Package: gglite (via r-universe)

November 5, 2026

Title Lightweight Data Visualization via the Grammar of Graphics

Version 0.0.34

Description A lightweight R interface to the AntV G2 JavaScript visualization library with a ggplot2-style API. Supports rendering in litedown, R Markdown, Quarto, Jupyter notebooks (via the R kernel), Shiny, and standalone HTML previews.

License MIT + file LICENSE

URL https://github.com/yihui/gglite

BugReports https://github.com/yihui/gglite/issues

Depends R (>= 4.1.0)

Imports xfun (>= 0.59)

Suggests htmltools, litedown, knitr, repr, shiny, testit

VignetteBuilder litedown

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-11-01 03:11:07 UTC

RemoteUrl https://github.com/yihui/gglite

RemoteRef HEAD

RemoteSha 7fdf86da704273149fa51f75326a197c4fba4ea1

Contents

+.g2 2
animate 3
axis_ 4
canvas 5
chart_html 6

coord_ . . . 6
coord_transpose . . . 8
encode . . . 9
facet_circle . . . 9
facet_rect . . . 10
g2 . . . 11
g2Output . . . 12
interact . . . 13
knit_print.g2 . . . 14
labels.g2 . . . 14
legend_ . . . 15
mark_ . . . 16
print.g2 . . . 22
renderG2 . . . 23
scale_ . . . 23
scroll_ . . . 25
slider_ . . . 25
style_mark . . . 26
theme_ . . . 27
titles . . . 28
tooltip . . . 28
transform.g2 . . . 29

Index **32**

`+.g2` *Add a Modifier to a G2 Chart*

Description

Enables ggplot2-style + syntax for building charts. The right-hand side must be a deferred modifier created by calling a modifier function without a chart argument (e.g., `mark_point()`, `theme_dark()`).

Usage

```
## S3 method for class 'g2'
e1 + e2
```

Arguments

`e1`	A `g2` object (the chart).
`e2`	A `g2_mod` object (a deferred modifier).

Value

The modified `g2` object.

Examples

```
# These two are equivalent:
g2(mtcars, hp ~ mpg) |> mark_point() |> theme_dark()
g2(mtcars, hp ~ mpg) + mark_point() + theme_dark()
```

`animate` *Set Animation Options*

Description

Configure animation for the most recently added mark, or the automatically-inferred mark when none has been added explicitly. G2 supports `enter`, `update`, and `exit` animations.

Usage

```
animate(chart = NULL, ...)
```

Arguments

`chart`	A `g2` object.
`...`	Animation configuration as named lists. Use `enter`, `update`, and `exit` to control each phase. Set to `FALSE` to disable animation entirely.

Value

The modified `g2` object.

Examples

```
# Fade-in animation on bars
g2(data.frame(x = c('A', 'B', 'C'), y = c(3, 7, 2)), y ~ x) |>
  animate(enter = list(type = 'fadeIn', duration = 1000))

# Wave-in animation
g2(data.frame(x = c('A', 'B', 'C'), y = c(3, 7, 2)), y ~ x) |>
  animate(enter = list(type = 'waveIn', duration = 800))

# Disable animation
g2(mtcars, hp ~ mpg) |>
  animate(FALSE)
```

axis_ *Configure an Axis*

Description

Customize the axis for a positional channel (`'x'` or `'y'`). Set to `FALSE` to hide the axis. When called immediately after a `mark_*()` function (or `style_mark()`, `labels()`, etc.), the axis is applied to that mark only, enabling per-mark axis customization for dual-axis charts. Otherwise it applies at the chart level.

Usage

```
axis_(chart = NULL, channel, ...)

axis_x(chart = NULL, ...)

axis_y(chart = NULL, ...)
```

Arguments

chart	A g2 object.
channel	Positional channel: `'x'` or `'y'`.
...	Axis options such as `title`, `labelFormatter`, `tickCount`, `grid`, `position`, etc., or `FALSE` to hide.

Value

The modified g2 object.

Examples

```
# Chart-level axis titles
g2(mtcars, hp ~ mpg) |>
  axis_x(title = 'Miles per Gallon') |>
  axis_y(title = 'Horsepower')

# Dual-axis chart: each mark gets its own axis immediately after mark_*()
air = aggregate(cbind(Temp, Wind) ~ Month, data = airquality, FUN = mean)
air$Month = month.abb[air$Month]
g2(air, x = 'Month') |>
  mark_interval(encode = list(y = 'Temp')) |>
  scale_y(independent = TRUE) |>
  axis_y(title = 'Temperature (°F)') |>
  mark_line(encode = list(y = 'Wind')) |>
  scale_y(independent = TRUE) |>
  axis_y(position = 'right', grid = FALSE, title = 'Wind Speed (mph)')
```

canvas *Configure Canvas Options*

Description

Set chart dimensions, layout spacing, and renderer for a G2 chart. See https://pkg.yihui.org/gglite/examples/canvas.html for full examples.

Usage

```
canvas(
  chart = NULL,
  width = NULL,
  height = 480,
  padding = NULL,
  margin = NULL,
  inset = NULL,
  renderer = NULL,
  ...
)
```

Arguments

chart	A g2 object, or NULL to create a deferred modifier.
width	Width of the chart in pixels. NULL (default) enables auto-fit to the container width.
height	Height of the chart in pixels. Default is 480.
padding, margin, inset	Layout spacing in pixels. Each can be a scalar (applied to all sides) or a length-4 vector c(top, right, bottom, left); use NA to skip individual sides. NULL (the default) leaves the value unset.
renderer	The rendering backend: "Canvas" (default), "SVG", or "WebGL" (case-insensitive).
...	Additional top-level chart options passed to chart.options() in JavaScript (e.g., clip = TRUE).

