerLayout	Layout algorithm for the outermost layer. Should be a Layout object or function. If NULL, uses ForceLayout as default.
spacing	Minimum spacing between node/combo edges when preventNodeOverlap or preventOverlap is true. Can be a number, function, or JS function for different nodes. Default is NULL.
treeKey	Tree key identifier as a character string. Default is NULL.
...	Additional parameters passed to the layout configuration. See https://g6.antv.antgroup.com/en/manual/layout/combo-combined-layout.

Details

The combo combined layout is particularly useful for graphs with hierarchical structures where you want different layout algorithms for different levels of the hierarchy. The inner layout handles elements within combos, while the outer layout manages the overall arrangement.

Value

A layout configuration object for use with G6 graphs.

See Also

antv_dagre_layout for dagre layout configuration

Examples

```
# Basic combo combined layout
layout <- combo_combined_layout()

# Custom configuration with specific center and padding
layout <- combo_combined_layout(
  comboPadding = 20,
  nodeSize = 15,
  spacing = 10
)
```

---

combo_options *Create Combo Options Configuration for G6 Graphs*

Description

Configures the general options for combos in a G6 graph. These settings control the type, style, state, palette, and animation of combos.

Usage

```
combo_options(
  type = "circle",
  style = NULL,
  state = NULL,
  palette = NULL,
  animation = NULL
)
```

Arguments

type	Combo type. Can be a built-in combo type name or a custom combo name. Built-in types include "circle", "rect", "polygon", etc. Default: "circle".
style	Combo style configuration. Controls the appearance of combos including color, size, border, etc. Default: NULL.
state	Defines the style of the combo in different states, such as hover, selected, disabled, etc. Should be a list mapping state names to style configurations. Default: NULL.
palette	Defines the color palette of the combo, used to map colors based on different data. Default: NULL.
animation	Defines the animation effect of the combo. Can be created with `animation_config()`. Default: NULL.

Details

Combo options allow defining how combos (node groupings) appear and behave in a G6 graph. This includes selecting combo types, setting styles, configuring state-based appearances, defining color palettes, and specifying animation effects.

Value

A list containing combo options configuration that can be passed to `g6_options()`.

Examples

```
# Basic combo options with default circle type
options <- combo_options()

# Rectangle combo with custom style
options <- combo_options(
  type = "rect",
  style = list(
    fill = "#F6F6F6",
    stroke = "#CCCCCC",
    lineWidth = 1
  )
)
```

compact_box_layout *Generate G6 AntV Compact Box layout configuration*

Description

This function creates a configuration list for G6 AntV Compact Box layout with all available options as parameters. The Compact Box layout is designed for efficiently laying out trees and hierarchical structures.

Usage

```
compact_box_layout(
  direction = c("LR", "RL", "TB", "BT", "H", "V"),
  getSide = NULL,
  getId = NULL,
  getWidth = NULL,
  getHeight = NULL,
  getHGap = NULL,
  getVGap = NULL,
  radial = FALSE,
  ...
)
```

Arguments

direction	Layout direction: "LR" (left to right), "RL" (right to left), "TB" (top to bottom), "BT" (bottom to top), "H" (horizontal), or "V" (vertical)
getSide	Function to set the nodes to be arranged on the left/right side of the root node. If not set, the algorithm automatically assigns the nodes to the left/right side. Note: This parameter is only effective when the layout direction is "H". Function format: `function(node) { return "left" or "right" }`
getId	Callback function for generating node IDs. Function format: `function(node) { return string }`
getWidth	Function to calculate the width of each node. Function format: `function(node) { return number }`
getHeight	Function to calculate the height of each node. Function format: `function(node) { return number }`
getHGap	Function to calculate the horizontal gap for each node. Function format: `function(node) { return number }`
getVGap	Function to calculate the vertical gap for each node. Function format: `function(node) { return number }`
radial	Whether to enable radial layout
...	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/compact-box-layout.

Value

A list containing the configuration for G6 AntV Compact Box layout.

Examples

```
# Basic compact box layout
box_config <- compact_box_layout()

# Vertical compact box layout
box_config <- compact_box_layout(
  direction = "TB"
)

# Radial layout
box_config <- compact_box_layout(
  radial = TRUE
)
```

---

concentric_layout *Generate G6 AntV Concentric layout configuration*

Description

This function creates a configuration list for G6 AntV Concentric layout with all available options as parameters. The Concentric layout places nodes in concentric circles based on a centrality measure.

Usage

```
concentric_layout(
  center = NULL,
  clockwise = FALSE,
  equidistant = FALSE,
  width = NULL,
  height = NULL,
  sortBy = "degree",
  maxLevelDiff = NULL,
  nodeSize = 30,
  nodeSpacing = 10,
  preventOverlap = FALSE,
  startAngle = 3/2 * pi,
  sweep = NULL,
  ...
)
```

Arguments

center	The center position of the circular layout [x, y] or [x, y, z]. By default, it's the center position of the current container.
clockwise	Whether nodes are arranged in clockwise order.
equidistant	Whether the distances between rings are equal.
width	The width of the layout. By default, the container width is used.
height	The height of the layout. By default, the container height is used.
sortBy	Specify the sorting basis (node attribute name). The higher the value, the more central the node will be placed. If it is "degree", the degree of the node will be calculated. The higher the degree, the more central the node will be placed.
maxLevelDiff	If the maximum attribute difference of nodes in the same layer is undefined, it will be set to maxValue / 4, where maxValue is the maximum attribute value of the sorting basis. For example, if sortBy is 'degree', then maxValue is the degree of the node with the largest degree among all nodes.
nodeSize	Node size (diameter). Used to prevent collision detection when nodes overlap. Can be a number, a 2-element vector [width, height], or a function that returns a number.
nodeSpacing	Minimum distance between rings, used to adjust the radius. Can be a number, a vector of numbers, or a function that returns a number.
preventOverlap	Whether to prevent node overlap. Must be coordinated with the nodeSize attribute or the data.size attribute in the node data. Only when data.size is set in the data or the nodeSize value that is the same as the current graph node size is configured in the layout, can node overlap collision detection be performed.
startAngle	The arc at which to start layout of nodes (in radians)
sweep	If the radian difference between the first and last nodes in the same layer is undefined, it will be set to 2 * Math.PI * (1 - 1 / |level.nodes|), where level.nodes is the number of nodes in each layer calculated by the algorithm, and |level.nodes| is the number of nodes in the layer.
...	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/concentric-layout.

Value

A list containing the configuration for G6 AntV Concentric layout

Examples

```
# Basic concentric layout
concentric_config <- concentric_layout()

# Custom concentric layout with degree-based sorting and overlap prevention
concentric_config <- concentric_layout(
  clockwise = TRUE,
  sortBy = "degree",
  preventOverlap = TRUE,
```

```
  nodeSize = 30,
  nodeSpacing = 20
)

# Custom concentric layout with specific center and dimensions
concentric_config <- concentric_layout(
  center = c(300, 300),
  width = 600,
  height = 600,
  equidistant = TRUE,
  startAngle = pi
)
```

`context_menu` *Configure Context Menu Behavior*

Description

Creates a configuration object for the context-menu behavior in G6. This allows users to display a context menu when right-clicking or clicking on graph elements.

Usage

```
context_menu(
  key = "contextmenu",
  className = "g6-contextmenu",
  trigger = "contextmenu",
  offset = c(4, 4),
  onClick = NULL,
  getItems = NULL,
  getContent = NULL,
  loadingContent = NULL,
  enable = JS("(e) => e.targetType === 'node'"),
  ...
)
```

Arguments

`key`	Unique identifier for the behavior, used for subsequent operations (string, default: "context-menu").
`className`	Additional class name for the menu DOM (string, default: "g6-contextmenu").
`trigger`	How to trigger the menu: "contextmenu" for right-click, "click" for click (string, default: "contextmenu").
`offset`	Offset of the menu display in X and Y directions (numeric vector, default: c(4, 4)).
`onClick`	Callback method triggered after menu item is clicked (JS function). Our default allows to create edge or either remove the current node.

getItems	Returns the list of menu items, supports Promise (JS function, default: NULL).
getContent	Returns the content of the menu, supports Promise (JS function, default: NULL).
loadingContent	Menu content used when getContent returns a Promise (string or HTML element, default: NULL).
enable	Whether the context menu is available (boolean or JS function, default: TRUE).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/contextmenu.

Value

A list with the configuration settings for the context menu plugin.

Examples

```
# Basic configuration
config <- context_menu()

# Custom configuration with JavaScript functions
config <- context_menu(
  key = "my-context-menu",
  className = "my-context-menu",
  trigger = "click",
  offset = c(10, 10),
  getItems = JS("(event) => {
    const type = event.itemType;
    const isNode = type === 'node';
    return [
      { key: 'delete', text: 'Delete' },
      { key: 'edit', text: 'Edit' },
      { key: 'details', text: 'View Details', disabled: !isNode }
    ];
  }"),
  onClick = JS("(value, target, current) => {
    if (value === 'delete') {
      // do stuff
  }")
)
if (interactive()) {
  library(shiny)
  library(g6R)
  library(bslib)

  nodes <- data.frame(id = c("node1", "node2"))
  edges <- data.frame(source = "node1", target = "node2")

  ui <- page_fluid(
    g6_output("graph"),
    verbatimTextOutput("contextmenu_info")
  )

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

```
    output$graph <- render_g6({
      g6(
        nodes = nodes,
        edges = edges
      ) |>
        g6_layout() |>
        g6_plugins(
          context_menu()
        )
    })

    output$contextmenu_info <- renderPrint({
      input[["graph-contextmenu"]]
    })
  }

  shinyApp(ui, server)
}
```

`create_edge` *Configure Create Edge Behavior*

Description

Creates a configuration object for the create-edge behavior in G6. This allows users to create edges between nodes by clicking or dragging.

Usage

```
create_edge(
  key = "create-edge",
  trigger = "drag",
  enable = FALSE,
  onCreate = NULL,
  onFinish = NULL,
  style = NULL,
  notify = FALSE,
  outputId = NULL,
  ...
)
```

Arguments

`key`	Behavior unique identifier. Useful to modify this behavior from JS side.
`trigger`	The way to trigger edge creation: "click" or "drag" (string, default: "drag").
`enable`	Whether to enable this behavior (boolean or function, default: FALSE). Our default implementation works in parallel with the context_menu plugin which is responsible for activating the edge behavior when edge creation is selected.

onCreate	Callback function for creating an edge, returns edge data (function, default: NULL).
onFinish	Callback function for successfully creating an edge (function). By default, we provide an internal implementation that disables the edge mode when the edge creation is succesful so that it does not conflict with other drag behaviors.
style	Style of the newly created edge (list, default: NULL).
notify	Whether to show a feedback message in the ui.
outputId	Manually pass the Shiny output ID. This is useful when the graph is initialised outside the shiny render function and the ID cannot be automatically inferred. This allows to set input values from the callback function with the right namespace and graph ID. You must typically pass `session$ns("graphid")` to ensure this also works in modules.
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/create-edge.

Value

A list with the configuration settings for the create-edge behavior.

Examples

```
# Basic configuration
config <- create_edge()
if (interactive()) {
  library(shiny)
  library(bslib)
  library(g6R)

  nodes <- list(
    list(
      id = "node1"
    ),
    list(
      id = "node2"
    )
  )

  modUI <- function(id) {
    ns <- NS(id)
    tagList(
      g6Output(ns("graph"))
    )
  }

  modServer <- function(id) {
    moduleServer(id, function(input, output, session) {
      output$graph <- renderG6({
        g6(nodes) |>
          g6_options(
            animation = FALSE,
```

```
            edge = edge_options(
              style = list(
                endArrow = TRUE
              )
            )
          ) |>
          g6_layout(d3_force_layout()) |>
          g6_behaviors(
            # avoid conflict with internal function
            g6R::create_edge(
              target = c("node", "combo", "canvas"),
              enable = JS(
                "(e) => {
                  return e.shiftKey;
                }"
              ),
              onFinish = JS(
                sprintf(
                  "(edge) => {
                    const graph = HTMLWidgets.find('#%s').getWidget();
                    // Avoid to create edges in combos. If so, we remove it
                    if (edge.targetType === 'combo') {
                      graph.removeEdgeData([edge.id]);
                      return;
                    }
                    Shiny.setInputValue('%s', edge);
                  }",
                  session$ns("graph"),
                  session$ns("added_edge")
                )
              )
            )
          )
      })

      observeEvent(input[["added_edge"]], {
        showNotification(
          sprintf("Edge dropped on: %s", input[["added_edge"]]$targetType),
          type = "message"
        )
      })
    })
  }

  ui <- page_fluid(
    modUI("test")
  )

  server <- function(input, output, session) {
    modServer("test")
  }

  shinyApp(ui, server)
```

```
}
```

`d3_force_layout` *Generate G6 D3 Force layout configuration*

Description

This function creates a configuration list for G6 D3 Force layout with all available options as parameters.

Usage

```
d3_force_layout(
  link = list(distance = 100, strength = 2),
  collide = list(radius = 40),
  ...
)
```

Arguments

`link`	A list specifying force parameters for links (edges), with components: **id** Edge id generation function, format: `function(edge, index, edges) { return string }`. Default is `function(e) e.id` **distance** Ideal edge length that edges will tend toward. Can be a number or a function `function(edge, index, edges) { return number }`. Default is 30 **strength** The strength of the force. Higher values make edge lengths closer to the ideal length. Can be a number or a function `function(edge, index, edges) { return number }`. Default is 1 **iterations** Number of iterations of link force. Default is 1
`collide`	A list specifying collision force parameters for nodes, with components: **radius** Collision radius. Nodes closer than this distance will experience a repulsive force. Can be a number or a function `function(node, index, nodes) { return number }`. Default is 10 **strength** The strength of the repulsive force. Higher values produce more obvious repulsion. Default is 1 **iterations** The number of iterations for collision detection. Default is 1
`...`	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/d3-force.

Value

A list containing the configuration for G6 AntV D3 Force layout.

Examples

```
# Basic D3 force layout
d3_force_config <- d3_force_layout()

# Custom link distance and collision radius
d3_force_config <- d3_force_layout(
  link = list(
    distance = 150,
    strength = 0.5,
    iterations = 3
  ),
  collide = list(
    radius = 30,
    strength = 0.8
  )
)
```

---

dag *Example DAG graph*

Description

The graph is a directed acyclic graph

Usage

```
data(dag)
```

Format

A list with 2 data frames:

nodes data frame with 12 rows for the nodes.

edges data frame with 12 rows for the edges.

combo data frame with 3 rows for the combos.

Source

https://gw.alipayobjects.com/os/antvdemo/assets/data/algorithm-category.json

dagre_layout *Generate G6 AntV Dagre layout configuration*

Description

This function creates a configuration list for G6 AntV Dagre layout with all available options as parameters. The Dagre layout is designed for directed graphs, creating hierarchical layouts with nodes arranged in layers.

Usage

```
dagre_layout(
  rankdir = c("TB", "BT", "LR", "RL"),
  align = c("UL", "UR", "DL", "DR"),
  nodesep = 50,
  ranksep = 100,
  ranker = c("network-simplex", "tight-tree", "longest-path"),
  nodeSize = NULL,
  controlPoints = FALSE,
  ...
)
```

Arguments

rankdir	Layout direction: "TB" (top to bottom), "BT" (bottom to top), "LR" (left to right), or "RL" (right to left).
align	Node alignment: "UL" (upper left), "UR" (upper right), "DL" (down left), or "DR" (down right).
nodesep	Node spacing (px). When rankdir is "TB" or "BT", it's the horizontal spacing of nodes; when rankdir is "LR" or "RL", it's the vertical spacing of nodes.
ranksep	Interlayer spacing (px). When rankdir is "TB" or "BT", it's the spacing between adjacent layers in the vertical direction; when rankdir is "LR" or "RL", it represents the spacing between adjacent layers in the horizontal direction.
ranker	The algorithm for assigning a level to each node: "network-simplex" (the network simplex algorithm), "tight-tree" (the compact tree algorithm), or "longest-path" (the longest path algorithm).
nodeSize	G6 custom attribute to specify the node size uniformly or for each node. Can be a single number (same width/height), an array `[width, height]`, or a function that returns either.
controlPoints	Whether to keep the control points of the edge.
...	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/dagre-layout.

Value

A list containing the configuration for G6 AntV Dagre layout.

Examples

```
# Basic dagre layout
dagre_config <- dagre_layout()

# Custom dagre layout with horizontal flow
dagre_config <- dagre_layout(
  rankdir = "LR",
  nodesep = 80,
  ranksep = 150,
  ranker = "tight-tree"
)

# Custom dagre layout with specific node size
dagre_config <- dagre_layout(
  nodeSize = 40,
  controlPoints = TRUE
)
```

---

dendrogram_layout *Generate G6 Dendrogram layout configuration*

Description

This function creates a configuration list for G6 Dendrogram layout with all available options as parameters.

Usage

