Package ‘ggiraph’

October 31, 2019

Type  Package
Title  Make 'ggplot2' Graphics Interactive
Description  Create interactive 'ggplot2' graphics using 'htmlwidgets'.
Version  0.7.0
License  GPL-3
Copyright  See file COPYRIGHTS.
Encoding  UTF-8
Imports  grid, ggplot2 (>= 3.2.0), htmlwidgets (>= 0.6), stats, xml2
  (>= 1.0.0), htmltools, Rcpp (>= 0.12.12), gdtools (>= 0.2.1), purrr, rlang, uuid
LinkingTo  Rcpp, gdtools
Suggests  knitr, testthat, rmarkdown, maps, shiny, sf (>= 0.7-3), quantreg, dplyr
VignetteBuilder  knitr
URL  https://davidgohel.github.io/ggiraph
BugReports  https://github.com/davidgohel/ggiraph/issues
RoxygenNote  6.1.1
Collate  'RcppExports.R' 'ipar.R' 'utils_ggplot2_performance.R'
  'utils_ggplot2.R' 'utils.R' 'annotate_interactive.R'
  'annotation_raster_interactive.R' 'dsvg.R' 'dsvg_view.R'
  'element_interactive.R' 'fonts.R' 'geom_abline_interactive.R'
  'geom_path_interactive.R' 'geom_polygon_interactive.R'
  'geom_rect_interactive.R' 'geom_bar_interactive.R'
  'geom_boxplot_interactive.R' 'geom_col_interactive.R'
  'geom_contour_interactive.R' 'geom_crossbar_interactive.R'
  'geom_density_2d_interactive.R' 'geom_density_interactive.R'
  'geom_errorbar_interactive.R' 'geom_errorbarh_interactive.R'
  'geom_freqpoly_interactive.R' 'geom_histogram_interactive.R'
  'geom_hline_interactive.R' 'geom_jitter_interactive.R'
  'geom_label_interactive.R' 'geom_linerange_interactive.R'
  'geom_map_interactive.R' 'geom_point_interactive.R'
R topics documented:

annotate_interactive ............................................. 4
annotation_raster_interactive .............................. 5
dsvg ................................................................. 6
dsvg_view ............................................................ 7
element_interactive .................................................. 7
gem_abline_interactive ............................................. 9
gem_bar_interactive ................................................. 12
gem_boxplot_interactive .......................................... 13
gem_contour_interactive .......................................... 14
gem_crossbar_interactive ......................................... 15
gem_density_2d_interactive .................................... 16
gem_density_interactive .......................................... 18
gem_errorbarh_interactive ...................................... 19
gem_freqpoly_interactive ...................................... 20
gem_jitter_interactive ........................................... 21
R topics documented:

geom_label_interactive .................................................. 22
geom_map_interactive ..................................................... 24
geom_path_interactive .................................................... 25
geom_point_interactive ................................................... 28
geom_polygon_interactive ............................................... 29
geom_quartile_interactive .............................................. 30
geom_raster_interactive .................................................. 31
geom_rect_interactive .................................................... 32
geom_ribbon_interactive ................................................ 34
geom_segment_interactive ............................................... 36
geom_sf_interactive ...................................................... 37
geom_smooth_interactive ................................................. 38
ggiraph ................................................................. 40
ggiraphOutput ............................................................ 41
girafe ................................................................. 42
girafeOutput .............................................................. 44
girafe_css ................................................................. 44
girafe_options ............................................................ 45
guide_colourbar_interactive ............................................. 46
guide_legend_interactive ................................................ 49
interactive_parameters .................................................. 53
interactive_path_grob .................................................... 55
interactive_points_grob ................................................ 56
interactive_polygon_grob ............................................... 57
interactive_polyline_grob .............................................. 57
interactive_raster_grob ................................................ 58
interactive_rect_grob ................................................... 59
interactive_roundrect_grob ............................................ 59
interactive_segments_grob ............................................. 60
interactive_text_grob ................................................... 61
label_interactive ......................................................... 61
opts_hover .............................................................. 62
opts_selection ........................................................... 63
opts_sizing .............................................................. 64
opts_toolbar ............................................................ 65
opts_tooltip ............................................................. 66
opts_zoom .............................................................. 67
renderggiraph ........................................................... 68
renderGirafe ............................................................. 68
run_girafe_example ....................................................... 69
scale_alpha_interactive ................................................ 69
scale_colour_brewer_interactive ...................................... 70
scale_colour_interactive ............................................... 72
scale_gradient_interactive ............................................. 73
scale_linetype_interactive ............................................ 77
scale_manual_interactive .............................................. 78
scale_shape_interactive ............................................... 81
scale_size_interactive ................................................. 83
Create interactive annotations

Description

The layer is based on `annotate`. See the documentation for that function for more details.

Usage

`annotate_interactive(...)`

Arguments

`...` arguments passed to base function, plus any of the `interactive_parameters`.

Details for `annotate_*_interactive` functions

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also

`girafe`

Examples

```r
# add interactive annotation to a ggplot -------
library(ggplot2)
library(ggiraph)

gg <- ggplot(mtcars, aes(x = disp, y = qsec )) +
  geom_point(size=2) +
  annotate_interactive(
    "rect", xmin = 100, xmax = 400, fill = "red",
    data_id = "an_id", tooltip = "a tooltip",
    ymin = 18, ymax = 20, alpha = .5)

x <- girafe(ggobj = gg, width_svg = 5, height_svg = 4)
if( interactive() ) print(x)
```
Create interactive raster annotations

Description
The layer is based on `annotation_raster`. See the documentation for that function for more details.

Usage
`annotation_raster_interactive(...)`

Arguments
`...` arguments passed to base function, plus any of the `interactive_parameters`.

Details for `annotate_*_interactive` functions
The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also
`girafe`

Examples
```
# add interactive raster annotation to a ggplot -------
library(ggplot2)
library(ggiraph)

# Generate data
rainbow <- matrix(hcl(seq(0, 360, length.out = 50 * 50), 80, 70), nrow = 50)
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  annotation_raster_interactive(rainbow, 15, 20, 3, 4, tooltip = "I am an image!")
x <- girafe(ggobj = p)
if( interactive() ) print(x)

# To fill up whole plot
p <- ggplot(mtcars, aes(mpg, wt)) +
  annotation_raster_interactive(rainbow, -Inf, Inf, -Inf, Inf, tooltip = "I am an image too!") +
  geom_point()
x <- girafe(ggobj = p)
if( interactive() ) print(x)
```
dsvg

SVG Graphics Driver

Description

This function produces SVG files (compliant to the current W3 SVG XML standard) where elements can be made interactive.