Value

The modified g2 object (or a g2_mod when chart is NULL).

Examples

```
p = g2(mtcars, hp ~ mpg)
p |> canvas(width = 600, height = 400)
p |> canvas(padding = 30)
p |> canvas(renderer = 'svg')
```

chart_html *Generate Chart HTML*

Description

Create an HTML string containing a container and a <script> block that renders the chart using G2.

Usage

```
chart_html(chart, id = NULL, width = NULL, height = NULL)
```

Arguments

chart	A g2 object.
id	Optional container element ID. When NULL (default), a class-based container is used instead (more git-friendly output).
width, height	Optional CSS dimensions for the container.

Details

The global option gglite.defer_render controls whether chart rendering is deferred until the container scrolls into the viewport. Set options(gglite.defer_render = TRUE) to use the default threshold (0.5, i.e., 50% of the chart visible), or supply a numeric value between 0 and 1 to customize it, e.g., options(gglite.defer_render = 0.3). When enabled, enter animations fire when the reader first sees each chart instead of on page load. This is useful for demo or documentation pages. It is not typically needed for regular plots.

Value

A character string of HTML.

coord_ *Set the Coordinate System*

Description

Specify the coordinate system for the chart. G2 supports these coordinate types: 'cartesian' (default), 'polar', 'theta', 'radial', 'radar', 'helix', 'parallel'. Use the transform argument to apply coordinate transforms such as 'transpose' (equivalent to ggplot2's coord_flip()) or 'fisheye'.

Usage

```
coord_(chart = NULL, type, ...)

coord_polar(chart = NULL, ...)

coord_theta(chart = NULL, ...)

coord_radial(chart = NULL, ...)

coord_radar(chart = NULL, ...)

coord_helix(chart = NULL, ...)

coord_parallel(chart = NULL, ...)
```

Arguments

chart	A g2 object.
type	Coordinate type string.
...	Additional options such as innerRadius, outerRadius, startAngle, endAngle, or transform.

Details

The 'parallel' coordinate requires a position encoding (a character vector of column names) instead of separate x/y encodings. For radar charts, use the 'polar' coordinate with long-format data (x/y/color).

coord_polar(): Shortcut for coord_(chart, 'polar', ...).

coord_theta(): Shortcut for coord_(chart, 'theta', ...). Used for pie and donut charts.

coord_radial(): Shortcut for coord_(chart, 'radial', ...). Suitable for radial bar charts.

coord_radar(): Shortcut for coord_(chart, 'radar', ...). Used with position encoding for radar (spider) charts.

coord_helix(): Shortcut for coord_(chart, 'helix', ...).

coord_parallel(): Shortcut for coord_(chart, 'parallel', ...). Used with position encoding for parallel coordinate plots.

Value

The modified g2 object.

Examples

```
df = data.frame(x = c('A', 'B', 'C'), y = c(3, 7, 2))
p = g2(df, y ~ x, color = ~ x)
# Polar coordinate (rose chart)
g2(df, y ~ x) |> coord_polar()
```

```
# Theta coordinate (pie / donut chart)
p |> transform('stackY') |> coord_theta()
p |> transform('stackY') |> coord_theta(innerRadius = 0.5)

# Radial coordinate (radial bar chart)
p |> coord_radial()

# Parallel coordinate (uses position encoding)
g2(iris, position = names(iris)[-5], color = ~ Species,
  padding = c(30, NA, NA, NA)) |>
  coord_parallel() |>
  legend_color(position = 'bottom')

# Radar coordinate (polar with long-format data)
df2 = data.frame(
  item = rep(c('Design', 'Dev', 'Marketing', 'Sales', 'Support'), 2),
  score = c(80, 90, 65, 75, 85, 60, 70, 85, 80, 70),
  team = rep(c('A', 'B'), each = 5)
)
g2(df2, score ~ item, color = ~ team) |>
  mark_area(style = list(fillOpacity = 0.5)) |>
  mark_line(style = list(lineWidth = 2)) |>
  coord_polar() |>
  scale_x(padding = 0.5, align = 0) |>
  scale_y(domainMin = 0, domainMax = 100) |>
  axis_x(grid = TRUE)
```

`coord_transpose` *Transpose (Flip) the Coordinate System*

Description

Swap x and y axes, equivalent to ggplot2's `coord_flip()`. This adds a `transpose` transform to the current coordinate system.

Usage

```
coord_transpose(chart = NULL)
```

Arguments

`chart`	A g2 object.

Value

The modified g2 object.

Examples

```
# Horizontal bar chart (coord_flip equivalent)
df = data.frame(x = c('A', 'B', 'C'), y = c(3, 7, 2))
g2(df, y ~ x) |>
  coord_transpose()
```

---

encode *Set Aesthetic Mappings*

Description

Map data columns to visual channels (x, y, color, size, shape, etc.). Use formulas (e.g., `x = ~ col`) or character strings (e.g., `x = 'col'`).

Usage

```
encode(chart = NULL, ...)
```

Arguments

`chart`	A g2 object.
`...`	Named mappings as `name = ~column` formulas or character strings, e.g., `x = ~ col1, color = ~ col2`.

Value

The modified g2 object.