```
dendrogram_layout(
  direction = c("LR", "RL", "TB", "BT", "H", "V"),
  nodeSep = 20,
  rankSep = 200,
  radial = FALSE,
  ...
)
```

Arguments

direction	Character. Layout direction. Options: "LR", "RL", "TB", "BT", "H", "V". Defaults to "LR".
nodeSep	Numeric. Node spacing, distance between nodes on the same level. Defaults to 20.
rankSep	Numeric. Rank spacing, distance between different levels. Defaults to 200.
radial	Logical. Whether to enable radial layout. Defaults to FALSE.
...	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/dendrogram-layout.

Value

A list containing the configuration for G6 dendrogram layout.

`drag_canvas` *Configure Drag Canvas Behavior*

Description

Creates a configuration object for the drag-canvas behavior in G6. This allows users to drag the canvas to pan the view.

Usage

```
drag_canvas(
  key = "drag-canvas",
  enable = NULL,
  animation = NULL,
  direction = c("both", "x", "y"),
  range = NULL,
  sensitivity = 10,
  trigger = NULL,
  onFinish = NULL,
  ...
)
```

Arguments

`key`	Behavior unique identifier. Useful to modify this behavior from JS side.
`enable`	Whether to enable this behavior (boolean or function, default: function that enables dragging only on canvas).
`animation`	Drag animation configuration for keyboard movement (list, default: NULL).
`direction`	Allowed drag direction: "x", "y", or "both" (string, default: "both").
`range`	Draggable viewport range in viewport size units (number or numeric vector, default: Inf).
`sensitivity`	Distance to trigger a single keyboard movement (number, default: 10).
`trigger`	Keyboard keys to trigger dragging (list, default: NULL).
`onFinish`	Callback function when dragging is completed (function, default: NULL).
`...`	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/drag-canvas.

Value

A list with the configuration settings for the drag-canvas behavior.

Examples

```
# Basic configuration
config <- drag_canvas()

# Custom configuration
config <- drag_canvas(
  enable = TRUE,
  direction = "x",
  range = c(-100, 100),
  sensitivity = 5,
  trigger = list(
   up = "ArrowUp",
   down = "ArrowDown",
   left = "ArrowLeft",
   right = "ArrowRight"
  )
)
```

---

drag_element *Configure Drag Element Behavior*

Description

Creates a configuration object for the drag-element behavior in G6. This allows users to drag nodes and combos in the graph.

Usage

```
drag_element(
  key = "drag-element",
  enable = NULL,
  animation = TRUE,
  state = "selected",
  dropEffect = c("move", "link", "none"),
  hideEdge = c("none", "out", "in", "both", "all"),
  shadow = FALSE,
  cursor = NULL,
  ...
)
```

Arguments

key	Unique identifier for the behavior, used for subsequent operations (string, default: NULL)
enable	Whether to enable the drag function (boolean or function, default: function that enables dragging for nodes and combos).
animation	Whether to enable drag animation (boolean, default: TRUE).

state	Identifier for the selected state of nodes (string, default: "selected").
dropEffect	Defines the operation effect after dragging ends: "link", "move", or "none" (string, default: "move").
hideEdge	Controls the display state of edges during dragging: "none", "out", "in", "both", or "all" (string, default: "none").
shadow	Whether to enable ghost nodes (boolean, default: FALSE).
cursor	Customize the mouse style during dragging (list, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/drag-element.

Value

A list with the configuration settings for the drag-element behavior.

Examples

```
# Basic configuration
config <- drag_element()

# Custom configuration
config <- drag_element(
  key = "my-drag-behavior",
  animation = FALSE,
  dropEffect = "link",
  hideEdge = "both",
  shadow = TRUE,
  cursor = list(
    default = "default",
    grab = "grab",
    grabbing = "grabbing"
  ),
  enable = JS(
   "(e) => {
     return e.targetType === 'node' || e.targetType === 'combo';
   }"
  )
)
```

---

drag_element_force *Configure Drag Element Force Behavior*

Description

Creates a configuration object for the drag-element-force behavior in G6. This allows users to drag nodes and combos with force-directed layout interactions.

Usage

```
drag_element_force(
  key = "drag-element-force",
  fixed = FALSE,
  enable = NULL,
  state = "selected",
  hideEdge = c("none", "out", "in", "both", "all"),
  cursor = NULL,
  ...
)
```

Arguments

key	Unique identifier for the behavior, used for subsequent operations (string, default: "drag-element-force").
fixed	Whether to keep the node position fixed after dragging ends (boolean, default: FALSE).
enable	Whether to enable the drag function (boolean or JS function, default: JS function that enables dragging for nodes and combos).
state	Identifier for the selected state of nodes (string, default: "selected").
hideEdge	Controls the display state of edges during dragging: "none", "out", "in", "both", or "all" (string, default: "none").
cursor	Customize the mouse style during dragging (list, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/drag-element-force.

Value

A list with the configuration settings for the drag-element-force behavior.

Examples

```
# Basic configuration
config <- drag_element_force()

# Custom configuration with JavaScript arrow function and custom key
config <- drag_element_force(
  key = "my-custom-drag-force",
  fixed = TRUE,
  enable = JS("(event) => { return event.targetType === 'node'; }"),
  hideEdge = "both",
  cursor = list(
    default = "default",
    grab = "grab",
    grabbing = "grabbing"
  )
)
```

edge_bundling *Configure Edge Bundling Plugin*

Description

Creates a configuration object for the edge-bundling plugin in G6. This plugin automatically bundles similar edges together to reduce visual clutter.

Usage

```
edge_bundling(
  key = "edge-bundling",
  bundleThreshold = 0.6,
  cycles = 6,
  divisions = 1,
  divRate = 2,
  iterations = 90,
  iterRate = 2/3,
  K = 0.1,
  lambda = 0.1,
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
bundleThreshold	Edge compatibility threshold, determines which edges should be bundled together (number, default: 0.6).
cycles	Number of simulation cycles (number, default: 6).
divisions	Initial number of cut points (number, default: 1).
divRate	Growth rate of cut points (number, default: 2).
iterations	Number of iterations executed in the first cycle (number, default: 90).
iterRate	Iteration decrement rate (number, default: 2/3).
K	Edge strength, affects attraction and repulsion between edges (number, default: 0.1).
lambda	Initial step size (number, default: 0.1).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/edge-bundling.

Value

A list with the configuration settings for the edge-bundling plugin.

Examples

```
# Basic configuration
config <- edge_bundling()

# Custom configuration
config <- edge_bundling(
  key = "my-edge-bundling",
  bundleThreshold = 0.8,
  cycles = 8,
  K = 0.2
)
```

---

edge_filter_lens *Configure Edge Filter Lens Plugin*

Description

Creates a configuration object for the edge-filter-lens plugin in G6. This plugin creates a lens that filters and displays edges within a specific area.

Usage

```
edge_filter_lens(
  key = "edge-filter-lens",
  trigger = c("pointermove", "click", "drag"),
  r = 60,
  maxR = NULL,
  minR = 0,
  scaleRBy = "wheel",
  nodeType = c("both", "source", "target", "either"),
  filter = NULL,
  style = NULL,
  nodeStyle = list(label = FALSE),
  edgeStyle = list(label = TRUE),
  preventDefault = TRUE,
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
trigger	Method to move the lens: "pointermove", "click", or "drag" (string, default: "pointermove").
r	Radius of the lens (number, default: 60).
maxR	Maximum radius of the lens (number, default: NULL - half of the smaller canvas dimension).

minR	Minimum radius of the lens (number, default: 0).
scaleRBy	Method to scale the lens radius (string, default: "wheel").
nodeType	Edge display condition: "both", "source", "target", or "either" (string, default: "both").
filter	Filter out elements that are never displayed in the lens (JS function, default: NULL).
style	Style of the lens (list, default: NULL).
nodeStyle	Style of nodes in the lens (list or JS function, default: list(label = FALSE)).
edgeStyle	Style of edges in the lens (list or JS function, default: list(label = TRUE)).
preventDefault	Whether to prevent default events (boolean, default: TRUE).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/edge-filter-lens.

Value

A list with the configuration settings for the edge-filter-lens plugin.

Examples

```
# Basic configuration
config <- edge_filter_lens()

# Custom configuration
config <- edge_filter_lens(
  key = "my-edge-lens",
  trigger = "drag",
  r = 100,
  nodeType = "either",
  style = list(
    fill = "rgba(200, 200, 200, 0.3)",
    stroke = "#999",
    lineWidth = 2
  ),
  filter = JS("(id, type) => {
    // Only display edges connected to specific nodes
    if (type === 'edge') {
      const edge = graph.getEdgeData(id);
      return edge.source === 'node1' || edge.target === 'node1';
    }
    return true;
  }")
)
```

edge_options *Create Edge Options Configuration for G6 Graphs*

Description

Configures the general options for edges in a G6 graph. These settings control the type, style, state, palette, and animation of edges.

Usage

```
edge_options(
  type = c("line", "polyline", "arc", "quadratic", "cubic", "cubic-vertical",
    "cubic-horizontal", "loop"),
  style = edge_style_options(),
  state = NULL,
  palette = NULL,
  animation = NULL
)
```

Arguments

type	Edge type. Can be a built-in edge type name or a custom edge name. Built-in types include "line", "polyline", "arc", "quadratic", "cubic", "cubic-vertical", "cubic-horizontal", "loop", etc. Default: "line".
style	Edge style configuration. Controls the appearance of edges including color, width, dash patterns, etc. Can be created with edge_style_options(). Default: NULL.
state	Defines the style of the edge in different states, such as hover, selected, disabled, etc. Should be a list mapping state names to style configurations. Default: NULL.
palette	Defines the color palette of the edge, used to map colors based on different data. Default: NULL.
animation	Defines the animation effect of the edge. Can be created with animation_config(). Default: NULL.

Details

Edge options allow defining how edges appear and behave in a G6 graph. This includes selecting edge types, setting styles, configuring state-based appearances, defining color palettes, and specifying animation effects.

Value

A list containing edge options configuration that can be passed to g6_options().

Examples

```
# Basic edge options with default line type
options <- edge_options()

# Curved edge with custom style
options <- edge_options(
  type = "cubic",
  style = edge_style_options(
    stroke = "#1783FF",
    lineWidth = 2,
    endArrow = TRUE
  )
)
```

---

edge_style_options *Create Edge Style Options for G6 Graphs*

Description

Configures the styling options for edges in a G6 graph. These settings control the appearance and interaction behavior of edges.

Usage

```
edge_style_options(
  class = NULL,
  cursor = valid_cursors,
  fill = NULL,
  fillRule = c("nonzero", "evenodd"),
  filter = NULL,
  increasedLineWidthForHitTesting = NULL,
  isBillboard = TRUE,
  lineDash = 0,
  lineDashOffset = 0,
  lineWidth = 1,
  opacity = 1,
  pointerEvents = NULL,
  shadowBlur = NULL,
  shadowColor = NULL,
  shadowOffsetX = NULL,
  shadowOffsetY = NULL,
  shadowType = NULL,
  sourcePort = NULL,
  stroke = "#000",
  strokeOpacity = 1,
  targetPort = NULL,
  transform = NULL,
```

```
  transformOrigin = NULL,
  visibility = c("visible", "hidden"),
  zIndex = -10000,
  ...
)
```

Arguments

class	Edge class name for custom styling with CSS. Default: NULL.
cursor	Edge mouse hover cursor style. Common values include "default", "pointer", "move", etc. Default: "default".
fill	Edge area fill color (for edges with area, like loops). Default: NULL.
fillRule	Edge internal fill rule. Options: "nonzero", "evenodd". Default: NULL.
filter	Edge shadow filter effect. Default: NULL.
increasedLineWidthForHitTesting	When the edge width is too small, this value increases the interaction area to make edges easier to interact with. Default: NULL.
isBillboard	Effective in 3D scenes, always facing the screen so the line width is not affected by perspective projection. Default: TRUE.
lineDash	Edge dashed line style. Numeric vector specifying dash pattern. Default: 0.
lineDashOffset	Edge dashed line offset. Default: 0.
lineWidth	Edge width in pixels. Default: 1.
opacity	Overall opacity of the edge. Value between 0 and 1. Default: 1.
pointerEvents	Whether the edge responds to pointer events. Default: NULL.
shadowBlur	Edge shadow blur effect amount. Default: NULL.
shadowColor	Edge shadow color. Default: NULL.
shadowOffsetX	Edge shadow X-axis offset. Default: NULL.
shadowOffsetY	Edge shadow Y-axis offset. Default: NULL.
shadowType	Edge shadow type. Options: "inner", "outer", "both". Default: NULL.
sourcePort	Source port of the edge connection. Default: NULL.
stroke	Edge color. Default: "#000".
strokeOpacity	Edge color opacity. Value between 0 and 1. Default: 1.
targetPort	Target port of the edge connection. Default: NULL.
transform	CSS transform attribute to rotate, scale, skew, or translate the edge. Default: NULL.
transformOrigin	Rotation and scaling center point. Default: NULL.
visibility	Whether the edge is visible. Options: "visible", "hidden". Default: "visible".
zIndex	Edge rendering level (for layering). Default: 1.
...	Extra parameters.

Details

Edge style options allow fine-grained control over how edges are rendered and behave in a G6 graph. This includes colors, widths, line styles, shadows, visibility, and interaction properties.

Value

A list containing edge style options that can be passed to `edge_options()`.

`fish_eye` *Configure Fish Eye Plugin*

Description

Creates a configuration object for the fisheye plugin in G6. This plugin creates a fisheye lens effect that magnifies elements within a specific area.

Usage

```
fish_eye(
  key = "fish-eye",
  trigger = c("pointermove", "click", "drag"),
  r = 120,
  maxR = NULL,
  minR = 0,
  d = 1.5,
  maxD = 5,
  minD = 0,
  scaleRBy = NULL,
  scaleDBy = NULL,
  showDPercent = TRUE,
  style = NULL,
  nodeStyle = list(label = TRUE),
  preventDefault = TRUE,
  ...
)
```

Arguments

`key`	Unique identifier for the plugin (string, default: NULL).
`trigger`	Method to move the fisheye: "pointermove", "click", or "drag" (string, default: "pointermove").
`r`	Radius of the fisheye (number, default: 120).
`maxR`	Maximum adjustable radius of the fisheye (number, default: NULL - half of the smaller canvas dimension).
`minR`	Minimum adjustable radius of the fisheye (number, default: 0).

d	Distortion factor (number, default: 1.5).
maxD	Maximum adjustable distortion factor (number, default: 5).
minD	Minimum adjustable distortion factor (number, default: 0).
scaleRBy	Method to adjust the fisheye radius: "wheel" or "drag" (string, default: NULL).
scaleDBy	Method to adjust the fisheye distortion factor: "wheel" or "drag" (string, default: NULL).
showDPercent	Whether to show the distortion factor value in the fisheye (boolean, default: TRUE).
style	Style of the fisheye (list, default: NULL).
nodeStyle	Style of nodes in the fisheye (list or JS function, default: list(label = TRUE)).
preventDefault	Whether to prevent default events (boolean, default: TRUE).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/fisheye.

Value

A list with the configuration settings for the fisheye plugin.

Examples

```
# Basic configuration
config <- fish_eye()

# Custom configuration
config <- fish_eye(
  key = "my-fisheye",
  trigger = "drag",
  r = 200,
  d = 2.5,
  scaleRBy = "wheel",
  scaleDBy = "drag",
  style = list(
    stroke = "#1890ff",
    fill = "rgba(24, 144, 255, 0.1)",
    lineWidth = 2
  ),
  nodeStyle = JS("(datum) => {
    return {
      label: true,
      labelCfg: {
        style: {
          fill: '#003a8c',
          fontSize: 14
        }
      }
    };
  }")
)
```

`fix_element_size` *Configure Fix Element Size Behavior*

Description

Creates a configuration object for the fix-element-size behavior in G6. This allows maintaining fixed visual sizes for elements during zoom operations.

Usage

```
fix_element_size(
  key = "fix-element-size",
  enable = TRUE,
  reset = FALSE,
  state = "",
  node = NULL,
  nodeFilter = JS("() => true"),
 edge = list(list(shape = "key", fields = c("lineWidth")), list(shape = "halo", fields =
    c("lineWidth")), list(shape = "label", fields = c("fontSize"))),
  edgeFilter = JS("() => true"),
  combo = NULL,
  comboFilter = JS("() => true"),
  ...
)
```

Arguments

`key`	Unique identifier for the behavior, used for subsequent operations (string, default: "fix-element-size").
`enable`	Whether to enable this interaction (boolean or JS function, default: TRUE).
`reset`	Whether to restore style when elements are redrawn (boolean, default: FALSE).
`state`	Specify the state of elements to fix size (string, default: "")
`node`	Node configuration item(s) to define which attributes maintain fixed size (list or array of lists, default: NULL).
`nodeFilter`	Node filter to determine which nodes maintain fixed size (JS function, default: returns TRUE for all nodes).
`edge`	Edge configuration item(s) to define which attributes maintain fixed size (list or array of lists, default: predefined list).
`edgeFilter`	Edge filter to determine which edges maintain fixed size (JS function, default: returns TRUE for all edges).
`combo`	Combo configuration item(s) to define which attributes maintain fixed size (list or array of lists, default: NULL).
`comboFilter`	Combo filter to determine which combos maintain fixed size (JS function, default: returns TRUE for all combos).
`...`	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/fix-element-size.

Value

A list with the configuration settings for the fix-element-size behavior.

Examples