Usage

dsvg(file = "Rplots.svg", width = 6, height = 6, bg = "white",
     pointsize = 12, standalone = TRUE, canvas_id = "svg_1",
     fonts = list())

Arguments

file the file where output will appear.
width, height Height and width in inches.
bg Default background color for the plot (defaults to "white").
pointsize default point size.
standalone Produce a stand alone svg file? If FALSE, omits xml header and default namespace.
canvas_id svg id within HTML page.
fonts Named list of font names to be aliased with fonts installed on your system. If unspecified, the R default families sans, serif, mono and symbol are aliased to the family returned by match_family().

See Also

Devices

Examples

dsvg()
plot(rnorm(10), main="Simple Example", xlab = "", ylab = "")
dev.off()
dsvg_view

Run plotting code and view svg in RStudio Viewer or web browser.

Description
This is useful primarily for testing. Requires the htmltools package.

Usage
dsvg_view(code, ...)

Arguments
code Plotting code to execute.
...

Other arguments passed on to dsvg.

Examples
dsvg_view(plot(1:10))
dsvg_view(hist(rnorm(100)))

element_interactive Create interactive theme elements

Description
With these functions the user can add interactivity to various theme elements.

They are based on element_rect, element_line and element_text. See the documentation for
those functions for more details.

Usage
element_line_interactive(...)
element_rect_interactive(...)element_text_interactive(...)

Arguments
...

arguments passed to base function, plus any of the interactive_parameters.
Details for element_*_interactive functions

The interactive parameters can be supplied as arguments in the relevant function and they should be scalar values.

For theme text elements (element_text_interactive), the interactive parameters can also be supplied while setting a label value, via the labs family of functions or when setting a scale/guide title or key label. Instead of setting a character value for the element, function label_interactive can be used to define interactive parameters to go along with the label. When the parameters are supplied that way, they override the default values that are set at the theme via element_text_interactive or via the guide’s theme parameters.

See Also
girafe

Examples

# add interactive theme elements -------
library(ggplot2)
library(ggiraph)


# plots
gg_point = ggplot(data = dataset) + geom_point_interactive(aes( 
x = wt,
y = qsec,
color = disp,
tooltip = carname,
data_id = carname 
)) + theme_minimal() + theme( 
plot.title = element_text_interactive( 
data_id = "plot.title",
tooltip = "plot title",
hover_css = "fill:red;stroke:none;font-size:12pt"
),
plot.subtitle = element_text_interactive( 
data_id = "plot.subtitle",
tooltip = "plot subtitle",
hover_css = "fill:none;"
),
axis.title.x = element_text_interactive( 
data_id = "axis.title.x",
tooltip = "Description for x axis",

geom_abline_interactive

Create interactive reference lines

Description

These geometries are based on geom_abline, geom_hline and geom_vline. See the documentation for those functions for more details.

Usage

geom_abline_interactive(...)

geom_hline_interactive(...)  

geom_vline_interactive(...)  

Arguments

...  

arguments passed to base function, plus any of the interactive_parameters.
Details for `geom_*_interactive` functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

See Also

- `girafe`
- `girafe`
if( interactive() ) print(x)
}

dataset <- data.frame(
x = c(1, 2, 5, 6, 8),
y = c(3, 6, 2, 8, 7),
vx = c(1, 1.5, 0.8, 0.5, 1.3),
vy = c(0.2, 1.3, 1.7, 0.8, 1.4),
)
dataset$clickjs <- rep(paste0("alert(\"", mean(dataset$y), \"\")", 5)
g2 <- ggplot(dataset, aes(x = year, y = y)) + geom_point() + geom_line()
gg_hline2 <- g2 + geom_hline_interactive(
aes(yintercept = mean(y),
tooltip = round(mean(y), 2),
data_id = y, onclick = clickjs))
x <- girafe(ggobj = gg_hline2)
if( interactive() ) print(x)

# add vertical interactive reference lines to a ggplot -------
library(ggplot2)
library(ggiraph)

if (requireNamespace("dplyr", quietly = TRUE)) {
g1 <- ggplot(diamonds, aes(carat)) + geom_histogram()
gg_vline1 <- g1 + geom_vline_interactive(
aes(xintercept = mean(carat),
tooltip = round(mean(carat), 2),
data_id = carat), size = 3)
x <- girafe(ggobj = gg_vline1)
if( interactive() ) print(x)
}
dataset <- data.frame(x = rnorm(100))
dataset$clickjs <- rep(paste0("alert(\"",
round(mean(dataset$x), 2), "\")", 100)
g2 <- ggplot(dataset, aes(x)) + geom_density(fill = "#000000", alpha = 0.7)
gg_vline2 <- g2 + geom_vline_interactive(
aes(xintercept = mean(x), tooltip = round(mean(x), 2),
data_id = x, onclick = clickjs), color = "white")
x <- girafe(ggobj = gg_vline2)
```
x <- girafe_options(x = x,
                    opts_hover(css = "cursor:pointer;fill:orange;stroke:orange;") )
if( interactive() ) print(x)
```

---

**geom_bar_interactive  Create interactive bars**

**Description**

The geometries are based on `geom_bar` and `geom_col`. See the documentation for those functions for more details.

**Usage**

```r
geom_bar_interactive(...)
geom_col_interactive(...)
```

**Arguments**

```r
...
```

arguments passed to base function, plus any of the `interactive_parameters`.

**Details for geom_*_interactive functions**

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see `layer`). In this way they can be set to a scalar value.

**See Also**

`girafe`

**Examples**

```r
# add interactive bar -------
library(ggplot2)
library(ggiraph)

p <- ggplot(mpg, aes( x = class, tooltip = class,
                      data_id = class ) ) +
    geom_bar_interactive()

x <- girafe(ggobj = p)
if( interactive() ) print(x)

dat <- data.frame( name = c( "David", "Constance", "Leonie" ),
```

```r
```
The geometry is based on \texttt{geom_boxplot}. See the documentation for those functions for more details.