Examples

```
g2(mtcars) |> encode(x = ~ mpg, y = ~ hp)
```

---

facet_circle *Facet in a Circular Layout*

Description

Facet in a Circular Layout

Usage

```
facet_circle(chart = NULL, ...)
```

Arguments

`chart`	A g2 object.
`...`	Facet encoding and options. Pass `position = ~var` to specify the faceting variable. Character strings are also accepted.

Value

The modified g2 object.

Examples

```
g2(iris, Sepal.Length ~ Sepal.Width) |>
  facet_circle(position = ~ Species)
```

---

facet_rect *Facet by a Rectangular Grid*

Description

Split the chart into a grid of panels based on one or two variables, similar to ggplot2's `facet_grid()` / `facet_wrap()`. Use the `x` and/or `y` arguments to specify faceting variables. An unnamed first formula argument is interpreted as: `~var` sets `x = 'var'`; `y ~ x` sets both `x = 'x'` and `y = 'y'`.

Usage

```
facet_rect(chart = NULL, ...)
```

Arguments

chart	A g2 object.
...	Facet encoding and options. Pass `x = ~var` and/or `y = ~var` to specify the faceting variable(s). An unnamed formula `~var` sets `x = 'var'`; `y ~ x` sets `x = 'x'` and `y = 'y'`. Character strings are also accepted.

Value

The modified g2 object.

Examples

```
g2(iris, Sepal.Length ~ Sepal.Width) |>
  facet_rect(x = ~ Species)

# Unnamed formula shorthand
g2(iris, Sepal.Length ~ Sepal.Width) |>
  facet_rect(~ Species)

# Two-sided formula: y ~ x
df = data.frame(
  x = rnorm(200), y = rnorm(200),
  sex = sample(c('M', 'F'), 200, replace = TRUE),
  species = sample(c('A', 'B'), 200, replace = TRUE)
)
g2(df, y ~ x) |>
  facet_rect(sex ~ species)
```

g2 *Create a G2 Chart Object*

Description

Construct a base chart object, optionally with data and aesthetic mappings. Use R formulas for aesthetic mappings (see **Formula Interface**).

Usage

```
g2(data = NULL, ..., title = NULL, subtitle = NULL)
```

Arguments

data	A data frame, a ts/mts time series object, or NULL. Time series objects are automatically converted to data frames (with columns time and value for univariate series, or time, series, and value for multivariate series) and default aesthetic mappings are set accordingly. By default, only the columns actually referenced by the chart are included in the HTML output. Wrap data in I() to opt out of trimming and preserve all columns — useful when extra columns are accessed inside inline JavaScript functions that cannot be statically detected.
...	Aesthetic mappings as name = ~column formulas or a positional formula for x/y. Character strings are also accepted.
title	Chart title string, a convenient alternative to piping into titles() separately.
subtitle	Chart subtitle string.

Value

A g2 object (S3 class).

Formula Interface

Aesthetic mappings use R formulas as a shorthand for column names:

y ~ x Sets x and y as the first positional argument.

~ x Sets only x (e.g., for histograms or bar counts).

~ x1 + x2 + x3 Creates a position encoding with multiple fields (for parallel coordinates).

y ~ x | z Facets the chart by z (column direction).

y ~ x | z1 + z2 Facets by z1 (columns) and z2 (rows).

color = ~ var Maps the color channel to var (one-sided formula in a named argument). Works for any aesthetic channel.

Examples

```
g2(mtcars, hp ~ mpg)
g2(mtcars, hp ~ mpg, color = ~ cyl)
g2(mtcars, ~ mpg)

# Time series
g2(sunspot.year)
g2(EuStockMarkets)

# Title and subtitle
g2(mtcars, hp ~ mpg, title = 'Motor Trend Cars', subtitle = 'mpg vs hp')
```

---

g2Output *Shiny Output for a G2 Chart*

Description

Create a placeholder in the UI for a G2 chart rendered by `renderG2()`. The required JavaScript dependencies (G2 library, column-major data helper, and output binding) are automatically attached to the page.

Usage

```
g2Output(outputId, width = "100%", height = "480px")
```

Arguments

`outputId`	Output variable name to read the chart from.
`width, height`	CSS dimensions for the chart container.

Value

A Shiny UI element.

Examples

```
## Not run:
library(shiny)
ui = fluidPage(g2Output('chart1'))
server = function(input, output, session) {
  output$chart1 = renderG2({
    g2(mtcars, x = 'mpg', y = 'hp') |> mark_point()
  })
}
shinyApp(ui, server)

## End(Not run)
```

`interact` *Add an Interaction*

Description

Enable an interaction behaviour on the chart. G2 interaction types include: `'tooltip'`, `'elementHighlight'`, `'elementHighlightByX'`, `'elementHighlightByColor'`, `'elementSelect'`, `'elementSelectByX'`, `'elementSelectByColor'`, `'elementHoverScale'`, `'fisheye'`, `'chartIndex'`, `'legendFilter'`, `'legendHighlight'`, `'brushHighlight'`, `'brushXHighlight'`, `'brushYHighlight'`, `'brushFilter'`, `'brushXFilter'`, `'brushYFilter'`, `'sliderFilter'`, `'poptip'`, `'drillDown'`.

Usage

```
interact(chart = NULL, type, ...)
```

Arguments

`chart`	A g2 object.
`type`	Interaction type string.
`...`	Additional interaction options.

Value

The modified g2 object.