```
# Basic configuration
config <- fix_element_size()

# Custom configuration with filters and specific shape configurations
config <- fix_element_size(
  key = "my-fix-size-behavior",
  reset = TRUE,
  state = "active",
  node = list(
    list(shape = "circle", fields = c("r", "lineWidth")),
    list(shape = "label", fields = c("fontSize"))
  ),
  nodeFilter = JS("(node) => node.type === 'important'"),
  edge = list(shape = "line", fields = c("lineWidth", "lineDash")),
  edgeFilter = JS("(edge) => edge.weight > 5")
)
```

---

focus_element *Configure Focus Element Behavior*

Description

Creates a configuration object for the focus-element behavior in G6. This behavior allows focusing on specific elements by automatically adjusting the viewport.

Usage

```
focus_element(
  key = "focus-element",
  animation = list(duration = 500, easing = "ease-in"),
  enable = TRUE,
  ...
)
```

Arguments

key	Unique identifier for the behavior, used for subsequent operations (string, default: "focus-element").
animation	Focus animation settings (list, default: list with duration 500ms and easing "ease-in").
enable	Whether to enable the focus feature (boolean or JS function, default: TRUE).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/focus-element.

Value

A list with the configuration settings for the focus-element behavior.

Examples

```
# Basic configuration
config <- focus_element()

# Custom configuration
config <- focus_element(
  key = "my-focus-behavior",
  animation = list(duration = 1000, easing = "ease-out"),
  enable = JS("(event) => event.targetType === 'node'")
)
```

---

force_atlas2_layout *Generate G6 Force Atlas2 layout configuration*

Description

This function creates a configuration list for G6 Force Atlas2 layout with all available options as parameters.

Usage

```
force_atlas2_layout(
  barnesHut = NULL,
  dissuadeHubs = FALSE,
  height = NULL,
  kg = 1,
  kr = 5,
  ks = 0.1,
  ksmax = 10,
  mode = "normal",
  nodeSize = NULL,
  preventOverlap = FALSE,
  prune = NULL,
  tao = 0.1,
  width = NULL,
  center = NULL,
  ...
)
```

Arguments

barnesHut	Logical. Whether to enable quadtree acceleration. When enabled, improves performance for large graphs but may affect layout quality. By default, enabled if node count > 250.

dissuadeHubs	Logical. Whether to enable hub mode. If TRUE, nodes with higher in-degree are more likely to be placed at the center than those with high out-degree. Defaults to FALSE.
height	Numeric. Layout height. Defaults to container height.
kg	Numeric. Gravity coefficient. The larger the value, the more concentrated the layout is at the center. Defaults to 1.
kr	Numeric. Repulsion coefficient. Adjusts the compactness of the layout. The larger the value, the looser the layout. Defaults to 5.
ks	Numeric. Controls the speed of node movement during iteration. Defaults to 0.1.
ksmax	Numeric. Maximum node movement speed during iteration. Defaults to 10.
mode	Character. Clustering mode. Options are "normal" or "linlog". In "linlog" mode, clusters are more compact. Defaults to "normal".
nodeSize	Numeric or function. Node size (diameter). Used for repulsion calculation when preventOverlap is enabled. If not set, uses data.size in node data.
preventOverlap	Logical. Whether to prevent node overlap. When enabled, layout considers node size to avoid overlap. Node size is specified by `nodeSize` or `data.size` in node data. Defaults to FALSE.
prune	Logical. Whether to enable auto-pruning. By default, enabled if node count > 100. Pruning speeds up convergence but may reduce layout quality. Set to FALSE to disable auto-activation.
tao	Numeric. Tolerance for stopping oscillation when layout is near convergence. Defaults to 0.1.
width	Numeric. Layout width. Defaults to container width.
center	Numeric vector of length 2. Layout center in the form `[x, y]`. Each node is attracted to this point, with gravity controlled by `kg`. If not set, uses canvas center.
...	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/force-atlas2-layout.

Value

A list containing the configuration for G6 force atlas2 layout.

Examples

```
if (interactive()) {
  g6(lesmis$nodes, lesmis$edges) |>
    g6_layout(force_atlas2_layout(
      kr = 20,
      preventOverlap = TRUE,
      center = c(250, 250))) |>
    g6_options(autoResize = TRUE) |>
    g6_behaviors(
```

```
      "zoom-canvas",
      drag_element()
    )
}
```

`fruchterman_layout` *Generate G6 Fruchterman layout configuration*

Description

This function creates a configuration list for G6 Fruchterman layout with all available options as parameters.

Usage

```
fruchterman_layout(height = NULL, width = NULL, gravity = 10, speed = 5, ...)
```

Arguments

`height`	Numeric. Layout height. Defaults to container height.
`width`	Numeric. Layout width. Defaults to container width.
`gravity`	Numeric. Central force attracting nodes to the center. Larger values make the layout more compact. Defaults to 10.
`speed`	Numeric. Node movement speed per iteration. Higher values may cause oscillation. Defaults to 5.
`...`	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/fruchterman-layout.

Value

A list containing the configuration for G6 fruchterman layout.

Examples

```
if (interactive()) {
  g6(lesmis$nodes, lesmis$edges) |>
   g6_layout(fruchterman_layout(
     gravity = 5,
     speed = 5
   )) |>
   g6_behaviors(
     "zoom-canvas",
     drag_element()
   )
}
```

`fullscreen` *Configure Fullscreen Plugin*

Description

Creates a configuration object for the fullscreen plugin in G6. This plugin enables fullscreen mode for the graph.

Usage

```
fullscreen(
  key = "fullscreen",
  autoFit = TRUE,
  trigger = NULL,
  onEnter = NULL,
  onExit = NULL,
  ...
)
```

Arguments

`key`	Unique identifier for the plugin (string, default: NULL).
`autoFit`	Whether to auto-fit the canvas size to the screen when in fullscreen mode (boolean, default: TRUE).
`trigger`	Methods to trigger fullscreen, e.g., list(request = "button", exit = "escape") (list, default: NULL).
`onEnter`	Callback function after entering fullscreen mode (JS function, default: NULL).
`onExit`	Callback function after exiting fullscreen mode (JS function, default: NULL).
`...`	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/fullscreen.

Value

A list with the configuration settings for the fullscreen plugin.

Examples

```
# Basic configuration
config <- fullscreen()

# Custom configuration
config <- fullscreen(
  key = "my-fullscreen",
  autoFit = TRUE,
  trigger = list(
    request = "F",
    exit = "Esc"
```

```
  ),
  onEnter = JS("() => {
    console.log('Entered fullscreen mode');
  }"),
  onExit = JS("() => {
    console.log('Exited fullscreen mode');
  }")
)
```

---

g6 *Create a G6 Graph Visualization*

Description

Creates an interactive graph visualization using the G6 graph visualization library. This function is the main entry point for creating G6 graph visualizations in R.

Usage

```
g6(
  nodes = NULL,
  edges = NULL,
  combos = NULL,
  jsonUrl = NULL,
  iconsUrl = "//at.alicdn.com/t/font_2678727_za4qjydwkkh.js",
  width = "100%",
  height = NULL,
  elementId = NULL
)
```

Arguments

nodes	A data frame or list of nodes in the graph. Each node should have at least an "id" field. See 'Data Structure' section for more details. Default: NULL.
edges	A data frame or list of edges in the graph. Each edge should have "source" and "target" fields identifying the connected nodes. See 'Data Structure' section for more details. Default: NULL.
combos	A data frame or list of combo groups in the graph. Each combo should have at least an "id" field. Nodes can be assigned to combos using their "combo" field. See 'Data Structure' section for more details. Default: NULL.
jsonUrl	An url pointing to a valid JSON containing the graph data in G6 format. See https://assets.antv.antgroup.com/g6/20000.json for an example. Can't be used at the same time as nodes, edges, and combos.
iconsUrl	A URL pointing to a JavaScript file containing custom icons. Default provides reasonable set of icons from https://at.alicdn.com/t/project/2678727/caef142c-804a-4a2f-a914-ae82666a31ee.html?spm=a313x.7781069.1998910419.35.

width	Width of the graph container in pixels or as a valid CSS unit. Default: NULL (automatic sizing).
height	Height of the graph container in pixels or as a valid CSS unit. Default: NULL (automatic sizing).
elementId	A unique ID for the graph HTML element. Default: NULL (automatically generated).

Details

The g6 function creates a G6 graph as an htmlwidget that can be used in R Markdown, Shiny applications, or rendered to HTML. It takes graph data in the form of nodes, edges, and optional combo groupings, along with various configuration options for customizing the appearance and behavior of the graph.

Nodes: The nodes parameter should be a data frame or list of nodes with at least an id field for each node. Additional fields can include:

- id (required): Unique identifier for the node. Must be a character.
- type: Node type (e.g., "circle", "rect", "diamond").
- data: Custom data associated with the node.
- style: List of style attributes (color, size, etc.).
- states: String. Initial states for the node, such as selected, active, hover, etc.
- combo: ID of the combo this node belongs to.
- ports: Node ports. Can also be passed inside the style field.

Edges: The edges parameter should be a data frame or list of edges with at least source and target fields identifying the connected nodes. Additional fields can include:

- source (required): ID of the source node. Must be a character.
- target (required): ID of the target node. Must be a character.
- id: Unique identifier for the edge.
- type: Edge type (e.g., "line", "cubic", "arc").
- data: Custom data associated with the edge.
- style: List of style attributes (color, width, etc.).
- states: String. Initial states for the edge.

Combos: The combos parameter is used for grouping nodes and can be a data frame or list with combo definitions. Fields include:

- id (required): Unique identifier for the combo. Must be a character.
- type: String: Combo type. It can be the type of built-in Combo, or the custom Combo.
- data: Custom data associated with the combo.
- style: List of style attributes.
- states: String. Initial states for the combo.
- combo: String. Parent combo ID. If there is no parent combo, it is null.

Nodes are assigned to combos by setting their combo field to the ID of the combo.

Value

An htmlwidget object that can be printed, included in R Markdown documents, or used in Shiny applications. This widget contains the graph data and configuration necessary to render the G6 graph visualization.

Examples

```
# Create a simple graph with two nodes and one edge
nodes <- data.frame(
  id = c("node1", "node2")
)

edges <- data.frame(
  source = "node1",
  target = "node2"
)

g6(nodes = nodes, edges = edges)
```

---

g6-shiny *Shiny bindings for g6*

Description

Output and render functions for using g6 within Shiny applications and interactive Rmd documents.

Usage

```
g6Output(outputId, width = "100%", height = "400px")

g6_output(outputId, width = "100%", height = "400px")

renderG6(expr, env = parent.frame(), quoted = FALSE)

render_g6(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	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 g6
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.

Value

g6Output and g6_output return a Shiny output function that can be used in the UI part of a Shiny app. renderG6 and render_g6 return a Shiny render function that can be used in the server part of a Shiny app to render a g6 element.

---

g6_add_nodes *Add nodes/edges/combos to a g6 graph via proxy*

Description

This function adds one or more nodes/edges/combos to an existing g6 graph instance using a proxy object. This allows updating the graph without completely re-rendering it.

Usage

```
g6_add_nodes(graph, ...)

g6_add_edges(graph, ...)

g6_add_combos(graph, ...)

g6_add_data(graph, data)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
...	Nodes or edges or combos. You can pass a list of nodes/edges/combos, a dataframe or leverage the g6_nodes(), g6_edges() or g6_combos() helpers or pass individual elements like g6_node(), g6_edge() or g6_combo(). Elements structure must be compliant with specifications listed at https://g6.antv.antgroup.com/manual/element/overview.
data	A nested list possibly containing nodes, edges and combo data. Can also be created with the g6_data helper.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

If a node with the same ID already exists, it will not be added again. See https://g6.antv.antgroup.com/en/api/data#graphaddnodedata, https://g6.antv.antgroup.com/en/api/data#graphaddedgedata and https://g6.antv.antgroup.com/en/api/data#graphaddcombodata for more details.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy, g6_remove_nodes

Examples

```
if (interactive()) {
  library(shiny)
  library(g6R)
  library(bslib)

  # Static data defined globally
  nodes <- data.frame(id = 1:3)
  edges <- data.frame(source = c(1, 2), target = c(2, 3))

  ui <- page_fluid(
    title = "Add Nodes Dynamically",
    g6_output("graph"),
    actionButton("add_node", "Add Node")
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(nodes = nodes, edges = edges) |> g6_layout()
    })

    next_id <- reactiveVal(max(nodes$id) + 1)

    observeEvent(input$add_node, {
      g6_add_nodes(g6_proxy("graph"), data.frame(id = next_id()))
      next_id(next_id() + 1)
    })
  }

  shinyApp(ui, server)
}
```

---

g6_add_plugin *Add a plugin to a g6 graph via proxy*

Description

This function adds one or more plugins to an existing g6 graph instance using a proxy object within a Shiny application.

Usage

```
g6_add_plugin(graph, ...)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
...	Named arguments where each name is a plugin type and each value is a list of configuration options for that plugin.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

G6 plugins extend the functionality of the graph visualization with features like minimaps, toolbar controls, contextual menus, and more. This function allows adding these plugins dynamically after the graph has been initialized.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy, g6_update_plugin

---

g6_behaviors *Create G6 Graph Behaviors Configuration*

Description

Configures interaction behaviors for a G6 graph visualization. This function collects and combines multiple behavior configurations into a list that can be passed to graph initialization functions.

Usage

```
g6_behaviors(graph, ...)
```

Arguments

graph	A g6 graph instance.
...	Behavior configuration objects created by behavior-specific functions. These can include any of the following behaviors: **Navigation behaviors:** • drag_canvas() - Drag the entire canvas view • zoom_canvas() - Zoom the canvas view • scroll_canvas() - Scroll the canvas using the wheel • optimize_viewport_transform() - Optimize view transform performance **Selection behaviors:** • click_select() - Click to select graph elements

- `brush_select()` - Select elements by dragging a rectangular area
- `lasso_select()` - Freely draw an area to select elements

Editing behaviors:

- `create_edge()` - Interactively create new edges
- `drag_element()` - Drag nodes or combos
- `drag_element_force()` - Drag nodes in force-directed layout

Data Exploration behaviors:

- `collapse_expand()` - Expand or collapse subtree nodes
- `focus_element()` - Focus on specific elements and automatically adjust the view
- `hover_activate()` - Highlight elements when hovering

Visual Optimization behaviors:

- `fix_element_size()` - Fix the element size to a specified value
- `auto_adapt_label()` - Automatically adjust label position

Value

A list of behavior configuration objects that can be passed to G6 graph initialization

Note

You can create custom behaviors from JavaScript and use them on the R side. See more at https://g6.antv.antgroup.com/en/manual/behavior/custom-behavior.

Examples

```
# Create a basic set of behaviors
behaviors <- g6_behaviors(
  g6(),
  drag_canvas(),
  zoom_canvas(),
  click_select()
)

# Create a more customized set of behaviors
behaviors <- g6_behaviors(
  g6(),
  drag_canvas(),
  zoom_canvas(sensitivity = 1.5),
  hover_activate(state = "highlight"),
  fix_element_size(
    node = list(
      list(shape = "circle", fields = c("r", "lineWidth"))
    )
  )
)
```

g6_canvas_resize *Resize the canvas of a g6 graph via proxy*

Description

This function changes the size of the canvas of an existing g6 graph instance using a proxy object. This allows updating the graph dimensions without completely re-rendering it.