\textbf{Usage}

\texttt{geom_boxplot\_interactive(\ldots)}

\textbf{Arguments}

\texttt{\ldots} \quad arguments passed to base function, plus any of the \texttt{interactive\_parameters}.

\textbf{Details for geom\_\_*\_interactive functions}

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via \texttt{aes}). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom\_\_*\_interactive function (see \texttt{layer}). In this way they can be set to a scalar value.
# geom_contour_interactive

Create interactive 2d contours of a 3d surface

## Description

The geometry is based on `geom_contour`. See the documentation for those functions for more details.

## Usage

```r
geom_contour_interactive(...)```

## Arguments

`...` arguments passed to base function, plus any of the `interactive_parameters`.

## Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see `layer`). In this way they can be set to a scalar value.

---

**Examples**

```r
# add interactive boxplot -------
library(ggplot2)
library(ggiraph)

p <- ggplot(mpg,
  aes(x = class, y = hwy, tooltip = class)) +
  geom_boxplot_interactive()

x <- girafe(ggobj = p)
if( interactive() ) print(x)

p <- ggplot(mpg, aes(x = drv, y = hwy, tooltip = class, fill = class, data_id=class)) +
  geom_boxplot_interactive(outlier.colour = "red") +
  guides(fill = "none") + theme_minimal()

x <- girafe(ggobj = p)
if( interactive() ) print(x)
```
geom_crossbar_interactive

See Also

girafe
girafe

Examples

# add interactive contours to a ggplot -------
library(ggplot2)
library(ggiraph)

v <- ggplot(faithfuld, aes(waiting, eruptions, z = density))
p <- v + geom_contour_interactive(aes(
  colour = stat(level),
  tooltip = paste("Level:" , stat(level))
))
x <- girafe(ggobj = p)
if (interactive()) print(x)

geom_crossbar_interactive

Create interactive vertical intervals: lines, crossbars & errorbars

Description

These geometries are based on geom_crossbar, geom_errorbar, geom_linerange and geom_pointrange. See the documentation for those functions for more details.

Usage

geom_crossbar_interactive(...)  
geom_errorbar_interactive(...)  
geom_linerange_interactive(...)  
geom_pointrange_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via aes). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see layer). In this way they can be set to a scalar value.
See Also

girafe

Examples

# add interactive intervals -------
library(ggplot2)
library(ggiraph)

# Create a simple example dataset
df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
  upper = c(1.1, 5.3, 3.3, 4.2),
  lower = c(0.8, 4.6, 2.4, 3.6)
)

p <- ggplot(df, aes(trt, resp, colour = group))
g <- p + geom_linerange_interactive(aes(ymin = lower, ymax = upper, tooltip = group))
x <- girafe(ggobj = g)
if( interactive() ) print(x)

g <- p + geom_pointrange_interactive(aes(ymin = lower, ymax = upper, tooltip = group))
x <- girafe(ggobj = g)
if( interactive() ) print(x)

g <- p + geom_crossbar_interactive(aes(ymin = lower, ymax = upper, tooltip = group), width = 0.2)
x <- girafe(ggobj = g)
if( interactive() ) print(x)

g <- p + geom_errorbar_interactive(aes(ymin = lower, ymax = upper, tooltip = group), width = 0.2)
x <- girafe(ggobj = g)
if( interactive() ) print(x)

geom_density_2d_interactive

Create interactive contours of a 2d density estimate

Description

The geometry is based on geom_density_2d. See the documentation for those functions for more details.

Usage

geom_density_2d_interactive(...)
Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via \texttt{aes}). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see \texttt{layer}). In this way they can be set to a scalar value.

See Also

\texttt{girafe}
\texttt{girafe}

Examples

# add interactive contours to a ggplot -------
library(ggplot2)
library(ggiraph)

m <- ggplot(faithful, aes(x = eruptions, y = waiting)) +
  geom_point_interactive(aes(tooltip = paste("Waiting: ", waiting, 
  "eruptions: ", eruptions))) +
  xlim(0.5, 6) +
  ylim(40, 110)
p <- m + geom_density_2d_interactive(aes(tooltip = paste("Level: ", stat(level))))
x <- girafe(ggobj = p)
if (interactive()) print(x)

set.seed(4393)
dsmall <- diamonds[sample(nrow(diamonds), 1000),]
d <- ggplot(dsmall, aes(x, y))

p <- d + geom_density_2d_interactive(aes(colour = cut, tooltip = cut, data_id = cut))
x <- girafe(ggobj = p)
x <- girafe_options(x = x,
  opts_hover(css = "stroke:red;stroke-width:3px;" ) )
if (interactive()) print(x)

p <- d + stat_density_2d(aes(fill = stat(nlevel),
  tooltip = paste("nlevel: ", stat(nlevel))),
  geom = "interactive_polygon") +
  facet_grid( . ~ cut) + scale_fill_viridis_c_interactive(tooltip = "nlevel")
x <- girafe(ggobj = p)
if (interactive()) print(x)
geom_density_interactive

Create interactive smoothed density estimates

Description

The geometry is based on `geom_density`. See the documentation for those functions for more details.

Usage

`geom_density_interactive(…)`

Arguments

`...` arguments passed to base function, plus any of the `interactive_parameters`.

Details for `geom_*_interactive` functions

The interactive parameters can be supplied in two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

See Also

girafe

Examples