Examples

```
# Tooltip on scatter plot
g2(mtcars, hp ~ mpg) |>
  interact('tooltip')

# Highlight elements on hover
g2(mtcars, hp ~ mpg, color = ~ cyl) |>
  interact('elementHighlight')

# Brush to highlight a region
g2(mtcars, hp ~ mpg) |>
  interact('brushHighlight')

# Legend filter
g2(iris, Sepal.Length ~ Sepal.Width, color = ~ Species) |>
  interact('legendFilter')
```

knit_print.g2 *Custom Printing in Knitr*

Description

Custom Printing in Knitr

Usage

```
knit_print.g2(x, ...)
```

Arguments

x	A g2 object.
...	Ignored.

Value

A knit_asis character vector.

labels.g2 *Add Labels to the Last Mark*

Description

Append a label configuration to the most recently added mark. Can be called multiple times to add several label layers.

Usage

```
## S3 method for class 'g2'
labels(object, text, ...)

## S3 method for class 'character'
labels(object, ...)

## S3 method for class 'formula'
labels(object, ...)
```

Arguments

object	A g2 object, a formula ~col, or a character string (the text channel) for deferred use with +.
text	Label text channel: a column name as ~col or 'col'.
...	Additional label options such as position, formatter, style, or innerHTML.

Details

Use `text` (a column name as a formula `~col` or a string `'col'`) to map a data column to label text. G2 also supports `innerHTML`, which overrides `text` when provided.

Caution when using +: S3 dispatch requires that the first argument is unnamed. Use `p + labels(~col)` or `p + labels('col')` (not `p + labels(text = 'col')`). To use `innerHTML`, pass a dummy first argument: `p + labels('', innerHTML = ~col)`. See `?labels.g2` for the full API.

Value

The modified `g2` object (or a `g2_mod` when `object` is not a `g2`).

Examples

```
df = data.frame(x = c('A', 'B', 'C'), y = c(3, 7, 2))
g2(df, y ~ x) |>
  labels(text = ~ y, position = 'inside')

# With + operator (first argument must be unnamed)
g2(df, y ~ x) + labels(~ y)
g2(df, y ~ x) + labels('y', position = 'inside')
```

---

legend_ *Configure a Legend*

Description

Customize the legend for a visual channel (`'color'`, `'size'`, `'shape'`, `'opacity'`). Set to `FALSE` to hide.

Usage

```
legend_(chart = NULL, channel, ...)

legend_color(chart = NULL, ...)

legend_size(chart = NULL, ...)

legend_shape(chart = NULL, ...)

legend_opacity(chart = NULL, ...)
```

Arguments

`chart`	A `g2` object.
`channel`	Visual channel name.
`...`	Legend options such as `position` (`'top'`, `'bottom'`, `'left'`, `'right'`), `layout`, `title`, etc., or `FALSE` to hide.

Value

The modified g2 object.

Examples

```
p = g2(iris, Sepal.Length ~ Sepal.Width, color = ~ Species)
p |> legend_color(position = 'right')

g2(mtcars, hp ~ mpg, size = ~ wt) |>
  legend_size(position = 'bottom')
```

---

mark_ *Add a Geometry Layer (Mark)*

Description

Generic function to add a mark (geometry layer) to the chart. Use the specific `mark_*()` wrappers for convenience.

Usage

```
mark_(chart = NULL, type, ...)

mark_interval(chart = NULL, ...)

mark_line(chart = NULL, ...)

mark_point(chart = NULL, ...)

mark_area(chart = NULL, ...)

mark_rect(chart = NULL, ...)

mark_cell(chart = NULL, ...)

mark_text(chart = NULL, ...)

mark_path(chart = NULL, ...)

mark_polygon(chart = NULL, ...)

mark_image(chart = NULL, ...)

mark_link(chart = NULL, ...)

mark_line_x(chart = NULL, ...)
```

```
mark_line_y(chart = NULL, ...)

mark_range(chart = NULL, ...)

mark_range_x(chart = NULL, ...)

mark_range_y(chart = NULL, ...)

mark_connector(chart = NULL, ...)

mark_box(chart = NULL, ...)

mark_boxplot(chart = NULL, ...)

mark_beeswarm(chart, ...)

mark_density(chart = NULL, ...)

mark_heatmap(chart = NULL, ...)

mark_vector(chart = NULL, ...)

mark_node(chart = NULL, ...)

mark_edge(chart = NULL, ...)

mark_sankey(chart = NULL, ...)

mark_chord(chart = NULL, ...)

mark_treemap(chart = NULL, ...)

mark_pack(chart = NULL, ...)

mark_force_graph(chart = NULL, ...)

mark_tree(chart = NULL, ...)

mark_word_cloud(chart = NULL, ...)

mark_gauge(chart = NULL, ...)

mark_liquid(chart = NULL, ...)

mark_shape(chart = NULL, ...)

mark_sunburst(chart = NULL, ...)
```

```
mark_partition(chart = NULL, ...)
```

Arguments

chart	A g2 object.
type	Character string for the G2 mark type.
...	Mark-level options passed to G2, such as data, encode, transform, style, animate, labels, tooltip, axis, legend. When data is a data frame, only columns referenced by the chart or mark-level encodings are kept. Wrap it in I() to preserve all columns.