Usage

```
g6_canvas_resize(graph, width, height)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
width	Numeric value specifying the new width of the canvas in pixels.
height	Numeric value specifying the new height of the canvas in pixels.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

See https://g6.antv.antgroup.com/en/api/canvas#graphsetsizewidth-height for more details.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy

g6_collapse_combo *Collapse or expand a combo element in a g6 graph*

Description

This function collapses/expands a specified combo (a group of nodes) in a g6 graph, hiding its member nodes and edges while maintaining the combo itself visible. This is useful for simplifying complex graphs with multiple hierarchical groups.

Usage

```
g6_collapse_combo(graph, id, options = NULL)

g6_expand_combo(graph, id, options = NULL)
```

Arguments

graph	A g6 graph object or a g6_proxy object for Shiny applications.
id	Character string specifying the ID of the combo to collapse/expand.
options	List containing optional configuration parameters for the collapse/expand action: • animate: Logical value indicating whether to animate the collapsing process. Default is TRUE. • align: Logical value to ensure the position of expanded/collapsed nodes remains unchanged.

Details

When a combo is collapsed, its member nodes and edges are hidden from view while the combo itself remains visible, typically shown as a single node. This helps to reduce visual complexity in large graphs with hierarchical groupings.

Value

The modified g6 graph or g6_proxy object, allowing for method chaining.

References

https://g6.antv.antgroup.com/en/api/element#graphcollapseelementid-options, https://g6.antv.antgroup.com/en/api/element#graphexpandelementid-options

---

g6_collapse_options *Configure collapse button for nodes*

Description

Configure collapse button for nodes

Check if object is a g6_collapse_options configuration

Usage

```
g6_collapse_options(
  collapsed = FALSE,
  visibility = c("visible", "hover"),
  placement = "right-top",
  r = 6,
  fill = "#fff",
  stroke = "#9cabd4",
  lineWidth = 1,
  iconStroke = "#9cabd4",
  iconLineWidth = 1.4,
  cursor = "pointer",
  zIndex = 999
)

is_g6_collapse_options(x)
```

Arguments

collapsed	Logical. Whether the node should be collapsed initially. Default is FALSE.
visibility	Character. Visibility mode of the collapse button. Either "visible" (always shown when children exist) or "hover" (shown only on mouse hover). Default is "visible".
placement	Character or numeric vector. Position of the collapse button. Can be one of: "top", "right", "bottom", "left", "right-top", "right-bottom", "left-top", "left-bottom", or a numeric vector of length 2 for custom coordinates.
r	Numeric. Radius of the button. Default is 8.
fill	Character. Fill color of the button background. Default is "#fff".
stroke	Character. Stroke color of the button border. Default is "#CED4D9".
lineWidth	Numeric. Width of the button border. Default is 1.
iconStroke	Character. Stroke color of the +/- icon. Default is "#000".
iconLineWidth	Numeric. Width of the +/- icon lines. Default is 1.4.
cursor	Character. CSS cursor style. Default is "pointer".
zIndex	Numeric. Z-index for layering. Default is 999.
x	An object to check.

Value

A list of collapse button configuration options.

Logical indicating if x is of class g6_collapse_options.

Examples

```
# Default collapse button
g6_collapse_options()

# Custom styled collapse button
g6_collapse_options(
  placement = "right-top",
  fill = "#f0f0f0",
  stroke = "#333",
  r = 10
)

# Collapse button with custom coordinates
g6_collapse_options(placement = c(1, 0.2))
```

---

g6_data *Create a g6_data object*

Description

Create compatible data structure for `g6_add_data()`, `g6_set_data()` or simply `g6()`.

Usage

```
g6_data(nodes = NULL, edges = NULL, combos = NULL)

as_g6_data(x, ...)

## S3 method for class 'g6_data'
as_g6_data(x, ...)

## S3 method for class 'list'
as_g6_data(x, ...)
```

Arguments

nodes	Nodes data.
edges	Edges data.
combos	Combo data.
x	A list with elements nodes, edges, and/or combos.
...	Additional arguments (unused).

Value

An object of class g6_data.

g6_fit_center *Center graph*

Description

This function pans the graph to the center of the viewport

Usage

```
g6_fit_center(graph, animation = NULL)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
animation	Optional list containing animation configuration parameters for the focus action. Common parameters include: • duration: Duration of the animation in milliseconds. • easing: Animation easing function name (e.g., "ease-in", "ease-out"). If NULL, no animation will be applied.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

See https://g6.antv.antgroup.com/en/api/viewport#graphfitcenteranimation for more details.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy

g6_get_nodes *Get the state of nodes/edges/combos in a g6 graph via proxy*

Description

This function gets the state of one or more nodes/edges/combos to an existing g6 graph instance using a proxy object.

Usage

```
g6_get_nodes(graph, nodes)

g6_get_edges(graph, edges)

g6_get_combos(graph, combos)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
nodes, edges, combos	A string or character vector with the IDs of the nodes/edges/combos.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

If a node with the same ID already exists, it will not be added again. See https://g6.antv.antgroup.com/en/api/data#graphgetnodedata for more details.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy

Examples

```
if (interactive()) {
  library(shiny)
  library(bslib)

   ui <- page_fluid(
     verbatimTextOutput("res"),
     g6Output("graph")
   )
```

```
  server <- function(input, output, session) {
    output$graph <- renderG6({
      g6(
        nodes = data.frame(id = c("node1", "node2"))
      ) |>
        g6_options(animation = FALSE) |>
        g6_layout() |>
        g6_behaviors(click_select())
    })

    # Send query to JS
    observeEvent(req(input[["graph-initialized"]]), {
      g6_proxy("graph") |> g6_get_nodes(c("node1", "node2"))
    })

    # Recover query result inside input[["<GRAPH_ID>-<ELEMENT_ID>-state"]]
    output$res <- renderPrint({
      list(
        node1_state = input[["graph-node1-state"]],
        node2_state = input[["graph-node2-state"]]
      )
    })
  }
  shinyApp(ui, server)
}
```

`g6_get_ports` *Extract ports from a g6 graph state via proxy*

Description

These helpers extract ports from the graph state stored in Shiny input, accessed via a g6_proxy object. Ports are grouped by node and can be filtered by type.

Usage

```
g6_get_ports(graph)

g6_get_input_ports(graph)

g6_get_output_ports(graph)
```

Arguments

`graph`	A g6_proxy object.

Value

A named list of ports for each node, optionally filtered by type.

g6_hide_elements *Hide/show elements in a g6 graph*

Description

This function hides/shows specified elements (nodes, edges, or combos) in a g6 graph. Hidden elements are removed from view but remain in the graph data structure.

Usage

```
g6_hide_elements(graph, ids, animation = NULL)

g6_hide_nodes(graph, ids, animation = NULL)

g6_hide_edges(graph, ids, animation = NULL)

g6_hide_combos(graph, ids, animation = NULL)

g6_show_elements(graph, ids, animation = NULL)

g6_show_nodes(graph, ids, animation = NULL)

g6_show_edges(graph, ids, animation = NULL)

g6_show_combos(graph, ids, animation = NULL)
```

Arguments

graph	A g6 graph object or a g6_proxy object for Shiny applications.
ids	Character vector specifying the IDs of elements to hide/show. Can include node IDs, edge IDs, or combo IDs.
animation	Boolean to toggle animation.

Details

When elements are hidden, they are removed from the visual display but still exist in the underlying data structure. This means they can be shown again later using g6_show_elements without having to recreate them.

Hidden elements will not participate in layout calculations, which may cause other elements to reposition. When elements are shown again, the graph may recalculate layout positions, which can cause other elements to reposition.

Value

The modified g6 graph or g6_proxy object, allowing for method chaining.

See Also

g6_show_elements

---

g6_igraph *Create a G6 Graph from an igraph Object*

Description

Converts an igraph graph object into the format required by the g6() function and creates an interactive G6 graph visualization. This is a convenience wrapper around g6() that allows you to work directly with igraph objects.

Usage

```
g6_igraph(
  graph,
  combos = NULL,
  width = "100%",
  height = NULL,
  elementId = NULL
)
```

Arguments

graph	An object of class igraph::igraph containing the graph to visualize.
combos	A data frame or list of combo groups in the graph. Each combo should have at least an "id" field. Nodes can be assigned to combos using their "combo" field. See 'Data Structure' section for more details. Default: NULL.
width	Width of the graph container in pixels or as a valid CSS unit. Default: NULL (automatic sizing).
height	Height of the graph container in pixels or as a valid CSS unit. Default: NULL (automatic sizing).
elementId	A unique ID for the graph HTML element. Default: NULL (automatically generated).

Details

This function extracts the node and edge data from an igraph object, converts them into the appropriate format for G6, and passes them to g6(). Node styling is derived from vertex attributes, and edge styling from edge attributes.

If the graph is directed, edges will automatically be rendered with arrows.

Value

An htmlwidget object that can be rendered in R Markdown, Shiny apps, or the R console.

See Also

g6() for more information about node, edge, and combo structure.

Examples

```
if (require(igraph)) {
  g <- igraph::make_ring(5)
  g6_igraph(g)
}
```

---

g6_layout *Set the layout algorithm for a g6 graph*

Description

This function configures the layout algorithm used to position nodes in a g6 graph.

Usage

```
g6_layout(graph, layout = d3_force_layout())
```

Arguments

graph	A g6 graph object created with g6().
layout	An existing layout function like circular_layout or a string like circular-layout. At minimum, this can be a list that should contain a type element specifying the layout algorithm. Additional parameters depend on the layout type chosen, for instance list(type = "force").

Details

G6 provides several layout algorithms, each suitable for different graph structures:

- **force**: Force-directed layout using physical simulation of forces.
- **random**: Random layout placing nodes randomly.
- **circular**: Arranges nodes on a circle.
- **radial**: Radial layout with nodes arranged outward from a central node.
- **grid**: Arranges nodes in a grid pattern.
- **concentric**: Concentric circles with important nodes in the center.
- **dagre**: Hierarchical layout for directed acyclic graphs.
- **fruchterman**: Force-directed layout based on the Fruchterman-Reingold algorithm.
- **mds**: Multidimensional scaling layout.
- **comboForce**: Force-directed layout specially designed for combo graphs.

Each layout algorithm has specific configuration options. See the G6 documentation for detailed information on each layout and its parameters: https://g6.antv.antgroup.com/en/manual/layout/overview.

Value

The modified g6 graph object with the specified layout, allowing for method chaining.

See Also

g6()

---

g6_node *G6 Graph Elements*

Description

Constructors and validators for G6 node, edge, and combo elements.

Usage

```
g6_node(
  id,
  type = NULL,
  data = NULL,
  style = NULL,
  states = NULL,
  combo = NULL,
  children = NULL,
  ports = NULL,
  collapse = NULL
)

g6_edge(
  source,
  target,
  id = paste(source, target, sep = "-"),
  type = NULL,
  data = NULL,
  style = NULL,
  states = NULL
)

g6_combo(
  id,
  type = NULL,
  data = NULL,
  style = NULL,
  states = NULL,
  combo = NULL,
  collapse = NULL
)
```

```
validate_element(x, ...)

## S3 method for class 'g6_element'
validate_element(x, ...)

## S3 method for class 'g6_node'
validate_element(x, ...)

## S3 method for class 'g6_edge'
validate_element(x, ...)

## S3 method for class 'g6_combo'
validate_element(x, ...)
```

Arguments

id	Character. Unique identifier for the node or combo (required). For edges, this is optional (id is constructed as source-target if not provided).
type	Character. Element type (optional).
data	List. Custom data for the element (optional).
style	List. Element style (optional).
states	Character vector. Initial states for the element (optional).
combo	Character or NULL. Combo ID or parent combo ID (optional).
children	Character vector. Child node IDs (optional, nodes only).
ports	List. Ports definition (optional, nodes only). See g6_ports and g6_port for details.
collapse	List. Collapse button configuration (optional, nodes and combos). See g6_collapse_options for details. For nodes, only used when node has children. For combos, when provided and type is NULL, auto-sets type to "rect-combo-with-extra-button".
source	Character. Source node ID (required, edges only).
target	Character. Target node ID (required, edges only).
x	An object of class g6_element, g6_node, g6_edge, or g6_combo.
...	Additional arguments (unused). the checks on source and target.

Value

An S3 object of class g6_node, g6_edge, or g6_combo (and g6_element).

Examples

```
# Create a node
node <- g6_node(
  id = "n1",
  type = "rect",
  data = list(label = "Node 1"),
```

```
  style = list(color = "red"),
  states = list("selected"),
  combo = NULL,
  children = c("n2", "n3")
)

# Create an edge
edge <- g6_edge(
  source = "n1",
  target = "n2",
  type = "line",
  style = list(width = 2)
)

# Create a combo
combo <- g6_combo(
  id = "combo1",
  type = "rect",
  data = list(label = "Combo 1"),
  style = list(border = "dashed"),
  states = list("active"),
  combo = NULL
)

# Validate a node explicitly
validate_element(node)
```

---

g6_nodes *Create and validate lists of G6 elements*

Description

Constructors for lists of G6 node, edge, and combo elements. Each function accepts multiple validated elements and returns a list with the appropriate class. All elements are validated on construction.

S3 generic for validating lists of G6 elements.

Usage

```
g6_nodes(...)

g6_edges(...)

g6_combos(...)

validate_elements(x, ...)

## S3 method for class 'g6_nodes'
validate_elements(x, ...)
```

```
## S3 method for class 'g6_edges'
validate_elements(x, ...)

## S3 method for class 'g6_combos'
validate_elements(x, ...)

## S3 method for class 'g6_nodes'
as_g6_nodes(x, ...)

## S3 method for class 'data.frame'
as_g6_nodes(x, ...)

## S3 method for class 'list'
as_g6_nodes(x, ...)
```

Arguments

...	Additional arguments (unused).
x	An object of class g6_nodes, g6_edges, or g6_combos.

Value

An object of class g6_nodes, g6_edges, or g6_combos.

Invisibly returns the validated object.

Examples

```
nodes <- g6_nodes(
  g6_node(id = "n1"),
  g6_node(id = "n2")
)
edges <- g6_edges(
  g6_edge(source = "n1", target = "n2")
)
combos <- g6_combos(
  g6_combo(id = "c1")
)
```

---

g6_options *Configure Global Options for G6 Graph*

Description

Sets up the global configuration options for a G6 graph including node, edge and combo styles, layout, canvas, animation, and interactive behavior settings.

Usage

```
g6_options(
  graph,
  node = NULL,
  edge = NULL,
  combo = NULL,
  autoFit = NULL,
  canvas = NULL,
  animation = TRUE,
  autoResize = FALSE,
  background = NULL,
  cursor = valid_cursors,
  devicePixelRatio = NULL,
  renderer = NULL,
  padding = NULL,
  rotation = 0,
  x = NULL,
  y = NULL,
  zoom = 1,
  zoomRange = c(0.01, 10),
  theme = "light",
  ...
)
```

Arguments

graph	g6 graph instance.
node	Node configuration. Controls the default appearance and behavior of nodes. Created with node_options(). Default: NULL.
edge	Edge configuration. Controls the default appearance and behavior of edges. Created with edge_options(). Default: NULL.
combo	Combo configuration. Controls the default appearance and behavior of combo nodes. Created with combo_options(). Default: NULL.
autoFit	Automatically fit the graph content to the canvas. Created with auto_fit_config(). Default: NULL.
canvas	Canvas configuration for the graph rendering surface. Created with canvas_config(). Default: NULL.
animation	Global animation configuration for graph transitions. Created with animation_config(). Default: TRUE.
autoResize	Whether the graph should automatically resize when the window size changes. Default: FALSE.
background	Background color of the graph. If not specified, the background will be transparent. Default: NULL.
cursor	Default mouse cursor style when hovering over the graph. Options include: "default", "pointer", "move", etc. Default: "default".

devicePixelRatio
: Device pixel ratio for rendering on high-DPI displays. If NULL, the browser's default device pixel ratio will be used. Default: NULL.

renderer
: Rendering engine to use. A JS function. To render as svg, you can pass `() => new SVGRenderer()`. Default: NULL (G6 will choose the appropriate renderer).

padding
: Padding around the graph content in pixels. Can be a single number for equal padding on all sides or a vector of four numbers `[top, right, bottom, left]`. Default: NULL.

rotation
: Rotation angle of the entire graph in degrees. Default: 0.

x
: X-coordinate of the graph's center relative to the container. Default: NULL (will use container center).

y
: Y-coordinate of the graph's center relative to the container. Default: NULL (will use container center).

zoom
: Initial zoom level of the graph. 1 represents 100% (original size). Default: 1.

zoomRange
: Minimum and maximum allowed zoom levels, specified as a vector with two elements: c(min_zoom, max_zoom). Default: c(0.01, 10).

theme
: Color theme for the graph. Either `light` or `dark` or a list representing a custom theme: see https://g6.antv.antgroup.com/en/manual/theme/custom-theme.

...
: Other configuration parameters.

Details

The `g6_options` function provides a comprehensive configuration interface for G6 graphs. It allows you to control all aspects of graph rendering and behavior, from styling of individual elements to global visualization settings.

Value

A list containing all specified G6 graph configuration options.

Examples

```
# Basic usage with defaults
opts <- g6_options(g6())

# Customize node and edge styles
opts <- g6_options(
  g6(),
  node = node_options(
    type = "circle",
    style = node_style_options(
      fill = "#1783FF",
      stroke = "#0066CC"
    )
  ),
  edge = edge_options(
    type = "cubic",
    style = edge_style_options(
```