```r
# add interactive bar --------
library(ggplot2)
library(ggiraph)

p <- ggplot(diamonds, aes(carat)) +
  geom_density_interactive(tooltip="density", data_id="density")
x <- girafe(ggobj = p)
x <- girafe_options(x = x,
  opts_hover(css = "stroke:orange;stroke-width:3px;"))
if(interactive()) print(x)

p <- ggplot(diamonds, aes(depth, fill = cut, colour = cut)) +
  geom_density_interactive(aes(tooltip=cut, data_id=cut), alpha = 0.1) +
  xlim(55, 70)
x <- girafe(ggobj = p)
x <- girafe_options(x = x,
  opts_hover(css = "stroke:yellow;stroke-width:3px;fill-opacity:0.8;"))
```
if( interactive() ) print(x)

p <- ggplot(diamonds, aes(carat, fill = cut)) +
  geom_density_interactive(aes(tooltip=cut, data_id=cut), position = "stack")
x <- girafe(ggobj = p)
if( interactive() ) print(x)

p <- ggplot(diamonds, aes(carat, stat(count), fill = cut)) +
  geom_density_interactive(aes(tooltip=cut, data_id=cut), position = "fill")
x <- girafe(ggobj = p)
if( interactive() ) print(x)

---

gem_errorbarh_interactive

Create interactive horizontal error bars

Description
This geometry is based on geom_errorbarh. See the documentation for those functions for more details.

Usage
geom_errorbarh_interactive(...)

Arguments
...

Arguments passed to base function, plus any of the interactive_parameters.

Details for geom_*_interactive functions
The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via aes). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see layer). In this way they can be set to a scalar value.

See Also
girafe
Examples

```r
# add horizontal error bars -------
library(ggplot2)
library(ggiraph)

df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
  se = c(0.1, 0.3, 0.3, 0.2)
)

# Define the top and bottom of the errorbars
p <- ggplot(df, aes(resp, trt, colour = group))
g <- p + geom_point() +
  geom_errorbarh_interactive(aes(xmax = resp + se, xmin = resp - se, tooltip = group))
x <- girafe(ggobj = g)
if( interactive() ) print(x)

g <- p + geom_point() +
  geom_errorbarh_interactive(aes(xmax = resp + se, xmin = resp - se, height = .2, tooltip = group))
x <- girafe(ggobj = g)
if( interactive() ) print(x)
```

---

**geom_freqpoly_interactive**

Create interactive histograms and frequency polygons

Description

The geometries are based on `geom_histogram` and `geom_freqpoly`. See the documentation for those functions for more details.

This interactive version is only providing a single tooltip per group of data (same for data_id). It means it is only possible to associate a single tooltip to a set of bins.

Usage

```
geom_freqpoly_interactive(...)

geom_histogram_interactive(...)
```

Arguments

```
... arguments passed to base function, plus any of the interactive_parameters.
```
geom_jitter_interactive

Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see `layer`). In this way they can be set to a scalar value.

See Also

girafe

Examples

# add interactive histogram -------
library(ggplot2)
library(ggiraph)

p <- ggplot(diamonds, aes(carat)) +
  geom_histogram_interactive(bins=30, aes(tooltip = ..count..,
                                          data_id = carat) )
x <- girafe(ggobj = p)
if( interactive() ) print(x)

p <- ggplot(diamonds, aes(price, colour = cut, tooltip = cut, data_id = cut)) +
  geom_freqpoly_interactive(binwidth = 500)
x <- girafe(ggobj = p)
x <- girafe_options(x = x,
                     opts_hover(css = "stroke-width:3px;") )
if( interactive() ) print(x)

geom_jitter_interactive

Description

The geometry is based on geom_jitter. See the documentation for those functions for more details.

Usage

geom_jitter_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.
Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see `layer`). In this way they can be set to a scalar value.

See Also

`girafe`

Examples

```r
# add interactive paths to a ggplot --------
library(ggplot2)
library(ggiraph)

gg_jitter <- ggplot(mpg, aes(cyl, hwy,
    tooltip = paste(manufacturer, model, year, trans, sep = "\n")))+
  geom_jitter_interactive()

x <- girafe(ggobj = gg_jitter)
if( interactive() ) print(x)
```

---

`geom_label_interactive`

Create interactive textual annotations

Description

The geometries are based on `geom_text` and `geom_label`. See the documentation for those functions for more details.

Usage

```r
geom_label_interactive(...)
```

Arguments

```r
... arguments passed to base function, plus any of the interactive_parameters.
```
Details for `geom_*_interactive` functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

See Also

`girafe`

Examples

# add interactive labels to a ggplot -------
library(ggplot2)
library(ggiraph)

```r
p <- ggplot(mtcars, aes(wt, mpg, label = rownames(mtcars))) +
  geom_label_interactive(aes(tooltip = paste(rownames(mtcars), mpg, sep = "\n")))
x <- girafe(ggobj = p)
if( interactive() ) print(x)
```

```r
p <- ggplot(mtcars, aes(wt, mpg, label = rownames(mtcars))) +
  geom_label_interactive(aes(fill = factor(cyl),
                            tooltip = paste(rownames(mtcars), mpg, sep = "\n"),
                            colour = "white",
                            fontface = "bold")
                        )
x <- girafe(ggobj = p)
if( interactive() ) print(x)
```

# add interactive texts to a ggplot -------
library(ggplot2)
library(ggiraph)

```r
## the data
dataset = mtcars
dataset$label = row.names(mtcars)
dataset$tooltip = paste0("cyl: ", dataset$cyl, "<br/>",
                          "gear: ", dataset$gear, "<br/>",
                          "carb: ", dataset$carb)

## the plot
gg_text = ggplot(dataset,
  aes(x = mpg, y = wt, label = label,
       color = qsec,
       tooltip = tooltip, data_id = label ) ) +
  geom_text_interactive() +
  coord_cartesian(xlim = c(0,50))
```
## geom_map_interactive

Create interactive polygons from a reference map

**Description**

The geometry is based on `geom_map`. See the documentation for those functions for more details.

**Usage**

`geom_map_interactive(...)`

**Arguments**

`...` arguments passed to base function, plus any of the `interactive_parameters`.

**Details for geom_*_interactive functions**

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see `layer`). In this way they can be set to a scalar value.

**See Also**

`girafe`

**Examples**

```r
## display the plot
x <- girafe(ggobj = gg_text)
x <- girafe_options(x = x,
                    opts_hover(css = "fill:#FF4C3B;font-style:italic;" )
if( interactive() ) print(x)
```

```r
# add interactive maps to a ggplot -------
library(ggplot2)
library(ggiraph)