Details

mark_interval(): Add an interval mark for bar and column charts.

mark_line(): Add a line mark (connects points sorted by x).

mark_point(): Add a point mark (scatter plot).

mark_area(): Add an area mark (filled region under the line).

mark_rect(): Draw rectangles. Commonly used with a bin transform for 2-D histograms.

mark_cell(): Draw rectangular cells, commonly used for heatmaps and calendar charts.

mark_text(): Place text labels at data coordinates.

mark_path(): Connect points in data order (unlike line, which sorts by x).

mark_polygon(): Draw filled polygon shapes.

mark_image(): Place images at data coordinates. Requires an src encoding for image URLs.

mark_link(): Draw links (lines) between pairs of points.

mark_line_x(): Draw a vertical reference line at a given x position.

mark_line_y(): Draw a horizontal reference line at a given y position.

mark_range(): Shade a rectangular region defined by x and y intervals.

mark_range_x(): Shade a vertical band defined by an x interval.

mark_range_y(): Shade a horizontal band defined by a y interval.

mark_connector(): Draw a connector line with optional labels between two data points.

mark_box(): Draw pre-computed box elements (for custom box plots).

mark_boxplot(): A composite mark that automatically computes box plot statistics (median, quartiles, whiskers) from raw data.

mark_beeswarm(): Display individual data points using force simulation to avoid overlapping. Particularly useful for visualizing distributions within categories.

mark_density(): Visualize probability density using kernel density estimation (KDE). When the chart has a numeric x aesthetic, the KDE transform and encodings are configured automatically: the density is computed for the x column, and if color is also mapped, separate density curves are drawn for each group. Explicit data/encode in ... bypass this auto-configuration.

mark_heatmap(): A composite mark for rendering heatmaps from point data.

mark_vector(): Draw arrows or vectors. Useful for wind or flow field visualizations.

`mark_node()`: Used in graph visualizations together with `mark_edge()`.

`mark_edge()`: Used in graph visualizations together with `mark_node()`.

`mark_sankey()`: Draw a Sankey diagram. Data should have `source`, `target`, and `value` columns.

`mark_chord()`: Draw a chord diagram. Data should have `source`, `target`, and `value` columns.

`mark_treemap()`: Draw a treemap layout. Data should be hierarchical (a nested list with `name`, `value`, and optionally `children` fields).

`mark_pack()`: Draw a circle packing layout. Data format is the same as `mark_treemap()`.

`mark_force_graph()`: Draw a force-directed graph layout.

`mark_tree()`: Draw a tree layout.

`mark_word_cloud()`: Draw a word cloud.

`mark_gauge()`: Draw a gauge (speedometer) chart.

`mark_liquid()`: Draw a liquid fill gauge.

`mark_shape()`: A custom mark whose rendering is controlled by a JavaScript render function.

`mark_sunburst()`: A composite mark for sunburst (radial partition) visualization of hierarchical data. This wraps the partition mark with polar coordinates. The data should be a nested tree structure wrapped in a list, e.g., `data = list(type = 'inline', value = list(tree))`.

`mark_partition()`: A composite mark for hierarchical icicle chart visualization. For a radial (sunburst) layout, use `mark_sunburst()` instead. The data should be a nested tree structure wrapped in a list, e.g., `data = list(type = 'inline', value = list(tree))`.

Value

The modified g2 object.