```
      stroke = "#999999",
      lineWidth = 1.5
    )
  )
)

# Configure graph with dark theme, auto-resize, and custom background
opts <- g6_options(
  g6(),
  theme = "dark",
  autoResize = TRUE,
  background = "#222222",
  padding = 20,
  zoom = 0.8,
  zoomRange = c(0.5, 2)
)

# Configure with custom animations
opts <- g6_options(
  g6(),
  animation = animation_config(
    duration = 500,
    easing = "easeCubic"
  ),
  autoFit = auto_fit_config(duration = 300, easing = "ease-out")
)
```

`g6_plugins` *Create a List of G6 Plugins*

Description

Combines multiple G6 plugins into a list that can be passed to a G6 graph configuration. G6 plugins extend the functionality of the base graph visualization with additional features.

Usage

```
g6_plugins(graph, ...)
```

Arguments

`graph`	G6 graph instance.
`...`	G6 plugin configuration objects created with plugin-specific functions

Details

G6 plugins provide extended functionality beyond the core graph visualization capabilities. Plugins are divided into several categories:

Visual Style Enhancement:

- **Grid Line (grid-line):** Displays grid reference lines on the canvas
- **Background (background):** Adds background images or colors to the canvas
- **Watermark (watermark):** Adds a watermark to the canvas to protect copyright
- **Hull (hull):** Creates an outline for a specified set of nodes
- **Bubble Sets (bubble-sets):** Creates smooth bubble-like element outlines
- **Snapline (snapline):** Displays alignment reference lines when dragging elements

Navigation and Overview:

- **Minimap (minimap):** Displays a thumbnail preview of the graph, supporting navigation
- **Fullscreen (fullscreen):** Supports full-screen display and exit for charts
- **Timebar (timebar):** Provides filtering and playback control for temporal data

Interactive Controls:

- **Toolbar (toolbar):** Provides a collection of common operation buttons
- **Context Menu (contextmenu):** Displays a menu of selectable operations on right-click
- **Tooltip (tooltip):** Displays detailed information about elements on hover
- **Legend (legend):** Displays categories and corresponding style descriptions of chart data

Data Exploration:

- **Fisheye (fisheye):** Provides a focus + context exploration experience
- **Edge Filter Lens (edge-filter-lens):** Filters and displays edges within a specified area
- **Edge Bundling (edge-bundling):** Bundles edges with similar paths together to reduce visual clutter

Advanced Features:

- **History (history):** Supports undo/redo operations
- **Camera Setting (camera-setting):** Configures camera parameters in a 3D scene

Value

A list of G6 plugin configurations that can be passed to a G6 graph.

Note

You can also build your own plugins as described at https://g6.antv.antgroup.com/en/manual/plugin/custom-plugin.

Examples

```
# Create a configuration with multiple plugins
plugins <- g6_plugins(
  g6(),
  minimap(),
  grid_line(),
  tooltips(
    getContent = JS("(e, items) => {
      return `<div>${items[0].id}</div>`;
    }")
```

```
  )
)

# Add a context menu and toolbar
plugins <- g6_plugins(
  g6(),
  context_menu(
    key = "my-context-menu",
    className = "my-context-menu",
    trigger = "click",
    offset = c(10, 10),
    getItems = JS("(event) => {
      const type = event.itemType;
      const isNode = type === 'node';
      return [
        { key: 'delete', text: 'Delete' },
        { key: 'edit', text: 'Edit' },
        { key: 'details', text: 'View Details', disabled: !isNode }
      ];
    }"),
    onClick = JS("(value, target, current) => {
      if (value === 'delete') {
        // do stuff
    }")
  ),
  toolbar(
    position = "top-right",
    getItems = JS("() => [
      { id: 'zoom-in', value: 'zoom-in' },
      { id: 'zoom-out', value: 'zoom-out' },
      { id: 'fit', value: 'fit' }
    ]"),
    onClick = JS("(value) => {
      if (value === 'zoom-in') graph.zoomTo(1.1);
      else if (value === 'zoom-out') graph.zoomTo(0.9);
      else if (value === 'fit') graph.fitView();
    }")
  )
)
```

---

g6_port *Create a G6 Port*

Description

Create a G6 Port

Create a single-connection input port

Create an output port (single connection by default)

Usage

```
g6_port(
  key,
  label = key,
  type = c("input", "output"),
  arity = 1,
  visibility = c("visible", "hover", "hidden"),
  ...
)

g6_input_port(
  key,
  label = key,
  arity = 1,
  visibility = c("visible", "hover", "hidden"),
  fill = "#52C41A",
  ...
)

g6_output_port(
  key,
  label = key,
  arity = 1,
  visibility = c("visible", "hover", "hidden"),
  fill = "#FF4D4F",
  ...
)
```

Arguments

key	Character. Unique identifier for the port (required).
label	Character. Label text for the port (optional).
type	Character. Either "input" or "output" (required).
arity	Numeric. Maximum number of connections this port can accept (default: 1). Use 0, Inf, or any non-negative integer.
visibility	Character. Controls port visibility behavior: • "visible": Ports are always shown (default). • "hover": Ports appear only when hovering over the node. • "hidden": Ports are never visible.
...	Additional port style parameters. See https://g6.antv.antgroup.com/en/manual/element/node/base-node#portstyleprops.
fill	Character. Color of the port (default: "#52C41A" for input ports, "#FF4D4F" for output ports).

Value

An S3 object of class 'g6_port'.

Note

To create an (input/output) port with multiple connections, simply set the arity to `Inf` or any positive integer.

Examples

```
g6_port("input-1", label = "port 1", type = "input", arity = 2, placement = "left")
g6_port("output-1", label = "port 2", type = "output", placement = "right")
g6_port("input-2", type = "input", visibility = "hover")
```

---

g6_ports *Create a List of G6 Ports*

Description

Create a List of G6 Ports

Usage

```
g6_ports(...)
```

Arguments

...	One or more g6_port objects.

Value

An S3 object of class 'g6_ports'.

Examples

```
g6_ports(
  g6_port("input-1", type = "input", placement = "left"),
  g6_port("output-1", type = "output", placement = "right")
)
```

g6_proxy *Create a proxy object to modify an existing g6 graph instance*

Description

This function creates a proxy object that can be used to update an existing g6 graph instance after it has been rendered in the UI. The proxy allows for server-side modifications of the graph without completely re-rendering it.

Usage

```
g6_proxy(id, session = shiny::getDefaultReactiveDomain())
```

Arguments

id	Character string matching the ID of the g6 graph instance to be modified.
session	The Shiny session object within which the graph exists. By default, this uses the current reactive domain.

Value

A proxy object of class "g6_proxy" that can be used with g6 proxy methods such as `g6_add_nodes()`, `g6_remove_nodes()`, etc.

g6_remove_nodes *Remove nodes/edge/combos from a g6 graph via proxy*

Description

This function removes one or more nodes/edges/combos from an existing g6 graph instance using a proxy object. This allows updating the graph without completely re-rendering it.

Usage

```
g6_remove_nodes(graph, ids)

g6_remove_edges(graph, ids)

g6_remove_combos(graph, ids)
```

Arguments

graph	A g6_proxy object created with `g6_proxy`.
ids	Character vector or list containing the IDs of the nodes/edges/combos to be removed. If a single ID is provided, it will be converted to a list internally. You can't mix nodes, edges and combos ids, elements have to be of the same type.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

See https://g6.antv.antgroup.com/en/api/data#graphremovenodedata, https://g6.antv.antgroup.com/en/api/data#graphremoveedgedata and https://g6.antv.antgroup.com/en/api/data#graphremovecombodata for more details.

Value

The g6_proxy object (invisibly), allowing for method chaining.

Note

When a node is removed, its connected edges are also removed.

See Also

g6_proxy

Examples

```
if (interactive()) {
  library(shiny)
  library(g6R)
  library(bslib)

  # Static data defined globally
  nodes <- data.frame(id = 1:3)
  edges <- data.frame(source = c(1, 2), target = c(2, 3))

  ui <- page_fluid(
    title = "Remove Nodes Dynamically",
    g6_output("graph"),
    actionButton("remove_node", "Remove Last Node")
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(nodes = nodes, edges = edges) |> g6_layout()
    })

    # Track the next node id and current node ids
    current_ids <- reactiveVal(nodes$id)

    observeEvent(input$remove_node, {
      ids <- current_ids()
      if (length(ids) > 0) {
        remove_id <- tail(ids, 1)
        g6_remove_nodes(g6_proxy("graph"), remove_id)
        current_ids(ids[-length(ids)])
      }
```

```
  })
}

shinyApp(ui, server)

}
```

---

g6_set_nodes *Set the state of nodes/edges/combos in a g6 graph via proxy*

Description

This function sets the state of one or more nodes/edges/combos to an existing g6 graph instance using a proxy object. This allows updating the graph without completely re-rendering it. Valid states are "selected", "active", "inactive", "disabled", or "highlight".

Usage

```
g6_set_nodes(graph, nodes)

g6_set_edges(graph, edges)

g6_set_combos(graph, combos)

g6_set_data(graph, data)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
nodes	A key value pair list with the node id and its state.
edges	A key value pair list with the edge id and its state.
combos	A key value pair list with the combo id and its state.
data	A nested list containing all nodes, edges and combo data. Alternatively you can use g6_data to generate compatible data.

Details

g6_set_data allows to set all graph data at once (nodes, edges and combos).

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

If a node with the same ID already exists, it will not be added again. See https://g6.antv.antgroup.com/en/api/element#graphsetelementstateid-state-options for more details.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy

Examples

```
if (interactive()) {
  # Example setting node states
  library(shiny)
  library(g6R)
  library(bslib)

  nodes <- data.frame(id = 1:2)
  edges <- data.frame(source = 1, target = 2)

  ui <- page_fluid(
    g6_output("graph"),
    actionButton("set_state", "Set Node States")
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(nodes = nodes, edges = edges) |> g6_layout()
    })

    observeEvent(input$set_state, {
      g6_set_nodes(
        g6_proxy("graph"),
        list(`1` = "selected", `2` = "disabled")
      )
    })
  }

  shinyApp(ui, server)

  # Replace data dynamically
  ui <- page_fluid(
    g6_output("graph"),
    actionButton("remove", "Remove All"),
    actionButton("reset", "Reset Graph"),
    verbatimTextOutput("state")
  )

  server <- function(input, output, session) {
    # Store initial state after first render
    initial_state <- reactiveVal(NULL)

    output$graph <- render_g6({
      g6(nodes = nodes, edges = edges) |>
        g6_layout() |>
        g6_options(animation = FALSE)
    })
```

```
  # Save initial state once available
  observe({
    state <- input[["graph-state"]]
    if (!is.null(state) && is.null(initial_state())) {
      initial_state(state)
    }
  })

  # Remove all nodes and edges
  observeEvent(input$remove, {
    g6_set_data(g6_proxy("graph"), list(nodes = list(), edges = list()))
  })

  # Reset graph to initial state
  observeEvent(input$reset, {
    state <- initial_state()
    if (!is.null(state)) {
      g6_set_data(g6_proxy("graph"), state)
    }
  })

  output$state <- renderPrint({
    input[["graph-state"]]
  })
}

shinyApp(ui, server)
}
```

`g6_set_options` *Set options for a g6 graph via proxy*

Description

This function allows updating various configuration options of an existing g6 graph instance using a proxy object within a Shiny application.

Usage

```
g6_set_options(graph, ...)
```

Arguments

`graph`	A g6_proxy object created with `g6_proxy`.
`...`	Named arguments representing the options to update and their new values. These can include any valid g6 graph options such as fitView, animate, modes, etc.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

The function allows updating various graph options dynamically without having to re-render the entire graph. This is useful for changing behavior, appearance, or interaction modes in response to user input.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy

`g6_set_theme` *Set the theme for a g6 graph via proxy*

Description

This function sets the theme for an existing g6 graph instance

Usage

```
g6_set_theme(graph, theme)
```

Arguments

`graph`	A g6_proxy object created with `g6_proxy`.
`theme`	A character string representing the theme to apply to the graph. There are 2 internal predefined themes: `light` and `dark`. Alternatively, you can pass a custom theme object that conforms to the G6 theme specifications, according to the documentation at https://g6.antv.antgroup.com/en/manual/theme/custom-theme.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

`g6_proxy()`

Examples

```
if (interactive()) {
  library(shiny)
  library(bslib)
  library(g6R)

  themes <- list(
    "light" = list(
      theme = 'light',
      node = list(
        style = list(
          size = 4
        ),
        palette = list(
          type = 'group',
          field = 'cluster'
        )
      ),
      plugins = list(
        list(
          type = 'background',
          background = '#fff'
        )
      )
    ),
    "dark" = list(
      theme = 'dark',
      node = list(
        style = list(
          size = 4
        ),
        palette = list(
          type = 'group',
          field = 'cluster'
        )
      ),
      plugins = list(
        list(
          type = 'background',
          background = '#000'
        )
      )
    ),
    "blue" = list(
      theme = 'light',
      node = list(
        style = list(
          size = 4
        ),
        palette = list(
          type = 'group',
          field = 'cluster',
```

```
          color = "blues",
          invert = TRUE
        )
      ),
      plugins = list(
        list(
          type = 'background',
          background = '#f3faff'
        )
      )
    ),
    "yellow" = list(
      theme = 'light',
      node = list(
        style = list(
          size = 4
        ),
        palette = list(
          type = 'group',
          field = 'cluster',
          color = c(
            '#ffe7ba',
            '#ffd591',
            '#ffc069',
            '#ffa940',
            '#fa8c16',
            '#d46b08',
            '#ad4e00',
            '#873800',
            '#612500'
          )
        )
      ),
      plugins = list(
        list(
          type = 'background',
          background = '#fcf9f1'
        )
      )
    )
  )

  ui <- page_fluid(
    selectInput(
      "theme",
      "Theme",
      choices = c("light", "dark", "blue", "yellow"),
      selected = "default"
    ),
    g6Output("graph", height = "100vh")
  )

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

```
  output$graph <- renderG6({
    g6(
      jsonUrl = "https://assets.antv.antgroup.com/g6/20000.json"
    ) |>
      g6_options(
        animation = FALSE,
        autoFit = "view",
        padding = 20,
        node = list(
          style = list(size = 4),
          palette = list(
            type = "group",
            field = "cluster"
          )
        )
      ) |>
      g6_behaviors(
        "zoom-canvas",
        "drag-canvas",
        "optimize-viewport-transform"
      )
  })

  observeEvent(input$theme, {
    g6_proxy("graph") |>
      g6_set_theme(themes[[input$theme]])
  })

  observeEvent(input[["graph-initialized"]], {
    showNotification(
      "Graph initialized",
      type = "message"
    )
  })

  observeEvent(input[["graph-state"]], {
    showNotification(
      "Graph state updated",
      type = "message"
    )
  })
}

shinyApp(ui, server)

}
```

---

g6_update_behavior *Update a behavior in a g6 graph via proxy*

Description

This function allows updating the configuration of an existing behavior in a g6 graph instance using a proxy object within a Shiny application.

Usage

```
g6_update_behavior(graph, key, ...)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
key	Character string identifying the behavior to update.
...	Named arguments representing the behavior configuration options to update and their new values.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

Behaviors in G6 define how the graph responds to user interactions like dragging, zooming, clicking, etc. This function allows dynamically updating the configuration of these behaviors without having to reinitialize the graph.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy

Examples

```
if (interactive()) {
  library(shiny)
  library(g6R)
  library(bslib)

  nodes <- data.frame(id = c("node1", "node2", "node3"))
  edges <- data.frame(source = "node1", target = "node2")

  ui <- page_fluid(
    g6_output("graph"),
    checkboxInput("enable_click_select", "Enable Click Select", value = TRUE)
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(nodes = nodes, edges = edges) |>
        g6_layout() |>
```

```
      g6_options(animation = FALSE) |>
      g6_behaviors(
        click_select()
      )
  })

  observeEvent(input$enable_click_select, {
    g6_update_behavior(
      g6_proxy("graph"),
      key = "click-select",
      enable = input$enable_click_select
    )
  })
}

shinyApp(ui, server)
}
```

`g6_update_layout` *Execute layout for a g6 graph via proxy*

Description

This function order the execution of the layout of the current graph. It can also update layout options before running it.

Usage

```
g6_update_layout(graph, ...)
```

Arguments

`graph`	A g6_proxy object created with `g6_proxy`.
`...`	Any option to pass to the layout. If so, the layout is updated before running it.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

`g6_proxy`

g6_update_nodes *Update nodes/edges/combos to a g6 graph via proxy*

Description

This function updates one or more nodes/edges/combos to an existing g6 graph instance using a proxy object. This allows updating the graph without completely re-rendering it.