# create tooltips and onclick events
states_ <- sprintf("<p>%s</p>",
                  as.character(crimes$state )

table_ <- paste0("\n	\n\t"\n\t\n\t\t<\font\">\n\t| UrbanPop |<\table\>
\t\t\t<\font\">\n\t| UrbanPop |<\table\>\n\t\t\t<\font\">\n\t| UrbanPop |<\table\>\n```

```
 geom_path_interactive

Create interactive observations connections

Description

These geometries are based on geom_path, geom_line and geom_step. See the documentation for those functions for more details.

Usage

geom_path_interactive(...)

geom_line_interactive(...)

geom_step_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.
Details for `geom_*_interactive` functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

See Also

girafe

Examples

```r
# add interactive paths to a ggplot -------
library(ggplot2)
library(ggiraph)

# geom_line_interactive example -----
if( requireNamespace("dplyr", quietly = TRUE)){
  gg <- ggplot(economics_long,
          aes(date, value01, colour = variable, tooltip = variable, data_id = variable,
              hover_css = "fill:none;")) +
    geom_line_interactive(size = .75)
  x <- girafe(ggobj = gg)
  x <- girafe_options(x = x,
                    opts_hover(css = "stroke:red;fill:orange") )
  if( interactive() ) print(x)
}

# geom_step_interactive example -----
if( requireNamespace("dplyr", quietly = TRUE)){
  recent <- economics[ economics$date > as.Date("2013-01-01"), ]
  gg = ggplot(recent, aes(date, unemploy)) +
    geom_step_interactive(aes(tooltip = date, data_id = date))
  x <- girafe(ggobj = gg)
  x <- girafe_options(x = x,
                    opts_hover(css = "stroke:red;"))
  if( interactive() ) print(x)
}

# create datasets ----
id = paste0("id", 1:10)
data = expand.grid(list(
    id = id
  )
)
groups = sample(LETTERS[1:3], size = length(id), replace = TRUE)
data$group = groups[match(data$id, id)]
data$value = runif(n = nrow(data))
```
data$tooltip = paste0('line ', data$id )
data$onclick = paste0("alert("", data$id, ")")

cols = c("orange", "orange1", "orange2", "navajowhite4", "navy")
dataset2 <- data.frame(x = rep(1:20, 5),
y = rnorm(100, 5, .2) + rep(1:5, each=20),
z = rep(1:20, 5),
grp = factor(rep(1:5, each=20)),
color = factor(rep(1:5, each=20)),
label = rep(paste0("id ", 1:5 ), each=20),
onclick = paste0( "alert("", sample(letters, 100, replace = TRUE), ",")")
)

# plots ---
gg_path_1 = ggplot(data, aes(variable, value, group = id, colour = group, tooltip = tooltip, onclick = onclick, data_id = id)) + geom_path_interactive(alpha = 0.5)

rgg_path_2 = ggplot(data, aes(variable, value, group = id, data_id = id, tooltip = tooltip)) + geom_path_interactive(alpha = 0.5) + facet_wrap(~ group )

rgg_path_3 = ggplot(dataset2) + geom_path_interactive(aes(x, y, group=grp, data_id = label, color = color, tooltip = label, onclick = onclick), size = 1 )

# ggiraph widgets ---
x <- girafe(ggobj = gg_path_1)
x <- girafe_options(x = x,
opts_hover(css = "stroke-width:3px;") )
if( interactive() ) print(x)

x <- girafe(ggobj = gg_path_2)
x <- girafe_options(x = x,
opts_hover(css = "stroke:orange;stroke-width:3px;") )
if( interactive() ) print(x)

x <- girafe(ggobj = gg_path_3)
x <- girafe_options(x = x,
opts_hover(css = "stroke-width:10px;") )
if( interactive() ) print(x)

m <- ggplot(economics, aes(unemploy/pop, psavert))
p <- m + geom_path_interactive(aes(colour = as.numeric(date), tooltip=date))
x <- girafe(ggobj = p)
if( interactive() ) print(x)
Description

The geometry is based on `geom_point`. See the documentation for those functions for more details.

Usage

`geom_point_interactive(...)`

Arguments

`...` arguments passed to base function, plus any of the `interactive_parameters`.

Details for `geom_*_interactive` functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

Note

The following shapes id 3, 4 and 7 to 14 are composite symbols and should not be used.

See Also

girafe

Examples

```r
# add interactive points to a ggplot -------
library(ggplot2)
library(ggiraph)

dataset <- structure(list(qsec = c(16.46, 17.02, 18.61, 19.44, 17.02, 20.22),
                        disp = c(160, 160, 108, 258, 360, 225),
                        carname = c("Mazda RX4", "Mazda RX4 Wag", "Datsun 710", "Hornet 4 Drive", "Hornet Sportabout", "Valiant"),
                        wt = c(2.62, 2.875, 2.32, 3.215, 3.44, 3.46)),
row.names = c("Mazda RX4", "Mazda RX4 Wag", "Datsun 710", "Hornet 4 Drive", "Hornet Sportabout", "Valiant"),
class = "data.frame")
dataset

# plots
gg_point = ggplot(data = dataset) +
```

geom_polygon_interactive

Create interactive polygons

Description

The geometry is based on `geom_polygon`. See the documentation for those functions for more details.

Usage

```r
geom_polygon_interactive(...)
```

Arguments

`...` arguments passed to base function, plus any of the `interactive_parameters`.

Details for `geom_*_interactive` functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

See Also

`girafe`

Examples

```r
# add interactive polygons to a ggplot --------
library(ggplot2)
library(ggiraph)

# create data
ids <- factor(c("1.1", "2.1", "1.2", "2.2", "1.3", "2.3"))

values <- data.frame(id = ids,
value = c(3, 3.1, 3.1, 3.2, 3.15, 3.5))
positions <- data.frame(

x <- girafe(ggobj = gg_point)
if( interactive() ) print(x)
```
### geom_quantile_interactive

Create interactive quantile regression

#### Description

The geometry is based on `geom_quantile`. See the documentation for those functions for more details.

#### Usage

