Package: g2r (via r-universe)

September 1, 2024

Title Interactive Grammar of Graphics

Version 1.0.1

Description Interactive grammar of graphics.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Imports R6, purrr, rlang, tibble, magrittr, jsonlite, crosstalk, htmltools, htmlwidgets

Suggests sf, zoo, covr, maps, Rtsne, alter, broom, shiny, packer, igraph, pkgdown, survival, quantmod, forecast, contoureR, geojsonio, yardstick, tidyquant, rmapshaper, testthat (>= 3.0.0)

Remotes devOpifex/alter

Config/testthat/edition 3

URL `https://g2r.opifex.org/`, `https://github.com/devOpifex/g2r/`

BugReports `https://github.com/devOpifex/g2r/issues/`

Repository https://devopifex.r-universe.dev

RemoteUrl https://github.com/devOpifex/g2r

RemoteRef HEAD

RemoteSha 2cfcb5d38f7d1556bfa2f0009195809d312e2df0

Contents

action_filter_data 3
action_select_data 4
action_toggle_visibility 5
adjust 6

aka 7
animation 7
asp 9
axis 10
axis_title 12
capture_event 12
change_data 13
config 14
coord 15
crosstalk_select 16
elements 16
figures 18
fig_area 19
fig_bin 20
fig_boxplot 21
fig_candle 22
fig_contour 23
fig_density 25
fig_edge 26
fig_error 26
fig_heatmap 27
fig_histogram 28
fig_line 29
fig_map 30
fig_path 30
fig_pie 31
fig_point 32
fig_polygon 33
fig_range 33
fig_ribbon 34
fig_rug 35
fig_schema 36
fig_segment 37
fig_smooth 37
fig_voronoi 38
fig_waffle 39
g2 40
g2r-shiny 41
g2_action 42
g2_proxy 42
gauge 43
gaugeViews 45
global_digits 47
info 47
input_button 49
input_select 49
input_slider 50
interplay 50

layout_arc 52
layout_igraph 53
map_data 53
motif 54
new_animation 56
palettes 56
planes 59
qg2 60
radio 60
remove_figure 61
render 62
scrollbar 62
slider 63
state 63
subject 64
template 64
tooltip 65
to_g2r 66

Index **67**

`action_filter_data` *Filter Data*

Description

Filter data.

Usage

```
action_filter_data(g, input, asp, operator = ">")
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`input`	The `id` of the input that triggers the filter, either `input_select()` or `input_slider()`.
`asp`	Aspect (column) to filter.
`operator`	Operator of the filter, this is combined with the value from the `input` and the `asp` to form a filter statement with the following template; `asp operator inputValue`. For instance, a filter on input `id` = `"theFilter"` on the column speed (of the `cars` dataset) with the operator > (greater than) will create the following filter statement: `speed` > `inputValue`

Examples

```
# works in Rmarkdown
input_slider(
  "yFilter",
  "Filter Y >",
  value = 50,
  min = 40,
  max = 70,
  step = 5
)

## Not run:
g2(cars, asp(speed, dist)) %>%
  fig_point() %>%
  fig_smooth() %>%
  action_filter_data(
    "yFilter",
    dist,
    operator = ">"
  )

## End(Not run)
```

action_select_data *Select Data*

Description

Select a dataset with an `input_select()`.

Usage

```
action_select_data(g, input, datasets)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`input`	Id of the `input_select()` used to choose the dataset.
`datasets`	A key value pair `list` where the key is the name of the dataset as listed in the choices of the `input_select()`.

Examples

```
# works in Rmarkdown
input_select(
  "selector",
  "Select a dataset",
  c("Cars", "More Cars")
)

cars1 <- cars
cars2 <- cars + c(1, -4)

g2(cars, asp(dist, speed)) %>%
  fig_point() %>%
  action_select_data(
    "selector",
    datasets = list(
      "Cars" = cars1,
      "More Cars" = cars2
    )
  )
```

`action_toggle_visibility` *Toggle Visibility*

Description

Toggle the visibily of a chart.

Usage

```
action_toggle_visibility(g, btn)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`btn`	Id of the input_button that toggles the visibility.

Examples

```
# works in Rmarkdown
input_button("toggle", "Show/hide chart")

g2(mtcars, asp(qsec, mpg)) %>%
  fig_point() %>%
  action_toggle_visibility("toggle")
```

`adjust` *Adjust*

Description

Adjust a figure.

Usage

```
adjust(type, margin = NULL, dodge_by = NULL)
```

Arguments

`type`	A vector of types of adjustement to apply to the figure, see the "types" section below for valid values.
`margin`	Margin, between `0` and `1`.
`dodge_by`	Bare column name to use as group for dodge.

Types

Valid values for the `type` argument.

- `stack`
- `dodge`
- `jitter`
- `symmetric`

Examples

```
df <- data.frame(
  x = c(letters, letters),
  y = runif(52),
  grp = rep(c("A", "Z"), each = 2)
)

g2(df, asp(x, y, color = grp)) %>%
  fig_interval(adjust("stack"))
```

aka *Aliases*

Description

Aliases an aspect, this changes the name of the aspects when displayed in axis titles, tooltips, labels, and other places.

Usage

```
aka(g, asp, alias)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`asp`	Bare name of aspect to alias.
`alias`	A string defining the alias.

Examples

```
# see tooltip
g2(cars, asp(speed, dist)) %>%
  fig_point() %>%
  aka(dist, "SO FAR")
```

animation *Animations*

Description

Helper function to build animations.

Methods

Public methods:

- `Animation$enter()`
- `Animation$leave()`
- `Animation$appear()`
- `Animation$update()`
- `Animation$print()`
- `Animation$retrieve()`
- `Animation$clone()`

Method `enter()`:

Usage:

```
Animation$enter(animation = NULL, easing = NULL, delay = NULL, duration = NULL)
```

Arguments:

`animation` Name of animation; `clipIn`, `zoomIn`, `pathIn`, `scaleInY`, `scaleInX`, `fanIn`, or `fadeIn`.

`easing` Name of easing function.

`delay, duration` Delay and duration in milliseconds.

Details: Animation to use on enter

Method `leave()`**:**

Usage:

```
Animation$leave(animation = NULL, easing = NULL, delay = NULL, duration = NULL)
```

Arguments:

`animation` Name of animation; `lineWidthOut`, `zoomOut`, `pathOut`, or `fadeOut`.

`easing` Name of easing function.

`delay, duration` Delay and duration in milliseconds.

Details: Animation to use on leave

Method `appear()`**:**

Usage:

```
Animation$appear(
  animation = NULL,
  easing = NULL,
  delay = NULL,
  duration = NULL
)
```

Arguments:

`animation` Name of animation; `clipIn`, `zoomIn`, `pathIn`, `scaleInY`, `scaleInX`, `fanIn`, or `fadeIn`.

`easing` Name of easing function.

`delay, duration` Delay and duration in milliseconds.

Details: Animation to use on appear

Method `update()`**:**

Usage:

```
Animation$update(
  animation = NULL,
  easing = NULL,
  delay = NULL,
  duration = NULL
)
```

Arguments:

`animation` Name of animation; `fadeIn`, or `fanIn`.

`easing` Name of easing function.

`delay, duration` Delay and duration in milliseconds.

Details: Animation to use on appear

Method `print()`**:**

Usage:

```
Animation$print()
```

Details: Print

Method `retrieve()`**:**

Usage:

```
Animation$retrieve()
```

Details: Retrieve the animation list

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# create animation
anim <- Animation$
  new()$
  appear(
  duration = 2000,
  delay = 500
)

iris %>%
  g2(asp(Sepal.Length, Sepal.Width, color = Species)) %>%
  fig_point(anim)
```

`asp` *Aspects*

Description

Define aspects of a visualisation.

Usage

```
asp(x, y, ...)
```

Arguments

`x, y`	Defines values to map to cartesian coordinates.
`...`	Any other key value pair of aspect.

Figure aspects

- `x`, `y`
- `ymin`, `ymax`
- `size`
- `color`
- `shape`
- `label`
- `tooltip`
- `style`
- `interaction`
- `color`

Info aspects

- `x`, `y`
- `start`, `end`
- `content`
- `top`

`axis` *Axes*

Description

Configure the axes.

Customise the legend.

Usage

```
axis_x(g, ...)

axis_y(g, ...)

axis_asps(g, asps, ...)

axis_hide(g)

legend_color(g, ...)
```

```
legend_size(g, ...)

legend_asps(g, asps, ...)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the legend, pass `FALSE` to hide the axis. Visit the official documentation for the full list of options.
`asps`	Aspect (column names) to change the legend.