Usage

```
g6_update_nodes(graph, ...)

g6_update_edges(graph, ...)

g6_update_combos(graph, ...)
```

Arguments

graph	A g6_proxy object created with g6_proxy.
...	Nodes or edges or combos. You can pass a list of nodes/edges/combos, a dataframe or leverage the g6_nodes(), g6_edges() or g6_combos() helpers or pass individual elements like g6_node(), g6_edge() or g6_combo(). Elements structure must be compliant with specifications listed at https://g6.antv.antgroup.com/manual/element/overview.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

See https://g6.antv.antgroup.com/en/api/data#graphupdatenodedata, https://g6.antv.antgroup.com/en/api/data#graphupdateedgedata and https://g6.antv.antgroup.com/en/api/data#graphupdatecombodata for more details.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy, g6_remove_nodes

Examples

```
if (interactive()) {
  library(shiny)
  library(g6R)
  library(bslib)
```

```
  # Static data defined globally
  nodes <- data.frame(id = 1:3)
  edges <- data.frame(source = c(1, 2), target = c(2, 3))

  ui <- page_fluid(
    title = "Update Nodes Dynamically",
    g6_output("graph"),
    actionButton("update_node", "Update Node 1 Label")
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(nodes = nodes, edges = edges) |> g6_layout()
    })

    observeEvent(input$update_node, {
      # Update label for node 1
      g6_update_nodes(
        g6_proxy("graph"),
        g6_node(id = 1, style = list(labelText = "Node label updated"))
      )
    })
  }

  shinyApp(ui, server)
}
```

`g6_update_plugin` *Update a plugin in a g6 graph via proxy*

Description

This function allows updating the configuration of an existing plugin in a g6 graph instance using a proxy object within a Shiny application.

Usage

```
g6_update_plugin(graph, key, ...)
```

Arguments

`graph`	A g6_proxy object created with `g6_proxy`.
`key`	Character string identifying the plugin to update.
`...`	Named arguments representing the plugin configuration options to update and their new values.

Details

This function can only be used with a g6_proxy object within a Shiny application. It will not work with regular g6 objects outside of Shiny.

The function allows dynamically updating the configuration of an existing plugin without having to reinitialize it. This is useful for changing plugin behavior or appearance in response to user interactions.

Value

The g6_proxy object (invisibly), allowing for method chaining.

See Also

g6_proxy, g6_add_plugin

Examples

```
if (interactive()) {
  library(shiny)
  library(g6R)
  library(bslib)

  color_to_hex <- function(col) {
    rgb <- col2rgb(col)
    sprintf("#%02X%02X%02X", rgb[1], rgb[2], rgb[3])
  }

  nodes <- data.frame(id = c("node1", "node2", "node3"))
  edges <- data.frame(source = "node1", target = "node2")

  ui <- page_fluid(
    g6_output("graph"),
    actionButton("add_hull", "Add Hull Plugin"),
    selectInput(
      "hull_color",
      "Hull Color",
      choices = c("red", "blue", "green", "orange", "purple"),
      selected = "red"
    )
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(nodes = nodes, edges = edges) |>
        g6_layout() |>
        g6_options(animation = FALSE)
    })

    observeEvent(input$add_hull, {
      g6_add_plugin(
        g6_proxy("graph"),
```

```
      hull(
        members = c("node1", "node2", "node3"),
        fill = color_to_hex(input$hull_color),
        fillOpacity = 0.2
      )
    )
  })

  observeEvent(input$hull_color, {
    # Only update if hull plugin exists
    g6_update_plugin(
      g6_proxy("graph"),
      key = "hull",
      fill = color_to_hex(input$hull_color),
      stroke = color_to_hex(input$hull_color)
    )
  })
}

shinyApp(ui, server)
}
```

`g6_update_ports` *Update ports for one or more nodes in a g6 graph via proxy*

Description

This function sends a message to the client to update (add, remove, or modify) ports for one or more nodes. The actual update is handled on the JS side.

Usage

```
g6_update_ports(graph, ops)
```

Arguments

`graph`	A g6_proxy object.
`ops`	A named list of operations, one entry per node (named by node ID). Each entry can contain: • `add`: a list of port objects to add. • `remove`: a character vector of port keys to remove • `update`: a list of port objects (with key) to update

Details

Removing a port that is currently used by an edge removes the edge as well. Conversely, removing an edge does not remove the ports it was using.

Value

The g6_proxy object.

Examples

```
if (interactive()) {
  library(shiny)
  library(g6R)

  ui <- fluidPage(
    g6_output("graph", height = "500px"),
    actionButton("update_ports", "Update Ports")
  )

  server <- function(input, output, session) {
    output$graph <- render_g6({
      g6(
        nodes = g6_nodes(
          g6_node(
            id = "A",
            type = "custom-rect-node",
            ports = g6_ports(
              g6_input_port(key = "in1", label = "in1", placement = "left"),
              g6_output_port(key = "out1", label = "out1", placement = "right"),
              g6_output_port(key = "out2", label = "out2", placement = c(1, 1))
            ),
            style = list(x = 100, y = 200, labelText = "Node A")
          ),
          g6_node(
            id = "B",
            type = "custom-rect-node",
            ports = g6_ports(
              g6_input_port(key = "in2", label = "in2", placement = "left"),
              g6_output_port(key = "out3", label = "out3", placement = "right"),
              g6_output_port(key = "out4", label = "out4", placement = c(1, 0))
            ),
            style = list(x = 300, y = 200, labelText = "Node B")
          )
        ),
        edges = g6_edges(
       g6_edge(source = "A", target = "B", style = list(sourcePort = "out1", targetPort = "in2"))
        )
      ) |> g6_behaviors(click_select(), drag_element(), drag_canvas())
    })

    observeEvent(input$update_ports, {
      g6_update_ports(
        g6_proxy("graph"),
        list(
          A = list(remove = c("out1", "out2")),
          B = list(
            add = list(g6_port(key = "new", label = "new", placement = "top")),
```

```
            update = list(g6_port(key = "in2", label = "Updated label"))
          )
        )
      )
    })
  }

  shinyApp(ui, server)
}
```

`grid_line` *Configure Grid Line Plugin*

Description

Creates a configuration object for the grid-line plugin in G6. This plugin adds a background grid to the graph canvas.

Usage

```
grid_line(
  key = "grid-line",
  border = TRUE,
  borderLineWidth = 1,
  borderStroke = "#eee",
  borderStyle = "solid",
  follow = FALSE,
  lineWidth = 1,
  size = 20,
  stroke = "#eee",
  ...
)
```

Arguments

`key`	Unique identifier for the plugin (string, default: NULL).
`border`	Whether to display the border (boolean, default: TRUE).
`borderLineWidth`	Border line width (number, default: 1).
`borderStroke`	Border color (string, default: "#eee").
`borderStyle`	Border style (string, default: "solid").
`follow`	Whether the grid follows canvas movements (boolean or list, default: FALSE).
`lineWidth`	Grid line width (number or string, default: 1).
`size`	Grid unit size in pixels (number, default: 20).
`stroke`	Grid line color (string, default: "#eee").
`...`	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/grid-line.

Value

A list with the configuration settings for the grid-line plugin.

Examples

```
# Basic configuration
config <- grid_line()

# Custom configuration
config <- grid_line(
  key = "my-grid",
  border = TRUE,
  borderLineWidth = 2,
  borderStroke = "#ccc",
  borderStyle = "dashed",
  follow = list(
    translate = TRUE,
    zoom = FALSE
  ),
  lineWidth = 0.5,
  size = 30,
  stroke = "#e0e0e0"
)
```

---

history *Configure History Plugin*

Description

Creates a configuration object for the history plugin in G6. This plugin enables undo/redo functionality for graph operations.

Usage

```
history(
  key = "history",
  afterAddCommand = NULL,
  beforeAddCommand = NULL,
  executeCommand = NULL,
  stackSize = 0,
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
afterAddCommand	Callback function called after a command is added to the undo/redo queue (JS function, default: NULL).

beforeAddCommand
: Callback function called before a command is added to the undo/redo queue (JS function, default: NULL).

executeCommand
: Callback function called when executing a command (JS function, default: NULL).

stackSize
: Maximum length of history records to be recorded, 0 means unlimited (number, default: 0).

...
: Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/history.

Value

A list with the configuration settings for the history plugin.

Examples

```
# Basic configuration
config <- history()

# Custom configuration
config <- history(
  key = "my-history",
  stackSize = 50,
  beforeAddCommand = JS("function(cmd, revert) {
    console.log('Before adding command:', cmd);
    // Only allow certain operations to be recorded
    return cmd.method !== 'update';
  }"),
  afterAddCommand = JS("function(cmd, revert) {
    console.log('Command added to ' + (revert ? 'undo' : 'redo') + ' stack');
  }"),
  executeCommand = JS("function(cmd) {
    console.log('Executing command:', cmd);
  }")
)
```

---

hover_activate *Configure Hover Activate Behavior*

Description

Creates a configuration object for the hover-activate behavior in G6. This behavior activates elements when the mouse hovers over them.

Usage

```
hover_activate(
  key = "hover-activate",
  animation = TRUE,
```

```
  enable = TRUE,
  degree = 0,
  direction = c("both", "in", "out"),
  state = "active",
  inactiveState = NULL,
  onHover = NULL,
  onHoverEnd = NULL,
  ...
)
```

Arguments

key	Unique identifier for the behavior (string, default: "hover-activate").
animation	Whether to enable animation (boolean, default: TRUE).
enable	Whether to enable hover feature (boolean or JS function, default: TRUE).
degree	Degree of relationship to activate elements (number or JS function, default: 0).
direction	Specify edge direction: "both", "in", or "out" (string, default: "both").
state	State of activated elements (string, default: "active").
inactiveState	State of inactive elements (string, default: NULL).
onHover	Callback when element is hovered (JS function, default: NULL).
onHoverEnd	Callback when hover ends (JS function, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/hover-activate.

Value

A list with the configuration settings for the hover-activate behavior.

Examples

```
# Basic configuration
config <- hover_activate()

# Custom configuration
config <- hover_activate(
  key = "my-hover-behavior",
  animation = FALSE,
  degree = 1,
  direction = "out",
  state = "highlight",
  inactiveState = "inactive",
  onHover = JS("(event) => { console.log('Hover on:', event.target.id); }")
)
```

`hull` *Configure Hull Plugin*

Description

Creates a configuration object for the hull plugin in G6. This plugin creates a hull (convex or concave) that surrounds specified graph elements.

Usage

```
hull(
  members,
  key = "hull",
  concavity = Inf,
  corner = c("rounded", "smooth", "sharp"),
  padding = 10,
  label = TRUE,
  labelText = NULL,
  labelPlacement = c("bottom", "left", "right", "top", "center"),
  labelBackground = FALSE,
  labelPadding = 0,
  labelCloseToPath = TRUE,
  labelAutoRotate = TRUE,
  labelOffsetX = 0,
  labelOffsetY = 0,
  labelMaxWidth = NULL,
  ...
)
```

Arguments

`members`	Elements within the hull, including nodes and edges (character/numeric vector, required).
`key`	Unique identifier for the plugin (string, default: NULL).
`concavity`	Concavity parameter, larger values create less concave hulls (number, default: Infinity).
`corner`	Corner type: "rounded", "smooth", or "sharp" (string, default: "rounded").
`padding`	Padding around the elements (number, default: 10).
`label`	Whether to display the label (boolean, default: TRUE).
`labelText`	Label text content. Default to NULL.
`labelPlacement`	Label position: "left", "right", "top", "bottom", or "center" (string, default: "bottom").
`labelBackground`	Whether to display the background (boolean, default: FALSE).

labelPadding	Label padding (number or numeric vector, default: 0).
labelCloseToPath	Whether the label is close to the hull (boolean, default: TRUE).
labelAutoRotate	Whether the label rotates with the hull, effective only when closeToPath is true (boolean, default: TRUE).
labelOffsetX	X-axis offset (number, default: 0).
labelOffsetY	Y-axis offset (number, default: 0).
labelMaxWidth	Maximum width of the text, exceeding will be ellipsized (number or NULL, default: NULL).
...	Other options. See https://g6.antv.antgroup.com/en/manual/plugin/hull.

Value

A list with the configuration settings for the hull plugin.

Examples

```
# Basic configuration
config <- hull(members = c("node1", "node2", "node3"))

# Custom configuration for a cluster
config <- hull(
  key = "cluster-hull",
  members = c("node1", "node2", "node3", "node4"),
  concavity = 0.8,
  corner = "smooth",
  padding = 15,
  label = TRUE,
  labelPlacement = "top",
  labelBackground = TRUE,
  labelPadding = c(4, 8),
  labelMaxWidth = 100
)
```

---

is_g6_data *Check if an object is a g6_data object*

Description

Check if an object is a g6_data object

Usage

```
is_g6_data(x)
```

Arguments

x	An object to check.

Value

Logical indicating if x is of class g6_data.

---

is_g6_node *Check if an object is a G6 element*

Description

Check if an object is a G6 element

Usage

```
is_g6_node(x)

is_g6_edge(x)

is_g6_combo(x)
```

Arguments

x	An object of class g6_node, g6_edge, or g6_combo.

Value

Logical indicating if x is of the specified class.

---

is_g6_nodes *Check if an object is a list of G6 elements*

Description

Check if an object is a list of G6 elements

Usage

```
is_g6_nodes(x)

is_g6_edges(x)

is_g6_combos(x)
```

Arguments

x	An object of class g6_nodes, g6_edges, or g6_combos.

is_g6_port *Check if an object is a g6_port*

Description

Check if an object is a g6_port

Usage

```
is_g6_port(x)

is_g6_ports(x)
```

Arguments

x	An object to check.

Value

Logical indicating if x is of class g6_port.

Examples

```
p <- g6_port("input-1", type = "input")
is_g6_port(p)
is_g6_port(list(key = "input-1", type = "input"))
is_g6_port(as_g6_port(list(key = "input-1", type = "input")))
is_g6_ports(g6_ports(
  g6_port("input-1", type = "input"),
  g6_port("output-1", type = "output")
))
```

JS *Marks as string to be processed as a JS function*

Description

Useful for htmlwidgets

Usage

```
JS(...)
```

Arguments

...	Any valid JS element.

Value

A character vector with class "JS_EVAL" that can be used in htmlwidgets to mark is as a JavaScript function.

`lasso_select` *Configure Lasso Select Behavior*

Description

Creates a configuration object for the lasso-select behavior in G6. This behavior allows selecting elements by drawing a lasso around them.

Usage

```
lasso_select(
  key = "lasso-select",
  animation = FALSE,
  enable = TRUE,
  enableElements = "node",
  immediately = FALSE,
  mode = c("default", "union", "intersect", "diff"),
  onSelect = NULL,
  state = "selected",
  style = NULL,
  trigger = c("shift"),
  outputId = NULL,
  ...
)
```

Arguments

`key`	Unique identifier for the behavior (string, default: "lasso-select").
`animation`	Whether to enable animation (boolean, default: FALSE).
`enable`	Whether to enable lasso selection (boolean or JS function, default: TRUE).
`enableElements`	Types of elements that can be selected (character vector, default: c("node", "combo", "edge")).
`immediately`	Whether to select immediately, only effective when selection mode is default (boolean, default: FALSE).
`mode`	Selection mode: "union", "intersect", "diff", or "default" (string, default: "default").
`onSelect`	Callback for selected element state (JS function, default: NULL).
`state`	State to switch to when selected (string, default: "selected").
`style`	Style of the lasso during selection (list, default: NULL).

trigger	Press this shortcut key along with mouse click to select (character vector, default: c("shift")).
outputId	Manually pass the Shiny output ID. This is useful when the graph is initialised outside the shiny render function and the ID cannot be automatically inferred. This allows to set input values from the callback function with the right namespace and graph ID. You must typically pass `session$ns("graphid")` to ensure this also works in modules.
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/lasso-select.

Value

A list with the configuration settings for the lasso-select behavior.

Examples

```
# Basic configuration
config <- lasso_select()

# Custom configuration
config <- lasso_select(
  key = "my-lasso-select",
  animation = TRUE,
  enableElements = c("node", "combo"),
  mode = "union",
  state = "highlight",
  trigger = c("control"),
  style = list(
    stroke = "#1890FF",
    lineWidth = 2,
    fillOpacity = 0.1
  )
)
```

---

legend *Configure Legend Plugin*

Description

Creates a configuration object for the legend plugin in G6. This plugin adds a legend to the graph, allowing users to identify and interact with different categories of elements.

Usage

```
legend(
  key = "legend",
  trigger = c("hover", "click"),
 position = c("bottom", "top", "left", "right", "top-left", "top-right", "bottom-left",
```

```
    "bottom-right"),
  container = NULL,
  className = NULL,
  containerStyle = NULL,
  nodeField = NULL,
  edgeField = NULL,
  comboField = NULL,
  orientation = c("horizontal", "vertical"),
  layout = c("flex", "grid"),
  showTitle = FALSE,
  titleText = "",
  x = NULL,
  y = NULL,
  width = 240,
  height = 160,
  itemSpacing = 4,
  rowPadding = 10,
  colPadding = 10,
  itemMarkerSize = 16,
  itemLabelFontSize = 16,
  gridCol = NULL,
  gridRow = NULL,
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
trigger	How legend items trigger highlighting: "hover" or "click" (string, default: "hover").
position	Relative position of the legend on the canvas (string, default: "bottom").
container	Container to which the legend is mounted (HTML element or string, default: NULL).
className	Legend canvas class name (string, default: NULL).
containerStyle	Style of the legend container (list or JS object, default: NULL).
nodeField	Node classification identifier (string or JS function, default: NULL).
edgeField	Edge classification identifier (string or JS function, default: NULL).
comboField	Combo classification identifier (string or JS function, default: NULL).
orientation	Layout direction: "horizontal" or "vertical" (string, default: "horizontal").
layout	Layout method: "flex" or "grid" (string, default: "flex").
showTitle	Whether to display the title (boolean, default: FALSE).
titleText	Title content (string, default: "").
x	Relative horizontal position (number, default: NULL).
y	Relative vertical position (number, default: NULL).
width	Width of the legend (number, default: 240).

height	Height of the legend (number, default: 160).
itemSpacing	Spacing between text and marker (number, default: 4).
rowPadding	Spacing between rows (number, default: 10).
colPadding	Spacing between columns (number, default: 10).
itemMarkerSize	Size of the legend item marker (number, default: 16).
itemLabelFontSize	Font size of the legend item text (number, default: 16).
gridCol	Maximum number of columns for grid layout (number, default: NULL).
gridRow	Maximum number of rows for grid layout (number, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/legend.