```r
geom_quantile_interactive(...)
```
Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via aes). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see layer). In this way they can be set to a scalar value.

See Also

girafe

girafe

Examples

# add interactive quantiles to a ggplot -------
library(ggplot2)
library(ggiraph)

if (requireNamespace("quantreg", quietly = TRUE)) {
  m <- ggplot(mpg, aes(displ, 1 / hwy)) + geom_point()
  p <- m + geom_quantile_interactive(
    aes(
      tooltip = stat(quantile),
      data_id = stat(quantile),
      colour = stat(quantile)
    ),
    formula = y ~ x,
    size = 2,
    alpha = 0.5
  )
  x <- girafe(ggobj = p)
  x <- girafe_options(x = x,
    opts_hover(css = "stroke:red;stroke-width:10px;")
  )
  if (interactive()) print(x)
}

geom_raster_interactive

Create interactive raster rectangles

Description

The geometry is based on geom_raster. See the documentation for those functions for more details.
geom_rect_interactive

Usage

geom_raster_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via aes). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see layer). In this way they can be set to a scalar value.

See Also

girafe

Examples

# add interactive raster to a ggplot -----
library(ggplot2)
library(ggiraph)

df <- expand.grid(x = 0:5, y = 0:5)
df$z <- runif(nrow(df))

gg <- ggplot(df, aes(x, y, fill = z, tooltip = "tooltip")) +
geom_raster_interactive() +
scale_fill_gradient_interactive(
  data_id = "coco", onclick = "cici", tooltip = "cucu"
)

x <- girafe(ggobj = gg)
if( interactive() ) print(x)

geom_rect_interactive  Create interactive rectangles

Description

These geometries are based on geom_rect and geom_tile. See the documentation for those functions for more details.
Usage

```r
geom_rect_interactive(...)  
geom_tile_interactive(...)```

Arguments

```r
... arguments passed to base function, plus any of the interactive_parameters.
```

Details for `geom_*_interactive` functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

Note

Converting a raster to svg elements could inflate dramatically the size of the svg and make it unreadable in a browser. Function `geom_tile_interactive` should be used with caution, total number of rectangles should be small.

See Also

`girafe`

Examples

```r
# add interactive polygons to a ggplot -------
library(ggplot2)
library(ggiraph)

dataset = data.frame(  
x1 = c(1, 3, 1, 5, 4),
x2 = c(2, 4, 3, 6, 6),
y1 = c(1, 1, 4, 1, 3),
y2 = c(2, 2, 5, 3, 5),
t = c('a', 'a', 'a', 'b', 'b'),
r = c(1, 2, 3, 4, 5),  
tooltip = c("ID 1", "ID 2", "ID 3", "ID 4", "ID 5"),
uid = c("ID 1", "ID 2", "ID 3", "ID 4", "ID 5"),
oc = rep("alert(this.getAttribute("data-id\"))", 5)  
)

gg_rect = ggplot() +  
  scale_x_continuous(name="x") +  
  scale_y_continuous(name="y") +  
  geom_rect_interactive(data=dataset,  
    mapping = aes(xmin = x1, xmax = x2,  
    ymin = y1, ymax = y2, fill = t,
geom_ribbon_interactive

Create interactive ribbons and area plots
Description

The geometries are based on `geom_ribbon` and `geom_area`. See the documentation for those functions for more details.

Usage

```r
geom_ribbon_interactive(...)
geom_area_interactive(...)
```

Arguments

```r
... arguments passed to base function, plus any of the interactive_parameters.
```

Details for `geom_*_interactive` functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the `geom_*_interactive` function (see `layer`). In this way they can be set to a scalar value.

See Also

`girafe`

Examples

```r
# add interactive bar -------
library(ggplot2)
library(ggiraph)

# Generate data
huron <- data.frame(year = 1875:1972, level = as.vector(LakeHuron))
h <- ggplot(huron, aes(year))

# Add interactive ribbon

# With aesthetics

# add interactive bar -------
g <- h +
  geom_ribbon_interactive(aes(ymin = level - 1, ymax = level + 1),
                          fill = "grey70", tooltip = "ribbon1", data_id="ribbon1",
                          hover_css = "stroke:red;stroke-width:inherit;") +
  geom_line_interactive(aes(y = level), tooltip = "level", data_id="line1",
                        hover_css = "stroke:orange;fill:none;")

# With plain arguments

x <- girafe(ggobj = g)
x <- girafe_options(x = x, 
                    opts_hover(css = "fill:blue;stroke:orange;stroke-width:3px;"))
if( interactive() ) print(x)

# Add interactive area

# With aesthetics

g <- h + geom_area_interactive(aes(y = level), tooltip = "area1")
x <- girafe(ggobj = g)
```
if( interactive() ) print(x)

geom_segment_interactive

Create interactive line segments

Description
The geometry is based on geom_segment. See the documentation for those functions for more details.

Usage
geom_segment_interactive(...)

Arguments
...

arguments passed to base function, plus any of the interactive_parameters.

Details for geom_*_interactive functions
The interactive parameters can be supplied with two ways:

• As aesthetics with the mapping argument (via aes). In this way they can be mapped to data columns and apply to a set of geometries.
• As plain arguments into the geom_*_interactive function (see layer). In this way they can be set to a scalar value.

See Also
girafe

Examples
# add interactive segments to a ggplot -------
library(ggplot2)
library(ggiraph)

counts <- as.data.frame(table(x = rpois(100,5)))
counts$x <- as.numeric( as.character(counts$x) )
counts$xlab <- paste0("bar",as.character(counts$x) )

gg_segment_1 <- ggplot(data = counts, aes(x = x, y = Freq, yend = 0, xend = x, tooltip = xlab ) ) +
geom_segment_interactive( size = I(10) )
x <- girafe(ggobj = gg_segment_1)
if( interactive() ) print(x)

dataset = data.frame(x=c(1,2,5,6,8),
y=c(3,6,2,8,7),
vx=c(1,1.5,0.8,0.5,1.3),
vy=c(0.2,1.3,1.7,0.8,1.4),
labs = paste0("Lab", 1:5))
dataset$clickjs = paste0("alert(\"",dataset$labs, "\")")

gg_segment_2 = ggplot() +
    geom_segment_interactive(data=dataset, mapping=aes(x=x, y=y,
        xend=x+vx, yend=y+vy, tooltip = labs, onclick=clickjs ),
        arrow=grid::arrow(length = grid::unit(0.03, "npc")),
        size=2, color="blue") +
    geom_point(data=dataset, mapping=aes(x=x, y=y),
        size=4, shape=21, fill="white")

x <- girafe(ggobj = gg_segment_2)
if( interactive() ) print(x)

---

**geom_sf_interactive**

Create interactive sf objects

**Description**

These geometries are based on `geom_sf`, `geom_sf_label` and `geom_sf_text`. See the documentation for those functions for more details.

**Usage**

`geom_sf_interactive(...)`

`geom_sf_label_interactive(...)`

`geom_sf_text_interactive(...)`

**Arguments**

... arguments passed to base function, plus any of the interactive_parameters.

**Details for geom_*_interactive functions**

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see `layer`). In this way they can be set to a scalar value.

**See Also**

`girafe`
Examples