Examples

```
# Bar chart
df = data.frame(x = c('A', 'B', 'C'), y = c(3, 7, 2))
g2(df, y ~ x) |> mark_interval()

# Stacked bar chart
df = data.frame(
  x = rep(c('A', 'B'), each = 2), y = c(3, 2, 5, 4),
  color = rep(c('a', 'b'), 2)
)
g2(df, y ~ x, color = ~ color) |>
  mark_interval() |> transform('stackY')

# Line and area chart
df = data.frame(x = 1:5, y = c(3, 1, 4, 1, 5))
p = g2(df, y ~ x)
p |> mark_line()

# Scatter plot
g2(mtcars, hp ~ mpg, color = ~ cyl) |> mark_point()

# Area chart
```

```
p |> mark_area()

# 2-D histogram using bin transform
g2(mtcars, hp ~ mpg) |>
  mark_rect(
    transform = list(list(type = 'bin', thresholdsX = 10, thresholdsY = 10))
  )

# Heatmap cells
df = expand.grid(x = LETTERS[1:4], y = LETTERS[1:4])
df$value = seq_len(nrow(df))
g2(df, y ~ x, color = ~ value) |> mark_cell()

# Bar chart with text labels
df = data.frame(x = c('A', 'B', 'C'), y = c(3, 7, 2))
g2(df, y ~ x) |>
  mark_interval() |>
  mark_text(encode = list(text = 'y'))

# Spiral path (points connected in data order)
n = 100
t = seq(0, 4 * pi, length.out = n)
df = data.frame(x = t * cos(t), y = t * sin(t))
g2(df, y ~ x) |> mark_path()

# Triangle polygon
df = data.frame(x = c(0, 1, 0.5), y = c(0, 0, 1))
g2(df, y ~ x) |> mark_polygon()

# Image mark at two data points
df = data.frame(x = 1:2, y = 1:2)
g2(df, y ~ x) |>
  mark_image(style = list(
   src = 'https://gw.alipayobjects.com/mdn/rms_dfc253/afts/img/A*SZGfRaFPkIoAAAAAAAAAAAAAARQnAQ'
  ))

# Link mark connecting pairs of points
df = data.frame(x = c(0, 1), y = c(0, 0), x1 = c(1, 2), y1 = c(1, 1))
g2(df) |>
  mark_link(encode = list(x = c('x', 'x1'), y = c('y', 'y1')))

# Scatter plot with reference lines and shaded regions
p = g2(mtcars, hp ~ mpg) |> mark_point()
p |> mark_line_x(data = list(list(x = 20)),
  style = list(stroke = 'red', lineDash = c(4, 4)))
p |> mark_line_y(data = list(list(y = 150)),
  style = list(stroke = 'red', lineDash = c(4, 4)))
p |> mark_range(
  data = list(list(x = c(15, 25), y = c(100, 200))),
  style = list(fill = 'steelblue', fillOpacity = 0.15)
)
p |> mark_range_x(data = list(list(x = c(15, 25))),
  style = list(fill = 'steelblue', fillOpacity = 0.15))
```

```
p |> mark_range_y(data = list(list(y = c(100, 200))),
  style = list(fill = 'orange', fillOpacity = 0.15))

# Bar chart with connector annotation
df = data.frame(x = c('A', 'B'), y = c(3, 7))
g2(df, y ~ x) |>
  mark_interval() |>
  mark_connector(
    data = list(list(x = 'A', x1 = 'B')),
    encode = list(x = 'x', x1 = 'x1'),
    labels = list(list(text = '+133%'))
  )

# Box plot and beeswarm
p = g2(iris, Sepal.Width ~ Species)
p |> mark_boxplot()
p |> mark_beeswarm()

# Density plot by species
g2(iris, ~ Sepal.Width, color = ~ Species) |> mark_density()

# Heatmap from point data
g2(iris, Sepal.Length ~ Sepal.Width, color = ~ Petal.Length) |>
  mark_heatmap()

# Sankey and chord diagrams
df = data.frame(
  source = c('A', 'A', 'B'), target = c('B', 'C', 'C'),
  value = c(5, 3, 2)
)
g2(df) |>
  mark_sankey(
    encode = list(source = 'source', target = 'target', value = 'value'),
    layout = list(nodeAlign = 'center')
  )
g2(df) |>
  mark_chord(
    encode = list(source = 'source', target = 'target', value = 'value')
  )

# Treemap and circle packing
tree_data = list(
  name = 'root', children = list(
    list(name = 'A', value = 10),
    list(name = 'B', value = 20),
    list(name = 'C', value = 15)
  )
)
g2() |>
  mark_treemap(
    data = list(value = tree_data),
    encode = list(value = 'value')
  )
```

```
g2() |>
  mark_pack(
    data = list(value = tree_data),
    encode = list(value = 'value', color = 'name')
  )

# Word cloud
df = data.frame(
  text = c('Hello', 'Data', 'Science', 'R', 'G2', 'Chart'),
  value = c(30, 25, 20, 15, 10, 5)
)
g2(df) |>
  mark_word_cloud(encode = list(text = 'text', value = 'value', color = 'text'))

# Liquid fill gauge
g2() |>
  mark_liquid(data = list(list(value = 0.3)),
    encode = list(y = 'value'),
    style = list(textContent = '30%'))

# Sunburst chart (hierarchical radial partition)
tree = list(name = 'root', children = list(
  list(name = 'A', value = 10, children = list(
    list(name = 'A1', value = 5), list(name = 'A2', value = 5)
  )),
  list(name = 'B', value = 20)
))
g2() |> mark_sunburst(
  data = list(type = 'inline', value = list(tree)),
  encode = list(value = 'value')
)
```

`print.g2` *Preview a Chart in the Viewer or Browser*

Description

Preview a Chart in the Viewer or Browser

Usage

```
## S3 method for class 'g2'
print(x, ...)
```

Arguments

`x`	A g2 object.
`...`	Additional arguments passed to `chart_html()`.

Value

The chart object (invisibly).

`renderG2` *Render a G2 Chart in Shiny*

Description

Create a reactive G2 chart for use with `g2Output()`. Assign it to an output slot: `output$ID = renderG2({...})`.

Usage

```
renderG2(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`expr`	An expression that returns a `g2` object.
`env`	The environment in which to evaluate `expr`.
`quoted`	Whether `expr` is already quoted.

Value

A render function for use with `g2Output()`.

`scale_` *Configure a Scale*

Description

Add or modify scale settings for a given aesthetic channel. When called immediately after a `mark_*()` function (or after `style_mark()`, `labels()`, etc. that target the last mark), the scale is applied to that mark only. Otherwise it is applied at the chart level and affects all marks. This context-sensitivity enables dual-axis charts: pipe `scale_y()` right after each mark to give it its own independent y scale.

Usage

```
scale_(chart = NULL, field, ...)

scale_x(chart = NULL, ...)

scale_y(chart = NULL, ...)

scale_color(chart = NULL, ...)

scale_size(chart = NULL, ...)

scale_shape(chart = NULL, ...)

scale_opacity(chart = NULL, ...)
```

Arguments

chart	A g2 object.
field	Character string naming the channel (e.g., 'x', 'y', 'color').
...	Scale options passed to G2 (e.g., type = 'log', nice = TRUE, domain, range, zero = TRUE).

Details

G2 scale types: 'linear', 'ordinal', 'band', 'point', 'time', 'log', 'pow', 'sqrt', 'threshold', 'quantize', 'quantile', 'sequential', 'identity', 'constant'.

Value

The modified g2 object.

Examples

```
p = g2(mtcars, hp ~ mpg)

# Log-scaled x axis
p |> scale_x(type = 'log')

# Square-root y axis
p |> scale_y(type = 'sqrt')

# Ordinal color palette
g2(iris, Sepal.Length ~ Sepal.Width, color = ~ Species) |>
  scale_color(palette = 'category10')
```

scroll_ *Add a Scrollbar*

Description

Add a scrollbar to a positional channel for zooming/panning.

Usage

```
scroll_(chart = NULL, channel, ...)