Value

A list with the configuration settings for the legend plugin.

Examples

```
# Basic configuration for node categories
config <- legend(
  nodeField = "category"
)

# Advanced configuration
config <- legend(
  key = "my-legend",
  position = "top-right",
  nodeField = "type",
  edgeField = "relation",
  orientation = "vertical",
  layout = "grid",
  showTitle = TRUE,
  titleText = "Graph Elements",
  width = 300,
  height = 200,
  gridCol = 2,
  containerStyle = list(
    background = "#f9f9f9",
    border = "1px solid #ddd",
    borderRadius = "4px",
    padding = "8px"
  )
)

# Using a function for classification
config <- legend(
  nodeField = JS("(item) => {
    return item.data.importance > 0.5 ? 'Important' : 'Regular';
  }")
)
```

lesmis *Character network from "Les miserables" novel*

Description

A dataset containing Les Misérables characters network, encoding interactions between characters of Victor Hugo's novel. Two characters are connected whenever they appear in the same chapter. This dataset was first created by Donald Knuth as part of the Stanford Graph Base. (https://people.sc.fsu.edu/~jburkardt/datasets/sgb/sgb.html). It contains 77 nodes corresponding to characters of the novel, and 254 edges.

Usage

```
data(lesmis)
```

Format

A list with 2 data frames:

nodes data frame with 77 rows for the nodes. Contains node labels and x/y coordinates.

edges data frame with 254 rows for the edges. Contains souyrce/target and the number of times the interaction happened.

Source

https://networks.skewed.de/net/lesmis

minimap *Configure Minimap Plugin*

Description

Creates a configuration object for the minimap plugin in G6. This plugin adds a minimap/thumbnail view of the entire graph.

Usage

```
minimap(
  key = "minimap",
  className = NULL,
  container = NULL,
  containerStyle = NULL,
  delay = 128,
  filter = NULL,
  maskStyle = NULL,
  padding = 10,
```

```
  position = "right-bottom",
  renderer = NULL,
  shape = "key",
  size = c(240, 160),
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
className	Class name of the thumbnail canvas (string, default: NULL).
container	Container to which the thumbnail is mounted (HTML element or string, default: NULL).
containerStyle	Style of the thumbnail container (list or JS object, default: NULL).
delay	Delay update time in milliseconds for performance optimization (number, default: 128).
filter	Function to filter elements to display in minimap (JS function, default: NULL).
maskStyle	Style of the mask (list or JS object, default: NULL).
padding	Padding around the minimap (number or numeric vector, default: 10).
position	Position of the thumbnail relative to the canvas (string or numeric vector, default: "right-bottom").
renderer	Custom renderer (JS object, default: NULL).
shape	Method for generating element thumbnails (string or JS function, default: "key").
size	Width and height of the minimap `[width, height]` (numeric vector, default: c(240, 160)).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/minimap.

Value

A list with the configuration settings for the minimap plugin.

Examples

```
# Basic configuration
config <- minimap()

# Custom configuration
config <- minimap(
  key = "my-minimap",
  position = "left-top",
  size = c(200, 150),
  padding = 15,
  containerStyle = list(
    border = "1px solid #ddd",
    borderRadius = "4px",
```

```
    boxShadow = "0 0 8px rgba(0,0,0,0.1)"
  ),
  maskStyle = list(
    stroke = "#1890ff",
    strokeWidth = 2,
    fill = "rgba(24, 144, 255, 0.1)"
  )
)

# With custom filtering function
config <- minimap(
  filter = JS("(id, elementType) => {
    // Only show nodes and important edges in the minimap
    if (elementType === 'node') return true;
    if (elementType === 'edge') {
      // Assuming edges have an 'important' attribute
      const edge = graph.findById(id);
      return edge.getModel().important === true;
    }
    return false;
  }")
)
```

---

node_options *Create Node Options Configuration for G6 Graphs*

Description

Configures the general options for nodes in a G6 graph. These settings control the type, style, state, palette, and animation of nodes.

Usage

```
node_options(
  type = c("circle", "rect", "ellipse", "diamond", "triangle", "star", "image",
    "modelRect"),
  style = node_style_options(),
  state = NULL,
  palette = NULL,
  animation = NULL
)
```

Arguments

type	Node type. Can be a built-in node type name or a custom node name. Built-in types include "circle", "rect", "ellipse", "diamond", "triangle", etc. Default: "circle".
style	Node style configuration. Controls the appearance of nodes including color, size, border, etc. Can be created with node_style_options(). Default: NULL.

`state`	Defines the style of the node in different states, such as hover, selected, disabled, etc. Should be a list mapping state names to style configurations. Default: NULL.
`palette`	Defines the color palette of the node, used to map colors based on different data. Default: NULL.
`animation`	Defines the animation effect of the node. Can be created with `animation_config()`. Default: NULL.

Details

Node options allow defining how nodes appear and behave in a G6 graph. This includes selecting node types, setting styles, configuring state-based appearances, defining color palettes, and specifying animation effects.

Value

A list containing node options configuration that can be passed to `g6_options()`.

Examples

```
# Basic node options with default circle type
options <- node_options()

# Rectangle node with custom style
options <- node_options(
  type = "rect",
  style = node_style_options(
    fill = "#E8F7FF",
    stroke = "#1783FF",
    lineWidth = 2
  )
)
```

---

node_style_options *Create Node Style Options for G6 Graphs*

Description

Configures the styling options for nodes in a G6 graph. These settings control the appearance and interaction behavior of nodes. Used in node_options.

Usage

```
node_style_options(
  collapsed = FALSE,
  cursor = "default",
  fill = "#1783FF",
  fillOpacity = 1,
```

```
  increasedLineWidthForHitTesting = 0,
  lineCap = c("butt", "round", "square"),
  lineDash = NULL,
  lineDashOffset = NULL,
  lineJoin = c("miter", "round", "bevel"),
  lineWidth = 1,
  opacity = 1,
  shadowBlur = NULL,
  shadowColor = NULL,
  shadowOffsetX = NULL,
  shadowOffsetY = NULL,
  shadowType = c("outer", "inner"),
  size = 32,
  stroke = "#000",
  strokeOpacity = 1,
  transform = NULL,
  transformOrigin = NULL,
  visibility = c("visible", "hidden"),
  x = NULL,
  y = NULL,
  z = NULL,
  zIndex = NULL,
  ...
)
```

Arguments

collapsed	Whether the current node/group is collapsed. Default: FALSE.
cursor	Node mouse hover cursor style. Common values include "default", "pointer", "move", etc. Default: "default".
fill	Node fill color. Can be any valid CSS color value. Default: "#1783FF".
fillOpacity	Node fill color opacity. Value between 0 and 1. Default: 1.
increasedLineWidthForHitTesting	When lineWidth is small, this value increases the interactive area to make "thin lines" easier to interact with. Default: 0.
lineCap	Node stroke end style. Options: "round", "square", "butt". Default: "butt".
lineDash	Node stroke dash style. Vector of numbers specifying dash pattern.
lineDashOffset	Node stroke dash offset. Default: NULL.
lineJoin	Node stroke join style. Options: "round", "bevel", "miter". Default: "miter".
lineWidth	Node stroke width. Default: 1.
opacity	Node overall opacity. Value between 0 and 1. Default: 1.
shadowBlur	Node shadow blur amount. Default: NULL.
shadowColor	Node shadow color. Default: NULL.
shadowOffsetX	Node shadow offset in the x-axis direction. Default: NULL.
shadowOffsetY	Node shadow offset in the y-axis direction. Default: NULL.

shadowType	Node shadow type. Options: "inner", "outer". Default: "outer".
size	Node size. Can be a single number for equal width/height or a vector of two numbers [width, height]. Default: 32.
stroke	Node stroke (border) color. Default: "#000".
strokeOpacity	Node stroke color opacity. Value between 0 and 1. Default: 1.
transform	CSS transform attribute to rotate, scale, skew, or translate the node. Default: NULL.
transformOrigin	Rotation and scaling center point. Default: NULL.
visibility	Whether the node is visible. Options: "visible", "hidden". Default: "visible".
x	Node x coordinate. Default: 0.
y	Node y coordinate. Default: 0.
z	Node z coordinate (for 3D). Default: 0.
zIndex	Node rendering level (for layering). Default: 0.
...	Other parameters.

Details

Node style options allow fine-grained control over how nodes are rendered and behave in a G6 graph. This includes colors, sizes, borders, shadows, visibility, positioning, and interaction states.

Value

A list containing node style options that can be passed to node_options().

Examples

```
# Basic node style with blue fill and red border
styles <- node_style_options(
  fill = "#1783FF",
  stroke = "#FF0000",
  lineWidth = 2
)

# Create a node with shadow effects
styles <- node_style_options(
  fill = "#FFFFFF",
  stroke = "#333333",
  lineWidth = 1,
  shadowBlur = 10,
  shadowColor = "rgba(0,0,0,0.3)",
  shadowOffsetX = 5,
  shadowOffsetY = 5
)

# Custom sized node with dashed border
styles <- node_style_options(
  size = c(100, 50),
```

```
  fill = "#E8F7FF",
  stroke = "#1783FF",
  lineDash = c(5, 5),
  opacity = 0.8
)
```

`optimize_viewport_transform` *Configure Optimize Viewport Transform Behavior*

Description

Creates a configuration object for the optimize-viewport-transform behavior in G6. This behavior improves performance during viewport transformations by temporarily hiding certain elements.

Usage

```
optimize_viewport_transform(
  key = "optimize-viewport-transform",
  enable = TRUE,
  debounce = 200,
  shapes = JS("(type) => type === 'node'"),
  ...
)
```

Arguments

`key`	Unique identifier for the behavior (string, default: "optimize-viewport-transform").
`enable`	Whether to enable this behavior (boolean or JS function, default: TRUE).
`debounce`	How long after the operation ends to restore the visibility of all elements in milliseconds (number, default: 200).
`shapes`	Function to specify which graphical elements should remain visible during canvas operations (JS function, default: returns TRUE for nodes).
`...`	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/optimize-viewport-transform.

Value

A list with the configuration settings for the optimize-viewport-transform behavior.

Examples

```
# Basic configuration
config <- optimize_viewport_transform()

# Custom configuration
config <- optimize_viewport_transform(
```

```
  key = "my-optimize-transform",
  debounce = 500,
  shapes = JS("(type) => type === 'node' || type === 'edge'")
)

# With conditional enabling
config <- optimize_viewport_transform(
  enable = JS("(event) => event.getCurrentTransform().zoom < 0.5")
)
```

---

poke *Example pokemon data*

Description

Pokemon evolution network to showcase combo features.

Usage

```
data(poke)
```

Format

A list with 3 nested lists:

nodes list with 120 pokemon.

edges list with 69 connections.

combo list with 51 pokemon families. A family contains all evolutions of a pokemon.

Source

https://pokeapi.co/docs/v2

---

radial *Example Network for radial layouts*

Description

Example Network for radial layouts

Usage

```
data(radial)
```

Format

A list with 2 data frames:

nodes data frame with 34 rows for the nodes.

edges data frame with 58 rows for the edges.

Source

https://assets.antv.antgroup.com/g6/radial.json

`radial_layout` *Generate G6 Radial layout configuration*

Description

This function creates a configuration list for G6 Radial layout with all available options as parameters.

Usage

```
radial_layout(
  center = NULL,
  focusNode = NULL,
  height = NULL,
  width = NULL,
  nodeSize = NULL,
  nodeSpacing = 10,
  linkDistance = 50,
  unitRadius = 100,
  maxIteration = 1000,
  maxPreventOverlapIteration = 200,
  preventOverlap = FALSE,
  sortBy = NULL,
  sortStrength = 10,
  strictRadial = TRUE,
  ...
)
```

Arguments

`center`	Numeric vector of length 2. Center coordinates.
`focusNode`	Character, list (Node), or NULL. Radiating center node. Defaults to NULL.
`height`	Numeric. Canvas height.
`width`	Numeric. Canvas width.
`nodeSize`	Numeric. Node size (diameter).

nodeSpacing	Numeric or function. Minimum node spacing (effective when preventing overlap). Defaults to 10.
linkDistance	Numeric. Edge length. Defaults to 50.
unitRadius	Numeric or NULL. Radius per circle. Defaults to 100.
maxIteration	Numeric. Maximum number of iterations. Defaults to 1000.
maxPreventOverlapIteration	Numeric. Max iterations for overlap prevention. Defaults to 200.
preventOverlap	Logical. Whether to prevent node overlap. Defaults to FALSE.
sortBy	Character. Field for sorting nodes in the same layer.
sortStrength	Numeric. Sorting strength for nodes in the same layer. Defaults to 10.
strictRadial	Logical. Strictly place nodes in the same layer on the same ring. Defaults to TRUE.
...	Additional parameters to pass to the layout. See https://g6.antv.antgroup.com/en/manual/layout/radial-layout.

Value

A list containing the configuration for G6 radial layout.

Examples

```
if (interactive()) {
  g6(jsonUrl = "https://assets.antv.antgroup.com/g6/radial.json") |>
    g6_layout(
      radial_layout(
        unitRadius = 100,
        linkDistance = 200
      )
    ) |>
    g6_behaviors(
      "zoom-canvas",
      drag_element()
    )
}
```

---

scroll_canvas *Configure Scroll Canvas Behavior*

Description

Creates a configuration object for the scroll-canvas behavior in G6. This behavior allows scrolling the canvas with mouse wheel or keyboard.

Usage

```
scroll_canvas(
  key = "scroll-canvas",
  enable = TRUE,
  direction = NULL,
  range = 1,
  sensitivity = 1,
  trigger = NULL,
  onFinish = NULL,
  preventDefault = TRUE,
  ...
)
```

Arguments

key	Unique identifier for the behavior (string, default: "scroll-canvas").
enable	Whether to enable this behavior (boolean or JS function, default: TRUE).
direction	Allowed scrolling direction: "x", "y", or NULL for no limit (string or NULL, default: NULL).
range	Scrollable viewport range in viewport size units (numeric or numeric vector, default: 1).
sensitivity	Scrolling sensitivity, the larger the value, the faster the scrolling (numeric, default: 1).
trigger	Keyboard shortcuts to trigger scrolling (list, default: NULL).
onFinish	Callback function when scrolling is finished (JS function, default: NULL).
preventDefault	Whether to prevent the browser's default event (boolean, default: TRUE).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/scroll-canvas.

Value

A list with the configuration settings for the scroll-canvas behavior.

Examples

```
# Basic configuration
config <- scroll_canvas()

# Custom configuration
config <- scroll_canvas(
  key = "my-scroll-behavior",
  direction = "x",
  range = c(-2, 2),
  sensitivity = 1.5,
  preventDefault = FALSE
)
```

```
# With keyboard triggers and callback
config <- scroll_canvas(
  enable = JS("(event) => !event.altKey"),
  trigger = list(
    up = "w",
    down = "s",
    left = "a",
    right = "d"
  ),
  onFinish = JS("() => { console.log('Scrolling finished'); }")
)
```

`set_g6_max_collapse_depth` *Set max collapse depth*

Description

Controls which nodes display a collapse button based on their depth in the graph. Only nodes at depth <= `maxCollapseDepth` will show collapse buttons. Set to `Inf` (the default) to allow all nodes with children to be collapsible. Set to `0` to only allow root nodes to collapse. Set to `-1` to disable collapsing entirely.

Usage

```
set_g6_max_collapse_depth(val)
```

Arguments

`val`	A single number >= -1. Use `Inf` for no limit, `-1` to disable all collapsing.

Value

Invisibly returns `val`.

`snapline` *Configure Snapline Plugin*

Description

Creates a configuration object for the snapline plugin in G6. This plugin provides alignment guidelines when moving nodes.

Usage

```
snapline(
  key = "snapline",
  tolerance = 5,
  offset = 20,
  autoSnap = TRUE,
  shape = "key",
  verticalLineStyle = list(stroke = "#1783FF"),
  horizontalLineStyle = list(stroke = "#1783FF"),
  filter = NULL,
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
tolerance	The alignment accuracy in pixels (number, default: 5).
offset	The extension distance of the snapline (number, default: 20).
autoSnap	Whether to enable automatic snapping (boolean, default: TRUE).
shape	Specifies which shape to use as reference: "key" or a function (string or JS function, default: "key").
verticalLineStyle	Vertical snapline style (list or JS object, default: list(stroke = "#1783FF")).
horizontalLineStyle	Horizontal snapline style (list or JS object, default: list(stroke = "#1783FF")).
filter	Function to filter nodes that don't participate in alignment (JS function, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/snapline.