Functions

- `axis_x`: Customise the x axis.
- `axis_y`: Customise the y axis.
- `axis_asps`: Customise the axis by aspects (column names).
- `axis_hide`: Hide all axis.

- `legend_color`: Customise the x axis.
- `legend_size`: Customise the y axis.
- `legend_asps`: Customise the axis by aspects (column names).

Examples

```
g <- g2(cars, asp(speed, dist)) %>%
  fig_point()

# hide axis
g %>% axis_x(FALSE)

# same as above
g %>% axis_asps("speed", FALSE)

# change position
g %>% axis_x(position = "top")
g <- g2(mtcars, asp(mpg, qsec, color = gear)) %>%
  fig_point()

g %>% legend_color(position = "top")
g %>% legend_color(FALSE)
```

`axis_title` *Axis Title*

Description

Add axis titles.

Usage

```
axis_title_x(g, title, ..., fontSize = 10, offset = 30)

## S3 method for class 'g2r'
axis_title_x(g, title, ..., fontSize = 10, offset = 30)

axis_title_y(g, title, ..., fontSize = 10, offset = 50)

## S3 method for class 'g2r'
axis_title_y(g, title, ..., fontSize = 10, offset = 50)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`title`	Title to use on the axis.
`...`	Options to customise the title.
`fontSize`	Size of the font of the label.
`offset`	Offset between title and axis, if `0` the title is not visible.

`capture_event` *Events*

Description

Capture events in shiny.

Usage

```
capture_event(g, event, callback = NULL, when = c("on", "once", "off"))
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`event`	Name of event to trigger the callback.
`callback`	A callback function to run when the `event` is fired, if `NULL` then a default one is created, see details.
`when`	When the event should be triggered.

Details

The `callback` function should accept a single argument; the event data. If no callback function is passed (`NULL`) then one is generated. The generated callback function sets a shiny input `<chartId>_<eventName>` with the event data.

Examples

```
g2(iris, asp(Sepal.Width, Sepal.Length)) %>%
  fig_point() %>%
  capture_event("point:click")
```

`change_data` *Change the data*

Description

Dynamically change the data of a shiny plot.

Usage

```
change_data(g, data)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`data`	New dataset to replaced the one used to currently plot the data.

Examples

```
library(shiny)

makeData <- function(){
 data.frame(
   x = runif(100),
   y = runif(100),
   size = runif(100)
 )
}

ui <- fluidPage(
 g2Output("plot"),
 actionButton("change", "Change data")
)

server <- function(input, output){

 output$plot <- renderG2({
   g2(makeData(), asp(x, y, size = size)) %>%
     fig_point()
 })

 observeEvent(input$change, {
   g2_proxy("plot") %>%
     change_data(makeData())
 })
}

if(interactive()){
 shinyApp(ui, server)
}
```

config *Configure Figure*

Description

Configure a figure.

Usage

```
config(
  id = NULL,
  visible = NULL,
  region = NULL,
  padding = NULL,
  theme = NULL,
  ...
)
```

Arguments

`id`	Id of figure.
`visible`	Whether the figure is visible.
`region`	Region that the figure should occupy on the canvas.
`padding`	Padding around the figure.
`theme`	Theme of the figure.
`...`	Any other options from the official documentation.

Examples

```
mtcars %>%
  g2(asp(qsec)) %>%
  fig_point(
    asp(y = mpg),
    config(
      region = list(
        start = list(x = 0, y = 0),
        end = list(x = 0.5, y = 1)
      )
    )
  ) %>%
  fig_point(
    asp(y = wt),
    config(
      region = list(
        start = list(x = 0.5, y = 0),
        end = list(x = 1, y = 1)
      )
    )
  )
```

coord *Coordinates*

Description

Configure chart coordinates axis.

Usage

```
coord_type(g, type = c("rect", "polar", "theta", "helix"), ...)

coord_rotate(g, angle = 90)

coord_scale(g, x, y)

coord_reflect(g, axis = "xy")

coord_transpose(g)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`type`	Type of coordinate axis.
`...`	Any other options.
`angle`	Angle of axis rotation.
`x, y`	Scale of axis along `x` and `y` axis.
`axis`	Axis to reflect (reverse).

Functions

- `coord_type`: Type of coordinates to use where `rect` corresponds to cartesian.
- `coord_rotate`: Rotate the coordinates by a certain angle.
- `coord_scale`: Rescale the coordinates.
- `coord_reflect`: Mirror the axis along the x, y, or xy (both) axes.
- `coord_transpose`: x, y axes displacement.

Examples

```
g2(cars, asp(speed, dist, color = dist)) %>%
  fig_point() %>%
  coord_type("helix")
```

`crosstalk_select` *Crosstalk Customisation*

Description

Customise the crosstalk selection handle.

Usage

```
crosstalk_select(g, attribute, on, off = NULL)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`attribute`	Attribute to customise, e.g.: `stroke`, `fill`, `strokeOpacity`, etc.
`on`, `off`	Value to set the `attribute` to if the selected is on or off.

`elements` *Element*

Description

Function to use in `motif()` and style specific elements.

Usage

```
element(
  ...,
  shape = NULL,
  figure = c("point", "area", "edge", "line", "interval", "polygon", "schema"),
  state = c("default", "active", "inactive", "selected")
)

elementPoint(
  ...,
  shape = c("hollow-circle", "cross", "hypen", "line", "plus", "tick", "circle",
    "square", "bowtie", "diamond", "hexagon", "triangle", "triangle-down",
    "hollow-square", "hollow-bowtie", "hollow-triangle-down"),
  state = c("default", "active", "inactive", "selected")
)

elementLine(
  ...,
  shape = c("line", "dot", "dash", "smooth", "hv", "vh", "hvh", "vhv"),
  state = c("default", "active", "inactive", "selected")
)

elementArea(
  ...,
  shape = c("area", "smooth", "line", "smooth-line"),
  state = c("default", "active", "inactive", "selected")
)

elementEdge(
  ...,
  shape = c("line", "vhv", "smooth", "arc"),
  state = c("default", "active", "inactive", "selected")
)

elementInterval(
  ...,
  shape = c("rect", "hollow-rect", "line", "tick", "funnel", "pyramid"),
  state = c("default", "active", "inactive", "selected")
)

elementPolygon(..., state = c("default", "active", "inactive", "selected"))

elementSchema(
  ...,
  shape = c("box", "candle"),
  state = c("default", "active", "inactive", "selected")
)
```

Arguments

`...`	Key value pairs to pass to `style`.
`shape`	Shape to modify, if `NULL` selects a common default based on the figure, e.g.: `hollow-circle` for the `point` shape.
`figure`	Figure to modify.
`state`	State of the shape to modify.

Functions

`element()` will work for any figure, but other functions may be more convienient to use.

- `element`: Customise any element.
- `elementPoint`: Customise point.
- `elementLine`: Customise line.
- `elementArea`: Customise area.
- `elementEdge`: Customise edge.
- `elementInterval`: Customise interval.
- `elementPolygon`: Customise polygon.
- `elementSchema`: Customise schema.

Examples

```
g2(iris, asp(Sepal.Width, Sepal.Length)) %>%
  fig_point(
    asp(color = Species, shape = "circle")
  ) %>%
  motif(
    brandColor = "orange",
    backgroundColor = "black",
    elementPoint(
      shape = "circle",
      stroke = "white",
      fillOpacity = .7
    )
  )
```

`figures` *Interval*

Description

Add an interval figure.

Usage

```
fig_interval(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Examples

```
g2(sleep, asp(ID, extra, color = group)) %>%
  fig_interval()

df <- data.frame(
  cat = letters[1:5],
  value = c(0.15, .3, .65, .75, .9)
)

g2(df, asp(cat, value, color = cat, shape = "funnel")) %>%
  fig_interval(adjust("symmetric")) %>%
  coord_type("rect") %>%
  coord_transpose() %>%
  coord_scale(-1, 1) %>%
  axis_hide()
```

`fig_area` *Area*

Description

Add an area figure.