```r
# add interactive sf objects to a ggplot -------
library(ggplot2)
library(ggiraph)

## original code: see section examples of ggplot2::geom_sf help file
if (requireNamespace("sf", quietly = TRUE, versionCheck = c(op = "\geq", version = "0.7-3"))) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  gg <- ggplot(nc) +
    geom_sf_interactive(aes(fill = AREA, tooltip = NAME, data_id = NAME))
  x <- girafe(ggobj = gg)
  if( interactive() ) print(x)

  nc_3857 <- sf::st_transform(nc, "+init=epsg:3857")

  # Unfortunately if you plot other types of feature you'll need to use
  # show.legend to tell ggplot2 what type of legend to use
  nc_3857$mid <- sf::st_centroid(nc_3857$geometry)
  gg <- ggplot(nc_3857) +
    geom_sf(colour = "white") +
    geom_sf_interactive(aes(geometry = mid,
                              size = AREA, tooltip = NAME, data_id = NAME),
                             show.legend = "point")
  x <- girafe( ggobj = gg)
  if( interactive() ) print(x)

  # Example with texts.
  gg <- ggplot(nc_3857[1:3, ]) +
    geom_sf(aes(fill = AREA)) +
    geom_sf_text_interactive(aes(label = NAME, tooltip = NAME), color="white")
  x <- girafe( ggobj = gg)
  if( interactive() ) print(x)

  # Example with labels.
  gg <- ggplot(nc_3857[1:3, ]) +
    geom_sf(aes(fill = AREA)) +
    geom_sf_label_interactive(aes(label = NAME, tooltip = NAME))
  x <- girafe( ggobj = gg)
  if( interactive() ) print(x)
}
```

---

**geom_smooth_interactive**

Create interactive smoothed conditional means

---

Description

The geometry is based on `geom_smooth`. See the documentation for those functions for more details.
Usage

geom_smooth_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via aes). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see layer). In this way they can be set to a scalar value.

See Also

girafe

Examples

# add interactive bar -------
library(ggplot2)
library(ggiraph)

p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth_interactive(aes(tooltip="smoothed line", data_id="smooth"))
x <- girafe(ggobj = p)
x <- girafe_options(x = x,
  opts_hover(css = "stroke:orange;stroke-width:3px;" ) )
if( interactive() ) print(x)

p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth_interactive(method = lm, se = FALSE, tooltip="smooth", data_id="smooth")
x <- girafe(ggobj = p)
if( interactive() ) print(x)

p <- ggplot(mpg, aes(displ, hwy, colour = class, tooltip = class, data_id = class)) +
  geom_point_interactive() +
  geom_smooth_interactive(se = FALSE, method = lm)
x <- girafe(ggobj = p)
x <- girafe_options(x = x,
  opts_hover(css = "stroke:red;stroke-width:3px;" ) )
if( interactive() ) print(x)
Create a ggiraph object

Description

Create an interactive graphic to be used in a web browser.

This function is maintained for backward compatibility reasons, user should now use function girafe and girafe_options.

Usage

```
ggiraph(code, ggobj = NULL, pointsize = 12, width = 0.75,
width_svg = 6, height_svg = 5, tooltip_extra_css = NULL,
hover_css = NULL, tooltip_opacity = 0.9, tooltip_offx = 10,
tooltip_offy = 0, tooltip_zindex = 999, zoom_max = 1,
selection_type = "multiple", selected_css = NULL, dep_dir = NULL,
xml_reader_options = list(), ...)```

Arguments

- **code**: Plotting code to execute
- **ggobj**: ggplot object to print. Argument code will be ignored if this argument is supplied.
- **pointsize**: the default pointsize of plotted text in pixels, default to 12.
- **width**: widget width ratio (0 < width <= 1).
- **width_svg**: The width and height of the graphics region in inches. The default values are 6 and 5 inches. This will define the aspect ratio of the graphic as it will be used to define viewbox attribute of the SVG result.
- **height_svg**: The width and height of the graphics region in inches. The default values are 6 and 5 inches. This will define the aspect ratio of the graphic as it will be used to define viewbox attribute of the SVG result.
- **tooltip_extra_css**: extra css (added to `position: absolute; pointer-events: none;`) used to customize tooltip area.
- **hover_css**: css to apply when mouse is hover and element with a data-id attribute.
- **tooltip_opacity**: tooltip opacity
- **tooltip_offx**: tooltip x offset
- **tooltip_offy**: tooltip y offset
- **tooltip_zindex**: tooltip css z-index, default to 999.
- **zoom_max**: maximum zoom factor
- **selection_type**: row selection mode ("single", "multiple", "none") when widget is in a Shiny application.
- **selected_css**: css to apply when element is selected (shiny only).
**DEPRECATED**

The path where the output files are stored. If NULL, the current path for temporary files is used.

- **xml_reader_options**
  read_xml additional arguments to be used when parsing the svg result. This feature can be used to parse huge svg files by using `list(options = "HUGE")` but this is not recommended.

... arguments passed on to `dsvg`

### Examples

```r
# ggiraph simple example -------
library(ggplot2)
library(ggiraph)

dataset

# plots
gg_point = ggplot(data = dataset) + geom_point_interactive(aes(x = wt, y = qsec, color = disp, tooltip = carname, data_id = carname)) + theme_minimal()

x <- girafe(ggobj = gg_point)
if( interactive() ) print(x)
```

---

**ggiraphOutput**

Create a ggiraph output element

### Description

Render a ggiraph within an application page.