Value

A list with the configuration settings for the snapline plugin.

Examples

```
# Basic configuration
config <- snapline()

# Custom configuration
config <- snapline(
  key = "my-snapline",
  tolerance = 8,
  offset = 30,
  verticalLineStyle = list(
    stroke = "#f00",
    strokeWidth = 1.5,
```

```
    lineDash = c(5, 2)
  ),
  horizontalLineStyle = list(
    stroke = "#00f",
    strokeWidth = 1.5,
    lineDash = c(5, 2)
  )
)

# With custom filter function
config <- snapline(
  filter = JS("(node) => {
    // Only allow regular nodes to participate in alignment
    // Exclude special nodes like 'start' or 'end'
    const model = node.getModel();
    return model.type !== 'start' && model.type !== 'end';
  }")
)
```

`timebar` *Configure Timebar Plugin*

Description

Creates a configuration object for the timebar plugin in G6. This plugin adds a timeline or chart-based control for time-related data visualization.

Usage

```
timebar(
  data,
  key = "timebar",
  className = "g6-timebar",
  x = NULL,
  y = NULL,
  width = 450,
  height = 60,
  position = c("bottom", "top"),
  padding = 10,
  timebarType = c("time", "chart"),
  elementTypes = c("node", "edge", "combo"),
  mode = c("modify", "visibility"),
  values = NULL,
  loop = FALSE,
  getTime = NULL,
  labelFormatter = NULL,
  onChange = NULL,
  onReset = NULL,
```

```
  onSpeedChange = NULL,
  onPlay = NULL,
  onPause = NULL,
  onBackward = NULL,
  onForward = NULL,
  ...
)
```

Arguments

data	Time data, either a vector of timestamps or a list of objects with time and value (required).
key	Unique identifier for the plugin (string, default: NULL).
className	Additional class name for the timebar DOM (string, default: "g6-timebar").
x	X position, will be ignored if position is set (number, default: NULL).
y	Y position, will be ignored if position is set (number, default: NULL).
width	Timebar width (number, default: 450).
height	Timebar height (number, default: 60).
position	Timebar position: "bottom" or "top" (string, default: "bottom").
padding	Padding around the timebar (number or numeric vector, default: 10).
timebarType	Display type: "time" or "chart" (string, default: "time").
elementTypes	Filter element types: vector of "node", "edge", and/or "combo" (character vector, default: c("node")).
mode	Control element filtering method: "modify" or "visibility" (string, default: "modify").
values	Current time value (number, vector of two numbers, Date, or vector of two Dates, default: NULL).
loop	Whether to loop playback (boolean, default: FALSE).
getTime	Method to get element time (JS function, default: NULL).
labelFormatter	Custom time formatting in chart mode (JS function, default: NULL).
onChange	Callback when time interval changes (JS function, default: NULL).
onReset	Callback when reset (JS function, default: NULL).
onSpeedChange	Callback when playback speed changes (JS function, default: NULL).
onPlay	Callback when playback starts (JS function, default: NULL).
onPause	Callback when paused (JS function, default: NULL).
onBackward	Callback when moving backward (JS function, default: NULL).
onForward	Callback when moving forward (JS function, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/timebar.

Value

A list with the configuration settings for the timebar plugin.

Examples

```
# Basic timebar with array of timestamps
config <- timebar(
  data = c(1609459200000, 1609545600000, 1609632000000)  # Jan 1-3, 2021 in milliseconds
)

# Chart-type timebar with time-value pairs
config <- timebar(
  data = list(
    list(time = 1609459200000, value = 10),
    list(time = 1609545600000, value = 25),
    list(time = 1609632000000, value = 15)
  ),
  timebarType = "chart",
  width = 600,
  height = 100,
  position = "top"
)

# With custom callbacks
config <- timebar(
  data = c(1609459200000, 1609545600000, 1609632000000),
  onChange = JS("(values) => {
    console.log('Time changed:', values);
  }"),
  onPlay = JS("() => {
    console.log('Playback started');
  }")
)

# With custom time getter function for elements
config <- timebar(
  data = c(1609459200000, 1609545600000, 1609632000000),
  getTime = JS("(datum) => {
    return datum.created_at; // Get time from created_at property
  }")
)
```

---

toolbar *Configure Toolbar Plugin*

Description

Creates a configuration object for the toolbar plugin in G6. This plugin adds a customizable toolbar with items for graph operations.

Usage

```
toolbar(
```

```
  getItems = NULL,
  key = "toolbar",
  className = NULL,
  position = c("top-left", "top", "top-right", "right", "bottom-right", "bottom",
    "bottom-left", "left"),
  style = NULL,
  onClick = NULL,
  ...
)
```

Arguments

getItems	Function that returns the list of toolbar items (JS function, required).
key	Unique identifier for the plugin (string, default: NULL).
className	Additional CSS class name for the toolbar DOM element (string, default: NULL).
position	Toolbar position relative to the canvas (string, default: "top-left").
style	Custom style for the toolbar DOM element (list or JS object, default: NULL).
onClick	Callback function after a toolbar item is clicked (JS function, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/toolbar.

Value

A list with the configuration settings for the toolbar plugin.

Examples

```
# Basic toolbar with zoom controls
config <- toolbar(
  position = "top-right",
  getItems = JS("() => [
    { id: 'zoom-in', value: 'zoom-in' },
    { id: 'zoom-out', value: 'zoom-out' },
    { id: 'undo', value: 'undo' },
    { id: 'redo', value: 'redo' },
    { id: 'auto-fit', value: 'fit' }
  ]"),
  onClick = JS("(value) => {
    // redo, undo need to be used with the history plugin
    const history = graph.getPluginInstance('history');
    switch (value) {
      case 'zoom-in':
        graph.zoomTo(1.1);
        break;
      case 'zoom-out':
        graph.zoomTo(0.9);
        break;
      case 'undo':
        history?.undo();
```

```
          break;
        case 'redo':
          history?.redo();
          break;
        case 'fit':
          graph.fitView();
          break;
        default:
          break;
      }
    }")
)
```

---

tooltips *Configure Tooltip Plugin*

Description

Creates a configuration object for the tooltip plugin in G6. This plugin displays tooltips when interacting with graph elements.

Usage

```
tooltips(
  key = "tooltip",
  position = c("top-right", "top", "bottom", "left", "right", "top-left", "bottom-left",
    "bottom-right"),
  enable = TRUE,
  getContent = NULL,
  onOpenChange = NULL,
  trigger = c("hover", "click"),
  container = NULL,
  offset = c(10, 10),
  enterable = FALSE,
  title = NULL,
  style = NULL,
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
position	Tooltip position relative to the element (string, default: "top-right").
enable	Whether the plugin is enabled (boolean or function, default: TRUE).
getContent	Function to generate custom tooltip content (JS function, default: NULL).
onOpenChange	Callback for tooltip show/hide events (JS function, default: NULL).
trigger	Trigger behavior: "hover" or "click" (string, default: "hover").

container	Custom rendering container for tooltip (string or HTML element, default: NULL).
offset	Offset distance as a vector of two numbers [x, y] (numeric vector, default: c(10, 10)).
enterable	Whether the pointer can enter the tooltip (boolean, default: FALSE).
title	Title for the tooltip (string, default: NULL).
style	Custom style for the tooltip (list or JS object, default: list(".tooltip" = list(visibility = "hidden"))).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/tooltip.

Value

A list with the configuration settings for the tooltip plugin.

Examples

```
# Basic tooltip
config <- tooltips()

# Tooltip with custom position and content
config <- tooltips(
  position = "bottom",
  getContent = JS("(event, items) => {
    let result = `<h4>Custom Content</h4>`;
    items.forEach((item) => {
      result += `<p>Type: ${item.data.description}</p>`;
    });
    return result;
  }")
)

# Click-triggered tooltip with custom style
config <- tooltips(
  trigger = "click",
  position = "bottom-left",
  offset = c(15, 20),
  style = list(
    ".tooltip" = list(
      backgroundColor = "#fff",
      border = "1px solid #ccc",
      borderRadius = "4px",
      boxShadow = "0 2px 6px rgba(0,0,0,0.1)",
      padding = "10px",
      maxWidth = "300px"
    )
  )
)

# Conditional tooltip based on node type
config <- tooltips(
```

```
  enable = JS("(event, items) => {
    // Only show tooltip for nodes with type 'important'
    const item = items[0];
    return item.type === 'important';
  }"),
  enterable = TRUE,
  onOpenChange = JS("(open) => {
    console.log('Tooltip visibility changed:', open);
  }")
)
```

`tree` *Example tree graph*

Description

The graph contains a classification of algorithm categories

Usage

```
data(tree)
```

Format

A list with 2 data frames:

nodes data frame with 31 rows for the nodes.

edges data frame with 30 rows for the edges.

Source

https://gw.alipayobjects.com/os/antvdemo/assets/data/algorithm-category.json

---

`validate_edges_ports` *Validate that all edges connect input/output ports*

Description

Validate that all edges connect input/output ports

Usage

```
validate_edges_ports(edges, nodes)
```

Arguments

`edges`	List of g6_edge objects.
`nodes`	List of g6_node objects.

Value

Invisibly TRUE if all edges are valid, otherwise stops with an error.

---

validate_port *Validate a single G6 port*

Description

Validate a single G6 port

Usage

```
validate_port(x, ...)

## S3 method for class 'g6_port'
validate_port(x, ...)
```

Arguments

x	An object of class 'g6_port'.
...	Generic consistency.

Value

The validated port (invisibly).

Examples

```
validate_port(g6_port("input-1", type = "input"))
```

---

validate_ports *Validate a list of G6 ports*

Description

Validate a list of G6 ports

Usage

```
validate_ports(x, ...)

## S3 method for class 'list'
validate_ports(x, ...)

## S3 method for class 'g6_ports'
validate_ports(x, ...)
```

Arguments

x	A list of g6_port objects.
...	Generic consistency.

Value

The validated list (invisibly).

Examples

```
validate_ports(list(
  g6_port("input-1", type = "input"),
  g6_port("output-1", type = "output")
))
```

---

watermark *Configure Watermark Plugin*

Description

Creates a configuration object for the watermark plugin in G6. This plugin adds a watermark to the graph canvas.

Usage

```
watermark(
  key = "watermark",
  width = 200,
  height = 100,
  opacity = 0.2,
  rotate = pi/12,
  imageURL = NULL,
  text = NULL,
  textFill = "#000",
  textFontSize = 16,
  textFontFamily = NULL,
  textFontWeight = NULL,
  textFontVariant = NULL,
  textAlign = c("center", "end", "left", "right", "start"),
 textBaseline = c("alphabetic", "bottom", "hanging", "ideographic", "middle", "top"),
  backgroundRepeat = "repeat",
  backgroundAttachment = NULL,
  backgroundBlendMode = NULL,
  backgroundClip = NULL,
  backgroundColor = NULL,
  backgroundImage = NULL,
  backgroundOrigin = NULL,
```

```
  backgroundPosition = NULL,
  backgroundPositionX = NULL,
  backgroundPositionY = NULL,
  backgroundSize = NULL,
  ...
)
```

Arguments

key	Unique identifier for the plugin (string, default: NULL).
width	Width of a single watermark (number, default: 200).
height	Height of a single watermark (number, default: 100).
opacity	Opacity of the watermark (number, default: 0.2).
rotate	Rotation angle of the watermark in radians (number, default: pi/12).
imageURL	Image watermark URL, higher priority than text watermark (string, default: NULL).
text	Watermark text content (string, default: NULL).
textFill	Color of the text watermark (string, default: "#000").
textFontSize	Font size of the text watermark (number, default: 16).
textFontFamily	Font of the text watermark (string, default: NULL).
textFontWeight	Font weight of the text watermark (string, default: NULL).
textFontVariant	Font variant of the text watermark (string, default: NULL).
textAlign	Text alignment of the watermark (string, default: "center").
textBaseline	Baseline alignment of the text watermark (string, default: "middle").
backgroundRepeat	Repeat mode of the watermark (string, default: "repeat").
backgroundAttachment	Background attachment behavior of the watermark (string, default: NULL).
backgroundBlendMode	Background blend mode of the watermark (string, default: NULL).
backgroundClip	Background clip of the watermark (string, default: NULL).
backgroundColor	Background color of the watermark (string, default: NULL).
backgroundImage	Background image of the watermark (string, default: NULL).
backgroundOrigin	Background origin of the watermark (string, default: NULL).
backgroundPosition	Background position of the watermark (string, default: NULL).
backgroundPositionX	Horizontal position of the watermark background (string, default: NULL).

backgroundPositionY
: Vertical position of the watermark background (string, default: NULL).

backgroundSize
: Background size of the watermark (string, default: NULL).

...
: Extra parameters. See https://g6.antv.antgroup.com/en/manual/plugin/watermark.

Value

A list with the configuration settings for the watermark plugin.

Examples

```
# Basic text watermark
config <- watermark(
  text = "G6 Graph",
  opacity = 0.1
)

# Image watermark
config <- watermark(
 imageURL = "https://gw.alipayobjects.com/os/s/prod/antv/assets/image/logo-with-text-73b8a.svg",
  width = 150,
  height = 75,
  opacity = 0.15,
  rotate = 0
)

# Customized text watermark
config <- watermark(
  text = "CONFIDENTIAL",
  textFill = "#ff0000",
  textFontSize = 24,
  textFontWeight = "bold",
  opacity = 0.08,
  rotate = pi/6,
  backgroundRepeat = "repeat-x"
)

# Watermark with background styling
config <- watermark(
  text = "Draft Document",
  textFill = "#333",
  backgroundColor = "#f9f9f9",
  backgroundClip = "content-box",
  backgroundSize = "cover"
)
```

zoom_canvas *Configure Zoom Canvas Behavior*

Description

Creates a configuration object for the zoom-canvas behavior in G6. This behavior allows zooming the canvas with mouse wheel or keyboard shortcuts.

Usage

```
zoom_canvas(
  key = "zoom-canvas",
  animation = list(duration = 200),
  enable = TRUE,
  origin = NULL,
  onFinish = NULL,
  preventDefault = TRUE,
  sensitivity = 1,
  trigger = NULL,
  ...
)
```

Arguments

key	Unique identifier for the behavior (string, default: "zoom-canvas").
animation	Zoom animation effect settings (list, default: list with duration 200ms).
enable	Whether to enable this behavior (boolean or JS function, default: TRUE).
origin	Zoom center point in viewport coordinates (list with x, y values, default: NULL).
onFinish	Callback function when zooming is finished (JS function, default: NULL).
preventDefault	Whether to prevent the browser's default event (boolean, default: TRUE).
sensitivity	Zoom sensitivity, the larger the value, the faster the zoom (numeric, default: 1).
trigger	How to trigger zooming, supports mouse wheel and keyboard shortcuts (list, default: NULL).
...	Extra parameters. See https://g6.antv.antgroup.com/en/manual/behavior/zoom-canvas.

Value

A list with the configuration settings for the zoom-canvas behavior.

Examples

```
# Basic configuration
config <- zoom_canvas()

# Custom configuration
config <- zoom_canvas(
  key = "my-zoom-behavior",
  animation = list(duration = 300, easing = "ease-in-out"),
  origin = list(x = 0, y = 0),
  sensitivity = 1.5,
  preventDefault = FALSE
)

# With keyboard triggers and callback
config <- zoom_canvas(
  enable = JS("(event) => !event.altKey"),
  trigger = list(
    zoomIn = "+",
    zoomOut = "-",
    reset = "0"
  ),
  onFinish = JS("() => { console.log('Zooming finished'); }")
)
```

Index