Usage

```
fig_area(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Examples

```
g2(Orange, asp(age, circumference, color = Tree)) %>%
  fig_area(adjust("stack"))
```

`fig_bin` *Bin*

Description

Add a bin figure to the chart.

Usage

```
fig_bin(
  g,
  ...,
  type = c("rectangle", "hexagon"),
  bins = c(10, 10),
  size_count = TRUE,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  alias = "count"
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`type`	The shape of bin to create.
`bins`	Number of bins by dimension (width, height).
`size_count`	Whether to size the binds by count.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.
`alias`	Name of the range to display on tooltips, labels, etc.

Details

Requires the `x` and `y` aspects.

Examples

```
g2(cars, asp(speed, dist)) %>%
  fig_bin(size_count = FALSE)

g2(cars, asp(speed, dist)) %>%
  fig_bin(type = "hexagon")
```

`fig_boxplot` *Boxplot*

Description

Add a boxplot figure to the chart.

Usage

```
fig_boxplot(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Examples

```
# wide to long
# tidyr::pivot_longer(iris, -Species)
df <- reshape(
  iris,
  varying = names(iris)[1:4],
  direction = "long",
  v.names = "value",
  idvar = "Species",
  new.row.names = 1:600,
  timevar = "var",
  times = names(iris)[1:4]
)

g2(df, asp(var, value, color = Species)) %>%
  fig_boxplot(adjust("dodge"))

g2(iris, asp(y = Sepal.Length, color = Species)) %>%
  fig_boxplot(adjust("dodge"))

g2(iris, asp(x = Species, y = Sepal.Length, color = Species)) %>%
  fig_boxplot(adjust("dodge"))
```

fig_candle *Candle*

Description

Add a candle figure to the chart.

Usage

```
fig_candle(
  g,
  ...,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  alias = "range"
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.
`alias`	Name of the range to display on tooltips, labels, etc.

Details

Requires the following aspects defined:

- `open`
- `close`
- `high`
- `low`

If no `color` argument is passed the candles are colored according to their trend (open > close = "up").

Examples

```
stock <- structure(
  list(
    date = structure(c(18626, 18627, 18631, 18632), class = "Date"),
    open = c(39.52, 39.330002, 40.169998, 41.5),
    high = c(
      39.73,
      40,
      41.560001,
      42.040001
    ),
    low = c(
      39.200001,
      39.029999,
      39.939999,
      40.77
    ),
    close = c(
      39.34,
      39.880001,
      41.400002,
      41.16
    )
  ),
  row.names = c(NA, -4L),
  class = c(
    "tbl_df",
    "tbl",
    "data.frame"
  )
)

g2(stock, asp(date, open = open, close = close, high = high, low = low)) %>%
  fig_candle() %>%
  gauge_x_time_cat()
```

fig_contour *Contour*

Description

Add a contour line figure to the chart.

Usage

```
fig_contour(
  g,
  ...,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  colors = NULL,
  nlevels = 10,
  binwidth,
  levels,
  criticalRatio = 5,
  type = c("line", "filled")
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.
`colors`	A palette of colors to define the `stroke` of each path.
`nlevels`	Passed to `contoureR::getContourLines`. An integer number of bins to split the data into *iff* ‘levels’ or ‘binwidth’ have not been specified.
`binwidth`	Passed to `contoureR::getContourLines`. The desired width of the bins, if specified, will override ‘nlevels’.
`levels`	Passed to `contoureR::getContourLines`. A numeric vector of the explicitly specified levels (zvalues) to contour, by specifying this argument, it will override ‘nlevels’ and/or ‘binwidth’. If this argument is provided, the stacking order of the contours will be preserved in the order of first occurence within the supplied vector.
`criticalRatio`	Passed to `contoureR::getContourLines`. When producing the Delaunay Mesh, the quality of the mesh can be poor in the proximity to the convex hull, Del’s that have an aspect ratio greater than this value are not considered when producing the contours. In this context, the aspect ratio is defined as the circumradius to twice its inradius, equilateral triangles have an aspect ratio of 1, everything else is larger.
`type`	Whether to draw the lines or polygons.

Details

Requires the `x`, `y` and `z` aspects, the width of the error bars can be changed with the `size` aspect.

Examples

```
data(volcano)

x <- 1:nrow(volcano)
```

```
y <- 1:ncol(volcano)
df <- expand.grid(x = x, y = y)
df$z <- apply(df, 1, function(x) {
  volcano[x[1], x[2]]
})

g <- g2(df, asp(x, y, z = z))

fig_contour(g)

fig_contour(g, colors = c("red", "blue"))

fig_contour(g, type = "filled", colors = c("darkblue", "white"))
```

fig_density *Density*

Description

Add a density figure to the chart.

Usage

```
fig_density(
  g,
  ...,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  alias = "density"
)
```

Arguments

g	An object of class g2r or g2Proxy as returned by g2() or g2_proxy().
...	Options to pass to the figure, including asp(), and adjust(), active(), selected(), and config(), other key value pairs of are passed to style.
sync	Whether to sync the axis data (align) with that used in other figures, set to FALSE to not sync or set to a character string to use as name of sync group.
data	A dataset (data.frame or tibble) to use to draw the figure.
inherit_asp	Whether to inherit the aspects paseed to g2() initialisation function.
alias	Name of the density curve.

Details

Requires the x aspects.

Examples

```
g2(cars, asp(speed)) %>%
  fig_density()

g2(iris, asp(Sepal.Width, color = Species)) %>%
  fig_density()
```

fig_edge *Edge*

Description

Add an edge figure.

Usage

```
fig_edge(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

fig_error *Error*

Description

Add an error bar figure to the chart.

Usage

```
fig_error(
  g,
  ...,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  alias = "error"
)
```

Arguments

g	An object of class g2r or g2Proxy as returned by g2() or g2_proxy().
...	Options to pass to the figure, including asp(), and adjust(), active(), selected(), and config(), other key value pairs of are passed to style.
sync	Whether to sync the axis data (align) with that used in other figures, set to FALSE to not sync or set to a character string to use as name of sync group.
data	A dataset (data.frame or tibble) to use to draw the figure.
inherit_asp	Whether to inherit the aspects paseed to g2() initialisation function.
alias	Name of the range to display on tooltips, labels, etc.

Details

Requires the ymin and ymax aspects, the width of the error bars can be changed with the size aspect.

Examples

```
df <- data.frame(
  x = as.factor(c(1:10, 1:10)),
  y = runif(20, 10, 15),
  grp = rep(c("A", "B"), each = 2)
)

df$ymin <- df$y - runif(20, 1, 2)
df$ymax <- df$y + runif(20, 1, 2)

g2(df, asp(x = x, color = grp)) %>%
  fig_error(asp(ymin = ymin, ymax = ymax), adjust("dodge")) %>%
  fig_interval(
    asp(y = y),
    adjust("dodge"),
    fillOpacity = .4
  )
```

---

fig_heatmap *Heatmap*

Description

Add a path figure.

Usage

```
fig_heatmap(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

`fig_histogram` *Histogram*

Description

Add a histogram figure to the chart.

Usage

```
fig_histogram(
  g,
  ...,
  bin_width = 5,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  alias = "count"
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`bin_width`	Width of bin.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.
`alias`	Name of the range to display on tooltips, labels, etc.