scroll_x(chart = NULL, ...)

scroll_y(chart = NULL, ...)
```

Arguments

chart	A g2 object.
channel	Positional channel: 'x' or 'y'.
...	Scrollbar options.

Value

The modified g2 object.

Examples

```
df = data.frame(x = 1:100, y = cumsum(rnorm(100)))
p = g2(df, y ~ x) |> mark_line()
p |> scroll_x() |> scroll_y()
```

slider_ *Add a Slider*

Description

Add a range slider to a positional channel for zooming/panning.

Usage

```
slider_(chart = NULL, channel, ...)

slider_x(chart = NULL, ...)

slider_y(chart = NULL, ...)
```

Arguments

chart	A g2 object.
channel	Positional channel: 'x' or 'y'.
...	Slider options.

Value

The modified g2 object.

Examples

```
p = g2(mtcars, hp ~ mpg)
p |> slider_x() |> slider_y()
```

---

style_mark *Set Style on the Last Mark*

Description

Set Style on the Last Mark

Usage

```
style_mark(chart = NULL, ...)
```

Arguments

chart	A g2 object.
...	Style options such as fill, stroke, lineWidth, fillOpacity, strokeOpacity.

Value

The modified g2 object.

Examples

```
g2(mtcars, hp ~ mpg) |>
  style_mark(fill = 'steelblue', stroke = 'white', lineWidth = 1)
```

theme_ *Set the Chart Theme*

Description

G2 built-in themes: `'classic'` (default), `'classicDark'`, `'light'`, `'dark'`, `'academy'`.

Usage

```
theme_(chart = NULL, type, ...)

theme_classic(chart = NULL, ...)

theme_classic_dark(chart = NULL, ...)

theme_light(chart = NULL, ...)

theme_dark(chart = NULL, ...)

theme_academy(chart = NULL, ...)
```

Arguments

chart	A g2 object.
type	Theme name string or a list of custom theme options. Use the specific wrappers (`theme_classic()`, `theme_dark()`, etc.) instead.
...	Additional theme options merged with the type.

Details

To set global theme options for all charts, use `options(gglite.theme = list(...))`. This is useful for changing default font sizes, grid line visibility, and other theme properties without modifying each chart individually. For example:

```
options(gglite.theme = list(
  title = list(titleFontSize = 20),
  axis = list(labelFontSize = 16, gridStrokeOpacity = 0.3),
  legendCategory = list(itemLabelFontSize = 14)
))
```

Per-chart `theme_()` settings are merged on top of the global option.

Value

The modified g2 object.

Examples

```
p = g2(mtcars, hp ~ mpg)
p |> theme_classic()
p |> theme_dark()
p |> theme_academy()
```

`titles` *Set the Chart Title*

Description

Set the chart title and subtitle, as well as their styles.

Usage

```
titles(chart = NULL, main, ...)
```

Arguments

`chart`	A `g2` object passed via `|>`, or `NULL` when using `+`.
`main`	Title text string.
`...`	Additional title options such as `subtitle`, `align`, `style`.

Value

The modified `g2` object.

Examples

```
g2(mtcars, hp ~ mpg) |>
  titles('Motor Trend Cars', subtitle = 'mpg vs hp')
```

`tooltip` *Configure the Tooltip*

Description

Configure tooltip interaction behavior. All options are applied to `interaction.tooltip` in the G2 spec: pass `FALSE` to disable the tooltip, or pass named options such as `crosshairs`, `shared`, `marker`, and any `crosshairs*`/`marker*` style properties. To configure the data displayed in a tooltip for a specific mark (e.g., `channel`, `valueFormatter`, `items`), pass a `tooltip` list argument directly to the mark function instead, e.g., `mark_line(tooltip = list(channel = 'y', valueFormatter = '.0%'))`.

Usage

```
tooltip(chart = NULL, ...)
```

Arguments

chart	A g2 object.
...	Tooltip interaction options such as shared, crosshairs, marker, series, crosshairsStroke, crosshairsLineWidth, crosshairsStrokeOpacity, or FALSE to disable the tooltip. Series marks (mark_line(), mark_area()) show a vertical crosshair by default (crosshairsY = TRUE); pass crosshairs = FALSE to suppress it.

Value

The modified g2 object.

Examples

```
# Enable crosshairs (works best with line/area marks which use series tooltip)
df = data.frame(x = 1:6, y = c(3, 1, 4, 1, 5, 2))
g2(df, y ~ x) |>
  mark_line() |>
  tooltip(crosshairs = TRUE)

# Shared tooltip for multi-series line chart
df2 = data.frame(
  x = rep(1:5, 2), y = c(3, 1, 4, 1, 5, 2, 7, 1, 8, 3),
  group = rep(c('A', 'B'), each = 5)
)
g2(df2, y ~ x, color = ~ group) |>
  mark_line() |>
  tooltip(shared = TRUE)

# Disable tooltip
g2(mtcars, hp ~ mpg) |>
  tooltip(FALSE)
```

---

transform.g2 *Add a Data Transform to the Last Mark*

Description

Append a data transform to the most recently added mark. G2 transforms correspond roughly to ggplot2's stat_*() and position_*() functions.

Usage

```
## S3 method for class 'g2'
transform(`_data`, type, ...)