Examples

```
df <- data.frame(
  grp = rep(c("A", "B"), each = 200),
  val = c(
    rnorm(200, mean = 57, sd = 5),
    rnorm(200, mean = 53, sd = 5)
  )
)

g2(df, asp(val, color = grp)) %>%
  fig_histogram(adjust("stack"), bin_width = 1)
```

`fig_line` *Line*

Description

Add a line figure.

Usage

```
fig_line(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Examples

```
g2(CO2, asp(conc, uptake, color = Plant)) %>%
  fig_line()
```

`fig_map` *Map*

Description

Add a map figure.

Usage

```
fig_map(g, ..., inherit_asp = TRUE, sync = TRUE, map = get_world_map())
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`map`	Name of map to pass to the `region` argument of the `get_map_data()` function, or map object as returned by `get_world_map()`, `get_gadm_data()`, or `get_map_data()`, or a `SpatialPolygonsDataFrame` as returned by `raster::getData()`, or a `geo_list` as obtained from `geojsonio::geojson_list()`, or `MULTIPOLYGON` of the class`sf` as obtained from reading shapefiles.

Examples

```
g2() %>%
  fig_map(stroke = "#fff", fill = "gray") %>%
  axis_hide()
```

`fig_path` *Path*

Description

Add a path figure.

Usage

```
fig_path(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Examples

```
df <- data.frame(
  x = runif(100),
  y = runif(100)
)

g2(df, asp(x, y)) %>%
  fig_path()
```

`fig_pie` *Pie*

Description

Add a pie figure to the chart.

Usage

```
fig_pie(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Examples

```
df <- data.frame(
  label = letters[1:5],
  value = runif(5)
)

g2(df, asp(y = value, color = label)) %>%
  fig_pie()
```

`fig_point` *Point*

Description

Add a point figure.

Usage

```
fig_point(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Examples

```
g2(cars) %>%
  fig_point(asp(speed, dist))

g2(mtcars, asp(mpg, disp, size = qsec)) %>%
  fig_point(asp(color = "red", shape = "square"))
```

`fig_polygon` *Polygon*

Description

Add a polygon figure.

Usage

```
fig_polygon(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

`fig_range` *Range*

Description

Add a range figure to the chart.

Usage

```
fig_range(
  g,
  ...,
  type = c("interval", "area"),
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  alias = "range"
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`type`	Type of figure to use.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.
`alias`	Name of the range to display on tooltips, labels, etc.

Details

Requires the `ymin` and `ymax` aspects.

Examples

```
df <- data.frame(
  x = 1:100,
  ymin = runif(100, 1, 5),
  ymax = runif(100, 6, 13)
)

g2(df, asp(x, ymin = ymin, ymax = ymax)) %>%
  fig_range()
```

`fig_ribbon` *Ribbon*

Description

Add a ribbon figure to the chart.

Usage

```
fig_ribbon(
  g,
  ...,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE,
  alias = "ribbon"
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.
`alias`	Name of the range to display on tooltips, labels, etc.

Details

Requires the `ymin` and `ymax` aspects.

Examples

```
df <- data.frame(
  x = 1:100,
  ymin = runif(100, 1, 5),
  ymax = runif(100, 6, 13)
)

g2(df, asp(x, ymin = ymin, ymax = ymax)) %>%
  fig_ribbon()
```

`fig_rug` *Rug*

Description

Add a rug figure to the chart.

Usage

```
fig_rug(
  g,
  ...,
  strokeOpacity = 0.5,
  axis = c("x", "y"),
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`strokeOpacity`	Opacity of rug marks.
`axis`	Axis to place the rug marks on.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Details

Requires the `x` and `y` aspects.

Examples

```
g2(mtcars, asp(wt, mpg)) %>%
  fig_point() %>%
  fig_rug() %>%
  fig_rug(asp(size = 10), axis = "y")
```

`fig_schema` *Schema*

Description

Add a schema figure.

Usage

```
fig_schema(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

`fig_segment` *Segment*

Description

Add a segments figure to the chart.

Usage

```
fig_segment(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the figure, including `asp()`, and `adjust()`, `active()`, `selected()`, and `config()`, other key value pairs of are passed to style.
`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

`fig_smooth` *Smooth*

Description

Add a smooth(ing) figure to the chart.

Usage

```
fig_smooth(
  g,
  ...,
 method = c("linear", "gaussian", "cosine", "epanechnikov", "quartic", "triangular",
   "tricube", "triweight", "uniform", "polynomial", "logarithmic", "boxcar", "power",
    "exponential"),
  band_width = 1,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE
)
```

Arguments

g	An object of class g2r or g2Proxy as returned by g2() or g2_proxy().
...	Options to pass to the figure, including asp(), and adjust(), active(), selected(), and config(), other key value pairs of are passed to style.
method	Smoothing method to use.
band_width	Step size for Silverman's algorithm.
sync	Whether to sync the axis data (align) with that used in other figures, set to FALSE to not sync or set to a character string to use as name of sync group.
data	A dataset (data.frame or tibble) to use to draw the figure.
inherit_asp	Whether to inherit the aspects paseed to g2() initialisation function.

Details

Requires the x and y aspects.

This is a convenience function for a quick smoothing, see the online documentation to see how to use your own model for more control.

Examples

```
g2(cars, asp(speed, dist)) %>%
  fig_point() %>%
  fig_smooth(method = "gaussian")

g2(iris, asp(Sepal.Width, Sepal.Length, color = Species)) %>%
  fig_point() %>%
  fig_smooth()
```

---

fig_voronoi *Voronoi*

Description

Add a voronoi figure to the chart.

Usage

```
fig_voronoi(g, ..., sync = TRUE, data = NULL, inherit_asp = TRUE)
```

Arguments

g	An object of class g2r or g2Proxy as returned by g2() or g2_proxy().
...	Options to pass to the figure, including asp(), and adjust(), active(), selected(), and config(), other key value pairs of are passed to style.
sync	Whether to sync the axis data (align) with that used in other figures, set to FALSE to not sync or set to a character string to use as name of sync group.
data	A dataset (data.frame or tibble) to use to draw the figure.
inherit_asp	Whether to inherit the aspects paseed to g2() initialisation function.

Details

Requires the x, y, and color arguments.

Examples

```
df <- data.frame(
  x = runif(25, 1, 500),
  y = runif(25, 1, 500),
  value = runif(25, 1, 500)
)

g2(df, asp(x, y, color = value)) %>%
  fig_voronoi()
```

`fig_waffle` *Waffle*

Description

Add a waffle figure to the chart.

Usage

```
fig_waffle(
  g,
  ...,
  n = 500,
  rows = 10,
  size = c(1, 1),
  gap = 0.1,
  min_size = 15,
  sync = TRUE,
  data = NULL,
  inherit_asp = TRUE
)
```

Arguments

g	An object of class g2r or g2Proxy as returned by g2() or g2_proxy().
...	Options to pass to the figure, including asp(), and adjust(), active(), selected(), and config(), other key value pairs of are passed to style.
n	Number of squares to use.
rows	Number of rows.
size	Size of squares.
gap	Gap between squares.
min_size	Minimum size of squares.

`sync`	Whether to sync the axis data (align) with that used in other figures, set to `FALSE` to not sync or set to a character string to use as name of sync group.
`data`	A dataset (`data.frame` or `tibble`) to use to draw the figure.
`inherit_asp`	Whether to inherit the aspects paseed to `g2()` initialisation function.

Details

Requires the `x` and `color` aspects.

Examples

```
fruits <- data.frame(
  fruit = c("Apples", "Bananas", "Pears", "Oranges"),
  value = c(.45, .15, .35, .05) * 100
)

g2(fruits, asp(value, color = fruit)) %>%
  fig_waffle() %>%
  motif(padding = 50) %>%
  axis_hide()
```

`g2` *Initialise*

Description

Initialise a chart.

Usage

```
g2(
  data = NULL,
  ...,
  width = NULL,
  height = NULL,
  elementId = NULL,
  digits = NULL,
  reorder = TRUE
)
```

Arguments

`data`	A data.frame or tibble containing data to chart, an object of class `igraph`, an object of class `ts`, or as `crosstalk::sharedDataset`.
`...`	Aspects of the chart, see `asp()`.
`width, height`	Dimensions of the chart, accepts any valid CSS unit e.g.: `100%`, numerics are treated as pixels, e.g.: `400` = `400px`.

`elementId`	Valid CSS id attribute.
`digits`	Maximum number of digits after the comma to show on the chart.
`reorder`	Whether to internally reorder the data, namely the `x` and `color`. The `x` axis must be reordered in a descending order for most data type since G2.js plots data as-is. Moreover, `color` order of all data.frames passed either to this function or subsequent `fig_*` layers must be identical or the colors will might match the legends on the plot. However, one may sometimes not want the data to be reordered.

Examples

```
g2(cars) %>%
  fig_point(asp(speed, dist))
```

`g2r-shiny` *Shiny Bindings*

Description

Output and render functions for using g2r within Shiny applications and interactive Rmd documents.

Usage

```
g2Output(outputId, width = "100%", height = "400px")

renderG2(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 g2r
`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.

g2_action *Plot Action*

Description

Include dynamic elements in Rmarkdown.

Usage

```
g2_action(plot_id, btn_id, ..., data = NULL, reorder = TRUE)
```

Arguments

`plot_id`	Id of chart to interact with.
`btn_id`	Id of the `input_button()` that triggers the action.
`...`	Aspects, see `asp()`.
`data`	Data.frame containing data to plot.
`reorder`	Whether to internally reorder the data, namely the `x` and `color`. The `x` axis must be reordered in a descending order for most data type since G2.js plots data as-is. Moreover, `color` order of all data.frames passed either to this function or subsequent `fig_*` layers must be identical or the colors will might match the legends on the plot. However, one may sometimes not want the data to be reordered.

g2_proxy *Shiny Proxy*

Description

Proxy to dynamically interact with the chart in shiny.

Usage

```
g2_proxy(id, ..., data = NULL, session = shiny::getDefaultReactiveDomain())
```

Arguments

`id`	Id of chart to interact with.
`...`	Aspects, see `asp()`.
`data`	Data.frame containing data to plot.
`session`	A valid shiny session.

Examples

```
library(shiny)

dataset <- data.frame(x = 1:100, y = runif(100, 1, 100))

ui <- fluidPage(
  g2Output("plot"),
  actionButton("add", "Add figure")
)

server <- function(input, output, session) {
  output$plot <- renderG2({
    g2(dataset, asp(x, y)) %>%
      fig_point()
  })

  observeEvent(input$add, {
    df <- data.frame(x = 1:100, y = runif(100, 1, 100))
    g2_proxy("plot", data = df) %>%
      fig_point(asp(x, y)) %>%
      render()
  })
}

if (interactive()) {
  shinyApp(ui, server)
}
```

---

gauge *Gauge Grid*

Description

Gauge the variables (`aspects`) used to define axis and grid of the plot.

Usage

```
gauge(
  g,
  asp,
  ...,
  nice = TRUE,
  range = NULL,
  min = NULL,
  max = NULL,
  min_limit = NULL,
  max_limit = NULL,
  alias = NULL,
  tick_count = NULL,
  max_tick_count = NULL
)

gauge_x_time(g, ..., show_last = FALSE)

gauge_y_time(g, ..., show_last = FALSE)

gauge_x_linear(g, ..., tick_interval = NULL)

gauge_y_linear(g, ..., tick_interval = NULL)

gauge_x_cat(g, ...)

gauge_y_cat(g, ...)

gauge_x_time_cat(g, ...)

gauge_y_time_cat(g, ...)

gauge_x_log(g, ..., base = 10)

gauge_y_log(g, ..., base = 10)

gauge_x_pow(g, ...)

gauge_y_pow(g, ...)

gauge_x_quantile(g, ...)

gauge_y_quantile(g, ...)

gauge_x_quantize(g, ...)

gauge_y_quantize(g, ...)

gauge_x_identity(g, ...)

gauge_y_identity(g, ...)

gauge_asp(g, ...)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`asp`	Bare column name of aspect to apply the gauge to.
`...`	Options to gauge variables (`aspects` defined by `asp()`).
`nice`	Automatically adjust `min`, and `max`.
`range`	A vector of length 2 giving the minimum, and maximum.

`min`, `max`	Range of the gauge.
`min_limit`, `max_limit`	Strict range of the ticks.
`alias`	Alias name of the gauge and variable to display.
`tick_count`	Maximum number of ticks.
`max_tick_count`	Maximum number of ticks.
`show_last`	Whether to force show the last tick (only for `time` gauges).
`tick_interval`	Minimum tick interval, only applies to linear type of gauge.
`base`	Base of log.

Types

- `cat`: Categorical.
- `timeCat`: Categorical time.
- `linear`: Linear.
- `time`: Date, time, etc.
- `log`: Logarithmic.
- `pow`: Exponential.
- `quantize`: Manual quantiles.
- `quantile`: Auto-generated quantiles.
- `identity`: Constant.

Examples

```
g <- g2(cars, asp(speed, dist)) %>%
  fig_point()

g %>% gauge(speed, min = 0)
g %>% gauge_y_log(title = "Log")
g %>% gauge(dist, tickCount = 10)
```

---

gaugeViews *Gauge Aspects*

Description

Customise aspects of the chart.

Usage

```
gauge_color(g, ...)

gauge_size(g, ...)

gauge_shape(g, ...)

gauge_tooltip(g, ...)

gauge_label(g, ...)

gauge_style(g, ...)

gauge_interplay(g, ...)
```

Arguments

g	An object of class g2r or g2Proxy as returned by g2() or g2_proxy().
...	Arguments to customise the gauge. Generally, key value pairs of options, a vector of hex colors, or a JavaScript function (wrapped in htmlwidgets::JS()).

See Also

gauge to gauge aspects of the grid and axis.

Examples

```
# base plot
g <- g2(cars, asp(speed, dist)) %>%
  fig_point(asp(color = speed))

# color with vector
g %>% gauge_color(c("red", "white", "blue"))

# color with callback
cb <- "function(speed){
 if(speed > 10){
   return 'blue';
 }
 return 'red';
}"

g %>% gauge_color(htmlwidgets::JS(cb))
```

`global_digits` *Digits*

Description

Maximum number of digits to show on charts.

Usage

```
global_digits(n = 16L)
```

Arguments

`n`	Maximum number of digits.

`info` *Text*

Description

Add a point figure.

Usage

```
info_text(g, ..., style = NULL, data = NULL)

info_image(g, ..., style = NULL, data = NULL)

info_arc(g, ..., style = NULL, data = NULL)

info_line(g, ..., style = NULL, data = NULL)

info_vline(g, ..., style = NULL, data = NULL)

info_hline(g, ..., style = NULL, data = NULL)

info_abline(g, ..., style = NULL, data = NULL, direction = c(1, 2))

info_region(g, ..., style = NULL, data = NULL)

info_region_filter(g, ..., style = NULL, data = NULL)

info_marker(g, ..., style = NULL, data = NULL)

info_data_region(g, ..., style = NULL, data = NULL)
```

```
info_shape(g, ..., style = NULL, data = NULL)

info_html(g, ..., style = NULL, data = NULL)
```

Arguments

`g`	An object of class `g2r` as returned by `g2()`.
`...`	Options to pass to the informational annotation.
`style`	A list of options defning the style.
`data`	A dataset to use with `asp()`.
`direction`	Direction of diagonal line.

Details

`info_vline`, and `info_hline` use the `x`, and `y` `asp()` for placement.

See Also

Official annotation documentation for defails pon what to pass to `...`, and `asp()`.

Examples