## S3 method for class 'character'
transform(`_data`, ...)
```

Arguments

_data	A g2 object (via |>) or a transform type string (for deferred use with +).
type	Transform type string (e.g., 'stackY', 'dodgeX').
...	Additional transform options (e.g., thresholds, y).

Details

Common transforms: 'stackY' (stack, like position_stack()), 'dodgeX' (dodge, like position_dodge()), 'normalizeY' (normalize to 100%, like position_fill()), 'jitterX' / 'jitterY' (jitter, like position_jitter()), 'bin' / 'binX' (bin data, like stat_bin()), 'groupX' / 'groupY' / 'groupColor' (group and aggregate, like stat_summary()), 'sortX' / 'sortY' / 'sortColor' (sort data), 'symmetryY' (mirror, for pyramid / funnel charts), 'diffY' (difference, for waterfall charts), 'select' / 'selectX' / 'selectY' (filter), 'sample' (down-sample), 'pack' (circle-packing layout), 'flexX' (flexible x spacing, Marimekko charts).

Caution when using +: S3 dispatch requires that the first argument is unnamed. Use p + transform('stackY') (not p + transform(type = 'stackY')). See ?transform.g2 for the full API.

Value

The modified g2 object (or a g2_mod when _data is a string).

Examples

```
# Stacked bar chart
df = data.frame(
  x = rep(c('A', 'B'), each = 2), y = c(3, 2, 5, 4),
  color = rep(c('a', 'b'), 2)
)
g2(df, y ~ x, color = ~ color) |>
  mark_interval() |>
  transform('stackY')

# Grouped (dodged) bar chart
g2(df, y ~ x, color = ~ color) |>
  mark_interval() |>
  transform('dodgeX')

# Percent stacked bar (normalizeY + stackY)
g2(df, y ~ x, color = ~ color) |>
  mark_interval() |>
  transform('stackY') |>
```

```
  transform('normalizeY')

# Jitter on a scatter plot
g2(mtcars, hp ~ cyl) |>
  transform('jitterX')

# Histogram using binX
g2(mtcars, ~ mpg) |>
  mark_interval(encode = list(y = 'count')) |>
  transform('binX', thresholds = 15)

# With + operator (first argument must be unnamed)
g2(df, y ~ x, color = ~ color) |> mark_interval() + transform('stackY')
```

Index

+.g2, 2

animate, 3
axis_, 4
axis_x *(axis_)*, 4
axis_y *(axis_)*, 4

canvas, 5
chart_html, 6
chart_html(), *22*
coord_, 6
coord_helix *(coord_)*, 6
coord_parallel *(coord_)*, 6
coord_polar *(coord_)*, 6
coord_radar *(coord_)*, 6
coord_radial *(coord_)*, 6
coord_theta *(coord_)*, 6
coord_transpose, 8

encode, 9

facet_circle, 9
facet_rect, 10

g2, 11
g2Output, 12
g2Output(), *23*

I(), *11*, *18*
interact, 13

knit_print.g2, 14

labels.character *(labels.g2)*, 14
labels.formula *(labels.g2)*, 14
labels.g2, 14
legend_, 15
legend_color *(legend_)*, 15
legend_opacity *(legend_)*, 15
legend_shape *(legend_)*, 15
legend_size *(legend_)*, 15

mark_, 16
mark_area *(mark_)*, 16
mark_beeswarm *(mark_)*, 16
mark_box *(mark_)*, 16
mark_boxplot *(mark_)*, 16
mark_cell *(mark_)*, 16
mark_chord *(mark_)*, 16
mark_connector *(mark_)*, 16
mark_density *(mark_)*, 16
mark_edge *(mark_)*, 16
mark_edge(), *19*
mark_force_graph *(mark_)*, 16
mark_gauge *(mark_)*, 16
mark_heatmap *(mark_)*, 16
mark_image *(mark_)*, 16
mark_interval *(mark_)*, 16
mark_line *(mark_)*, 16
mark_line_x *(mark_)*, 16
mark_line_y *(mark_)*, 16
mark_link *(mark_)*, 16
mark_liquid *(mark_)*, 16
mark_node *(mark_)*, 16
mark_node(), *19*
mark_pack *(mark_)*, 16
mark_partition *(mark_)*, 16
mark_path *(mark_)*, 16
mark_point *(mark_)*, 16
mark_polygon *(mark_)*, 16
mark_range *(mark_)*, 16
mark_range_x *(mark_)*, 16
mark_range_y *(mark_)*, 16
mark_rect *(mark_)*, 16
mark_sankey *(mark_)*, 16
mark_shape *(mark_)*, 16
mark_sunburst *(mark_)*, 16
mark_sunburst(), *19*
mark_text *(mark_)*, 16
mark_tree *(mark_)*, 16
mark_treemap *(mark_)*, 16

mark_vector *(mark_)*, 16
mark_word_cloud *(mark_)*, 16

print.g2, 22

renderG2, 23
renderG2(), *12*

scale_, 23
scale_color *(scale_)*, 23
scale_opacity *(scale_)*, 23
scale_shape *(scale_)*, 23
scale_size *(scale_)*, 23
scale_x *(scale_)*, 23
scale_y *(scale_)*, 23
scroll_, 25
scroll_x *(scroll_)*, 25
scroll_y *(scroll_)*, 25
slider_, 25
slider_x *(slider_)*, 25
slider_y *(slider_)*, 25
style_mark, 26

theme_, 27
theme_academy *(theme_)*, 27
theme_classic *(theme_)*, 27
theme_classic_dark *(theme_)*, 27
theme_dark *(theme_)*, 27
theme_light *(theme_)*, 27
titles, 28
titles(), *11*
tooltip, 28
transform.character *(transform.g2)*, 29
transform.g2, 29