```
df <- head(cars, 5)

g2(cars, asp(speed, dist)) %>%
  fig_point() %>%
  info_text(
    position = c(20, 35),
    content = "Look here!"
  ) %>%
  info_text(
    asp(speed, dist),
    content = "Using aspects",
    data = df
  )

g2(cars, asp(speed, dist)) %>%
  fig_point() %>%
  info_vline(asp(x = 20)) %>%
  info_hline(asp(y = 20))
```

`input_button` *Button Input*

Description

Add a button input.

Usage

```
input_button(id, label, class = "default")
```

Arguments

`id`	Id of the button.
`label`	Label to display.
`class`	Class of the button.

Details

The `class` argument defines the style of the button in Bootstrap 3, generally accepts:`for`

- `default`
- `info`
- `success`
- `warning`
- `danger`

`input_select` *Select Input*

Description

Select Input

Usage

```
input_select(id, label, choices)
```

Arguments

`id`	Valid CSS id of the element.
`label`	Label to display.
`choices`	Vector of choices

`input_slider` *Slider Input*

Description

Add a slider to an R markdown document.

Usage

```
input_slider(id, label, value, min, max, step = 1)
```

Arguments

`id`	Valid CSS id of the element.
`label`	Label to display.
`value`	Initial value of the slider.
`min, max`	Minimum and maximum value the slider can be set.
`step`	Interval between steps.

Examples

```
input_slider(
  "mySlider",
  "The label",
  value = 5,
  min = 0,
  max = 10
)
```

`interplay` *Interplay*

Description

Configure global interplay (interactions) for the chart. See `gauge_interplay()` to customise figure-level interplay.

Usage

```
interplay(g, ...)

remove_interplay(g, ...)

register_interplay(g, name, ...)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	String(s) defining interactions.
`name`	Name of interaction to register

Examples

```
# global interaction on chart
df <- data.frame(
  x = letters,
  y = runif(26)
)

g2(df, asp(x, y)) %>%
  fig_interval(
    selected(fill = "orange")
  ) %>%
  interplay("element", "selected")

# brush
g2(cars, asp(speed, dist)) %>%
  fig_point(asp(interplay = "brush"))

# register
df <- data.frame(
  x = c(letters, letters),
  y = runif(52),
  grp = c(rep("a", 26), rep("b", 26))
)

g2(df, asp(x, y, color = grp)) %>%
  fig_interval(
    asp(interplay = "element-highlight-by-color"),
    adjust("dodge")
  ) %>%
  register_interplay(
    "element-highlight-by-color",
    start = list(
      list(
        trigger = "element:mouseenter",
        action = "element-highlight-by-color:highlight"
      )
    ),
    end = list(
      list(
        trigger = "element:mouseleave",
        action = "element-highlight-by-color:reset"
      )
    )
  )
```

layout_arc *Layout Arc*

Description

Layout as arc using the alter package.

Usage

```
layout_arc(
  g,
  sourceWeight = NULL,
  targetWeight = NULL,
  thickness = 0.05,
  marginRatio = 0.1
)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`sourceWeight, targetWeight`	Bare name of column containing weights of source and target in the edges data.frame.
`thickness`	Node height, between `0` and `1`.
`marginRatio`	Space ratio, between `0` and `1`.

Examples

```
ig <- igraph::erdos.renyi.game(100, 1 / 100)

g2(ig, asp(x, y)) %>%
  layout_arc() %>%
  fig_edge(asp(color = source, shape = "arc"), opacity = .3) %>%
  fig_point(asp(color = id, shape = "circle", size = value)) %>%
  coord_type("polar") %>%
  coord_reflect("y") %>%
  axis_hide()

g2(ig, asp(x, y)) %>%
  layout_arc() %>%
  fig_edge(asp(color = source, shape = "arc"), opacity = .3) %>%
  fig_point(asp(color = id, shape = "circle", size = value))
```

`layout_igraph` *Layout with igraph*

Description

Layout the graph using an igraph layout function. This function only works with the graph was initialised with an object of class `igraph`.

Usage

```
layout_igraph(g, ..., method = igraph::layout_nicely)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Any options to pass to the `method` function.
`method`	An igraph layout function to compute the nodes and edges (source and target) position on the canvas.

Details

The function runs the `method` to obtain the `x` and `y` coordinates. These are added to the nodes data.frame (extracted from initial graph) and to the edges data.frame, as `x` and `y` *nested* columns, e.g.: `c(source_x, target_x)`. These `x` and `y` coordinates can be used in `asp()` (see example).

Examples

```
ig <- igraph::make_ring(100)

# use x and y for positioning
g2(ig, asp(x, y)) %>%
  layout_igraph() %>%
  fig_edge() %>%
  fig_point(asp(shape = "circle")) %>%
  axis_hide()
```

`map_data` *Get Map*

Description

Retrieve map data to pass to the `maps` argument of the `fig_map()` function.

Usage

```
get_gadm_data(iso3c, level = c(0, 1, 2, 3, 4), keep = 0.05)

get_map_data(region = ".", level = c("region", "subregion"), name = "world")

get_world_map()
```

Arguments

`iso3c`	Iso3c code of the country to retrieve, e.g.: `USA`.
`level`	Level of the polygons to draw, either the region or subregion.
`keep`	Proportion of points to retain, it is highly recommended to reduce the detail of the map or it will take too long to load in the browser. Set to `NULL` to not reduce the amount of details.
`region`	Character vector that names the polygons to draw.
`name`	Name of the database to use.

Functions

- `get_gadm_data`: Retrieves country-level data from gadm.org.
- `get_map_data`: Uses the `maps::map()` function to retrieve the map data, similar to `ggplot2::map_data`.
- `get_world_map`: Retrives a world map (from GeoJSON).

`motif` *Motif*

Description

Set the motif of the chart, defaults to `light`.

Usage

```
motif(
  g,
  ...,
  brandColor = NULL,
  backgroundColor = "transparent",
  renderer = c("canvas", "svg"),
  padding = "auto",
  visible = TRUE
)

global_motif(
  ...,
  brandColor = NULL,
  backgroundColor = "transparent",
  renderer = c("canvas", "svg"),
  padding = "auto",
  visible = TRUE
)

motif_from_json(g, path)

motif_from_list(g, motif)

motif_as_list(..., brandColor = NULL, backgroundColor = "transparent")
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Key value pair defining style, or `element()`.
`brandColor`	Main default color.
`backgroundColor`	Plot background color.
`renderer`	Renderer to use, defaults to `canvas`.
`padding`	An integer, or a vector of length 4.
`visible`	Whether the chart is visible.
`path`	Path to JSON file.
`motif`	List defnining the theme, similar to JSON.

Details

The function `motif_from_json()` can be used to define the theme from a JSON file of theme, to see the default theme file: `system.file("theme.json", package = "g2r")`.

Functions

- `motif`: Defines the motif of a visualisation.
- `motif_from_json`: Defines the motif from a JSON file of theme, see the theme file. `system.file("theme.json", package = "g2r")`
- `motif_from_list`: Defines the motif from a `list`, derived from the JSON file.
- `motif_as_list`: Returns a motif as a `list` to use with `motif_from_list()`.
- `global_motif`: Define a global motif that will be used by all subsequent charts.

Examples

```
g2(iris, asp(Sepal.Width, Sepal.Length)) %>%
  fig_point(
    asp(color = Species, shape = "circle")
  ) %>%
  motif(
    brandColor = "orange",
    backgroundColor = "black",
    elementPoint(
      shape = "circle",
      stroke = "white",
      fillOpacity = .7
    )
  )
```

`new_animation` *New Animation*

Description

Convenience function to create a new animation, equivalent to `Animation$new()`.

Usage

```
new_animation()
```

See Also

Animation

`palettes` *Color Palettes*

Description

Convenience function to easily apply colors palettes.

Usage

```
gauge_color_viridis(g)

gauge_color_plasma(g)

gauge_color_inferno(g)

gauge_color_magma(g)

gauge_color_accent(g)

gauge_color_dark2(g)

gauge_color_paired(g)
```

```
gauge_color_pastel1(g)

gauge_color_pastel2(g)

gauge_color_set1(g)

gauge_color_set2(g)

gauge_color_set3(g)

gauge_color_neon(g)

gauge_color_std(g)

gauge_color_pink(g)

gauge_color_orange(g)

gauge_color_lime(g)

gauge_color_blue(g)

gauge_color_aw(g)

gauge_color_g2rq(g)

gauge_color_g2rc(g)

gauge_color_g2rd(g)

gauge_color_brbg(g)

gauge_color_piyg(g)

gauge_color_prgn(g)

gauge_color_puor(g)

gauge_color_rdbu(g)

gauge_color_rdgy(g)

gauge_color_rdylbu(g)

gauge_color_rdylgn(g)

gauge_color_spectral(g)
```

```
gauge_color_flashy(g)

gauge_color_red(g)

gauge_color_ryb(g)

gauge_color_bwg(g)
```

Arguments

g	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.

Palettes

Palletes from the `viridisLite` package, ideal for: continuous data.

- `gauge_color_viridis` (continuous)
- `gauge_color_plasma` (continuous)
- `gauge_color_inferno` (continuous)
- `gauge_color_magma` (continuous)

Palettes from color brewer:

- `gauge_color_accent` (qualitative)
- `gauge_color_dark2` (qualitative)
- `gauge_color_paired` (qualitative)
- `gauge_color_pastel1` (qualitative)
- `gauge_color_pastel2` (qualitative)
- `gauge_color_set1` (qualitative)
- `gauge_color_set2` (qualitative)
- `gauge_color_set3` (qualitative)
- `gauge_color_brbg` (diverging)
- `gauge_color_piyg` (diverging)
- `gauge_color_prgn` (diverging)
- `gauge_color_puor` (diverging)
- `gauge_color_rdbu` (diverging)
- `gauge_color_rdgy` (diverging)
- `gauge_color_rdylbu` (diverging)
- `gauge_color_rdylgn` (diverging)
- `gauge_color_spectral` (diverging)

Palettes taken from coolors.co:

- `gauge_color_neon` (continuous)

- gauge_color_std (continuous)
- gauge_color_orange (continuous)
- gauge_color_pink (continuous)
- gauge_color_lime (continuous)
- gauge_color_blue (continuous)
- gauge_color_red (discrete)
- gauge_color_flashy (discrete)
- gauge_color_ryb (discrete)
- gauge_color_bwg (diverging)

Palettes from awtools package:

- gauge_color_aw (qualitative)

Custom:

- gauge_color_g2rc (continuous)
- gauge_color_g2rq (qualitative)
- gauge_color_g2rd (diverging)

---

planes *Planes*

Description

Split the chart into planes according to variables.

Usage

```
planes(
  g,
  asp,
  ...,
  type = c("rect", "list", "matrix", "circle", "tree", "mirror"),
  sync = TRUE
)
```

Arguments

g	An object of class g2r or g2Proxy as returned by g2() or g2_proxy().
asp	Aspects that define split, these must be defined as a formula, e.g.: ~x+y.
...	Any other option.
type	Type of planes to use.
sync	Whether to sync the aspects used for the planes with others used elsewhere, similar to that of fig_point().

Examples

```
g2(iris, asp(Sepal.Length, Sepal.Width, color = Species)) %>%
  fig_point() %>%
  planes(~Species, type = "tree")
```

---

qg2 *Quick Plot*

Description

Draw a quick plot.

Usage

```
qg2(object, ..., conf_level = 0.95, intercept = FALSE, names = NULL)
```

Arguments

`object`	An object containing data to plot, often a model.
`...`	Ignored
`conf_level`	Confidence level.
`intercept`	Whether to display the intercept.
`names`	Names of the models.

---

radio *Checkbox and Radio*

Description

Add a checkbox or radio input.

Usage

```
input_checkbox(id, label, choices, selected = NULL, inline = TRUE)

input_radio(id, label, choices, selected = NULL, inline = TRUE)
```

Arguments

`id`	Id of input.
`label`	Label of the input.
`choices`	Vector of choices to define either the checboxes or radio inputs.
`selected`	Vector of `choices` that are selected by default.
`inline`	Whether the input is inline.

`remove_figure` *Remove a figure*

Description

Remove a figure from the plot.

Usage

```
remove_figure(g, index)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`index`	Index of figure to remove. Either the numeric index of figure (layer number in order it was added to the visualisation), or the `id` of the figure as set by `config()` (see examples).

Examples

```
g <- g2(mtcars, asp(qsec, wt)) %>%
  fig_point(config(id = "myPoints")) %>%
  fig_point(asp(y = drat))

# all figures
g

# remove figure
remove_figure(g, "myPoints")

library(shiny)

df <- data.frame(
  x = 1:100,
  y = runif(100),
  z = runif(100)
)

ui <- fluidPage(
  g2Output("plot"),
  actionButton("rm", "Randomly remove a figure")
)

server <- function(input, output) {
  output$plot <- renderG2({
    g2(df, asp(x, y)) %>%
      fig_point() %>%
      fig_line(asp(y = z))
  })
```

```
    observeEvent(input$rm, {
      g2_proxy("plot") %>%
        remove_figure(sample(1:2, 1))
    })
  }

  if (interactive()) {
    shinyApp(ui, server)
  }
```

`render` *Render*

Description

Render proxy calls.

Usage

```
render(g, update = TRUE)
```

Arguments

`g`	An object of class `g2Proxy` as returned by `g2_proxy()`.
`update`	Whether to trigger the update process.

`scrollbar` *Scrollbar*

Description

Add a scrollbar to the chart.

Usage

```
scrollbar(g, ...)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the slider.

See Also

The official documentation for the list options to pass to `...`.

`slider` *Slider*

Description

Add a slider to the chart.

Usage

```
slider(g, ...)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the slider.

See Also

The official documentation for the list options to pass to `...`.

`state` *State*

Description

Customise the styles of figures given states (active or selected).

Usage

```
active(...)

selected(...)
```

Arguments

`...`	Key value pair passed to styles.

Examples

```
g2(iris, asp(Sepal.Width, Sepal.Length)) %>%
  fig_point(
    selected(fill = "red")
  ) %>%
  interplay("element", "selected")
```

`subject` *Subject*

Description

Add a subject to the plot.

Usage

```
subject(g, subject, tag = htmltools::h3)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`subject`	Subject of chart to display.
`tag`	`htmltools` tag function to use.

Examples

```
g2(cars, asp(speed, dist)) %>%
 fig_point() %>%
 subject("Points")
```

`template` *Tooltip Template*

Description

Convenience function to create tooltip templates (`itemTp` argument in `tooltip()` function).

Usage

```
tpl(...)

tpl_item(name, value, marker = TRUE)
```

Arguments

`...`	One or more `tpl_item()`.
`name, value`	Name and value of the tooltip item.
`marker`	Whether to include the color marker (dot) in the tooltip.

Details

The arguments `title`, `name`, and `value` accept either a bare column name from the data to use as `{mustache}`/`{handlebar}` in the template. If a string is passed then it is treated as constant.

Examples

```
template <- tpl(
  tpl_item(
    island,
    bill_depth_mm
  )
)
```

`tooltip` *Tooltip*

Description

Configure the tooltip applied to the entire chart. See `gauge_tooltip()` to customise a specific tooltip (the tooltip of a specific figure).

Usage

```
tooltip(g, ...)
```

Arguments

`g`	An object of class `g2r` or `g2Proxy` as returned by `g2()` or `g2_proxy()`.
`...`	Options to pass to the axis, pass `FALSE` to hide the axis. Visit the official documentation for the full list of options.

Examples

```
g2(mtcars, asp(drat, qsec, color = hp)) %>%
  fig_point() %>%
  tooltip(
    showCrosshairs = TRUE,
    crosshairs = list(type = "xy")
  )
```

to_g2r *Convert to Tibble*

Description

Converts objects to objects g2r can work with, generally a `tibble::tibble`.

Usage

```
to_g2r(data = NULL)
```

Arguments

`data`	An object to convert.

Details

This is exposed so the user can understand what happens under the hood and which variables/columns can subsequently be used in figures with `asp()`.

These methods are used in the `g2()` function to preprocess `data` objects.

Examples

```
## Not run:
to_g2r(AirPassengers)

## End(Not run)
```

Index

action_filter_data, 3
action_select_data, 4
action_toggle_visibility, 5
active *(state)*, 63
active(), *19–22*, *24–39*
adjust, 6
adjust(), *19–22*, *24–39*
aka, 7
Animation, *56*
Animation *(animation)*, 7
animation, 7
asp, 9
asp(), *19–22*, *24–40*, *42*, *44*, *48*, *53*, *66*
axis, 10
axis_asps *(axis)*, 10
axis_hide *(axis)*, 10
axis_title, 12
axis_title_x *(axis_title)*, 12
axis_title_y *(axis_title)*, 12
axis_x *(axis)*, 10
axis_y *(axis)*, 10

capture_event, 12
change_data, 13
config, 14
config(), *19–22*, *24–39*, *61*
coord, 15
coord_reflect *(coord)*, 15
coord_rotate *(coord)*, 15
coord_scale *(coord)*, 15
coord_transpose *(coord)*, 15
coord_type *(coord)*, 15
crosstalk_select, 16

element *(elements)*, 16
element(), *18*, *55*
elementArea *(elements)*, 16
elementEdge *(elements)*, 16
elementInterval *(elements)*, 16
elementLine *(elements)*, 16
elementPoint *(elements)*, 16
elementPolygon *(elements)*, 16
elements, 16
elementSchema *(elements)*, 16

fig_area, 19
fig_bin, 20
fig_boxplot, 21
fig_candle, 22
fig_contour, 23
fig_density, 25
fig_edge, 26
fig_error, 26
fig_heatmap, 27
fig_histogram, 28
fig_interval *(figures)*, 18
fig_line, 29
fig_map, 30
fig_path, 30
fig_pie, 31
fig_point, 32
fig_point(), *59*
fig_polygon, 33
fig_range, 33
fig_ribbon, 34
fig_rug, 35
fig_schema, 36
fig_segment, 37
fig_smooth, 37
fig_voronoi, 38
fig_waffle, 39
figures, 18

g2, 40
g2(), *3–5*, *7*, *11–13*, *15, 16*, *19–22*, *24–40*, *44*, *46*, *48*, *51–53*, *55*, *58, 59*, *61–66*
g2_action, 42
g2_proxy, 42
g2_proxy(), *3–5*, *7*, *11–13*, *15, 16*, *19–22*, *24–39*, *44*, *46*, *51–53*, *55*, *58, 59*,

61–65
g2Output *(g2r-shiny)*, 41
g2r-shiny, 41
gauge, 43, *46*
gauge_asp *(gauge)*, 43
gauge_color *(gaugeViews)*, 45
gauge_color_accent *(palettes)*, 56
gauge_color_aw *(palettes)*, 56
gauge_color_blue *(palettes)*, 56
gauge_color_brbg *(palettes)*, 56
gauge_color_bwg *(palettes)*, 56
gauge_color_dark2 *(palettes)*, 56
gauge_color_flashy *(palettes)*, 56
gauge_color_g2rc *(palettes)*, 56
gauge_color_g2rd *(palettes)*, 56
gauge_color_g2rq *(palettes)*, 56
gauge_color_inferno *(palettes)*, 56
gauge_color_lime *(palettes)*, 56
gauge_color_magma *(palettes)*, 56
gauge_color_neon *(palettes)*, 56
gauge_color_orange *(palettes)*, 56
gauge_color_paired *(palettes)*, 56
gauge_color_pastel1 *(palettes)*, 56
gauge_color_pastel2 *(palettes)*, 56
gauge_color_pink *(palettes)*, 56
gauge_color_piyg *(palettes)*, 56
gauge_color_plasma *(palettes)*, 56
gauge_color_prgn *(palettes)*, 56
gauge_color_puor *(palettes)*, 56
gauge_color_rdbu *(palettes)*, 56
gauge_color_rdgy *(palettes)*, 56
gauge_color_rdylbu *(palettes)*, 56
gauge_color_rdylgn *(palettes)*, 56
gauge_color_red *(palettes)*, 56
gauge_color_ryb *(palettes)*, 56
gauge_color_set1 *(palettes)*, 56
gauge_color_set2 *(palettes)*, 56
gauge_color_set3 *(palettes)*, 56
gauge_color_spectral *(palettes)*, 56
gauge_color_std *(palettes)*, 56
gauge_color_viridis *(palettes)*, 56
gauge_interplay *(gaugeViews)*, 45
gauge_interplay(), *50*
gauge_label *(gaugeViews)*, 45
gauge_shape *(gaugeViews)*, 45
gauge_size *(gaugeViews)*, 45
gauge_style *(gaugeViews)*, 45
gauge_tooltip *(gaugeViews)*, 45
gauge_tooltip(), *65*
gauge_x_cat *(gauge)*, 43
gauge_x_identity *(gauge)*, 43
gauge_x_linear *(gauge)*, 43
gauge_x_log *(gauge)*, 43
gauge_x_pow *(gauge)*, 43
gauge_x_quantile *(gauge)*, 43
gauge_x_quantize *(gauge)*, 43
gauge_x_time *(gauge)*, 43
gauge_x_time_cat *(gauge)*, 43
gauge_y_cat *(gauge)*, 43
gauge_y_identity *(gauge)*, 43
gauge_y_linear *(gauge)*, 43
gauge_y_log *(gauge)*, 43
gauge_y_pow *(gauge)*, 43
gauge_y_quantile *(gauge)*, 43
gauge_y_quantize *(gauge)*, 43
gauge_y_time *(gauge)*, 43
gauge_y_time_cat *(gauge)*, 43
gaugeViews, 45
get_gadm_data *(map_data)*, 53
get_gadm_data(), *30*
get_map_data *(map_data)*, 53
get_map_data(), *30*
get_world_map *(map_data)*, 53
get_world_map(), *30*
global_digits, 47
global_motif *(motif)*, 54

htmlwidgets::JS(), *46*

info, 47
info_abline *(info)*, 47
info_arc *(info)*, 47
info_data_region *(info)*, 47
info_hline *(info)*, 47
info_html *(info)*, 47
info_image *(info)*, 47
info_line *(info)*, 47
info_marker *(info)*, 47
info_region *(info)*, 47
info_region_filter *(info)*, 47
info_shape *(info)*, 47
info_text *(info)*, 47
info_vline *(info)*, 47
input_button, *5*, 49
input_button(), *42*
input_checkbox *(radio)*, 60
input_radio *(radio)*, 60

input_select, 49
input_select(), *3*, *4*
input_slider, 50
input_slider(), *3*
interplay, 50

layout_arc, 52
layout_igraph, 53
legend_asps *(*axis*)*, 10
legend_color *(*axis*)*, 10
legend_size *(*axis*)*, 10

map_data, 53
motif, 54
motif(), *16*
motif_as_list *(*motif*)*, 54
motif_from_json *(*motif*)*, 54
motif_from_json(), *55*
motif_from_list *(*motif*)*, 54
motif_from_list(), *55*

new_animation, 56

palettes, 56
planes, 59

qg2, 60

radio, 60
raster::getData(), *30*
register_interplay *(*interplay*)*, 50
remove_figure, 61
remove_interplay *(*interplay*)*, 50
render, 62
renderG2 *(*g2r-shiny*)*, 41

scrollbar, 62
selected *(*state*)*, 63
selected(), *19–22*, *24–39*
slider, 63
state, 63
subject, 64

template, 64
to_g2r, 66
tooltip, 65
tooltip(), *64*
tpl *(*template*)*, 64
tpl_item *(*template*)*, 64
tpl_item(), *64*