### Usage

`ggiraphOutput(outputId, width = "100\%", height = "500px")`

### Arguments

- **outputId** output variable to read the ggiraph from.
- **width** widget width
- **height** widget height
Examples

```r
## Not run:
if( require(shiny) && interactive() ){
  app_dir <- file.path( system.file(package = "ggiraph"), "examples/shiny/cars" )
  shinyAppDir(appDir = app_dir )
}
if( require(shiny) && interactive() ){
  app_dir <- file.path( system.file(package = "ggiraph"), "examples/shiny/crimes" )
  shinyAppDir(appDir = app_dir )
}
## End(Not run)
```

---

**girafe**  
*Create a girafe object*

---

**Description**

Create an interactive graphic with a ggplot object to be used in a web browser. The function should replace function ggiraph.

**Usage**

```r
girafe(code, ggobj = NULL, pointsize = 12, width_svg = 6, height_svg = 5, xml_reader_options = list(), options = list(), ...)
```

**Arguments**

- `code`  
  Plotting code to execute

- `ggobj`  
  ggplot objet to print. argument code will be ignored if this argument is supplied.

- `pointsize`  
  the default pointsize of plotted text in pixels, default to 12.

- `width_svg`, `height_svg`  
  The width and height of the graphics region in inches. The default values are 6 and 5 inches. This will define the aspect ratio of the graphic as it will be used to define viewbox attribute of the SVG result.

- `xml_reader_options`  
  read_xml additional arguments to be used when parsing the svg result. This feature can be used to parse huge svg files by using `list(options = "HUGE")` but this is not recommended.

- `options`  
  a list of options for girafe rendering, see `opts_tooltip`, `opts_hover`, `opts_selection`, ...

- `...`  
  arguments passed on to `dsvg`
Details

Use `geom_zzz_interactive` to create interactive graphical elements.

Difference from original functions is that some extra aesthetics are understood: the `interactive_parameters`.

Tooltips can be displayed when mouse is over graphical elements.

If id are associated with points, they get animated when mouse is over and can be selected when used in shiny apps.

On click actions can be set with javascript instructions. This option should not be used simultaneously with selections in Shiny applications as both features are "on click" features.

When a zoom effect is set, "zoom activate", "zoom desactivate" and "zoom init" buttons are available in a toolbar.

When selection type is set to 'multiple' (in Shiny applications), lasso selection and lasso anti-selections buttons are available in a toolbar.

Widget options

`girafe` animations can be customized with function `girafe_options`. Options are available to customize tooltips, hover effects, zoom effects selection effects and toolbar.

Widget sizing

`girafe` graphics are responsive, which mean, they will be resized according to their container. There are two responsive behavior implementations: one for Shiny applications and flexdashboard documents and one for other documents (i.e. R markdown and `saveWidget`).

Graphics are created by an R graphic device (i.e pdf, png, svg here) and need arguments width and height to define a graphic region. Arguments `width_svg` and `height_svg` are used as corresponding values. They are defining the aspect ratio of the graphic. This proportion is always respected when the graph is displayed.

When a girafe graphic is in a Shiny application, graphic will be resized according to the arguments width and height of the function `girafeOutput`. Default values are '100%' and '500px'. These arguments determine the outer bounding box of the graphic (the HTML element that will contain the graphic with an aspect ratio).

When a girafe graphic is in an R markdown document (producing an HTML document), the graphic will be resized according to the argument `width` of the function `girafe`. Its value is being used to define a relative width of the graphic within its HTML container. Its height is automatically adjusted regarding to the argument `width` and the aspect ratio.

If this behavior does not fit with your need, I recommend you to use package `widgetframe` that wraps htmlwidgets inside a responsive iframe.

See Also

`girafe_options`
Examples

```r
library(ggplot2)

dataset <- mtcars
dataset$carname = row.names(mtcars)

gg_point = ggplot( data = dataset,
mapping = aes(x = wt, y = qsec, color = disp,
tooltip = carname, data_id = carname ) +
geom_point_interactive() + theme_minimal()

x <- girafe(ggobj = gg_point, width = 0.7)

if(interactive()){
  print(x)
}
```

girafeOutput

Create a girafe output element

Description

Render a girafe within an application page.

Usage

`girafeOutput(outputId, width = "100\%", height = "500px")`

Arguments

- `outputId`: output variable to read the girafe from. Do not use special JavaScript characters such as a period in the id, this would create a JavaScript error.
- `width`: widget width
- `height`: widget height

Girafe_css

CSS creation helper

Description

It allows specifying individual styles for various SVG elements.

Usage

`girafe_css(css, text = NULL, point = NULL, line = NULL,
area = NULL, image = NULL)`
girafe_options

Arguments

- **css**
  - The generic css style
- **text**
  - Override style for text elements (svg:text)
- **point**
  - Override style for point elements (svg:circle)
- **line**
  - Override style for line elements (svg:line, svg:polyline)
- **area**
  - Override style for area elements (svg:rect, svg:polygon, svg:path)
- **image**
  - Override style for image elements (svg:image)

Value

css as scalar character

Examples

```r
library(ggiraph)

girafe_css(
  css = "fill:orange;stroke:gray;",
  text = "stroke:none; font-size: larger",
  line = "fill:none",
  area = "stroke-width:3px",
  point = "stroke-width:3px",
  image = "outline:2px red"
)
```

girafe_options

Set girafe options

Description

Defines the animation options related to a girafe object.

Usage

```r
girafe_options(x, ...)
```

Arguments

- **x**
  - girafe object.
- **...**
  - set of options defined by calls to opts_* functions or to sizingPolicy from htmlwidgets (this won’t have any effect within a shiny context).

See Also

- [girafe](#)
- Other girafe animation options: [opts_hover](#), [opts_selection](#), [opts_sizing](#), [opts_toolbar](#), [opts_tooltip](#), [opts_zoom](#)
Examples

```r
library(ggplot2)
library(htmlwidgets)

dataset <- mtcars
dataset$carname = row.names(mtcars)

gg_point = ggplot( data = dataset,
mapping = aes(x = wt, y = qsec, color = disp, tooltip = carname, data_id = carname) ) + geom_point_interactive() + theme_minimal()

x <- girafe(ggobj = gg_point)
x <- girafe_options(x = x,
    opts_tooltip(opacity = .7),
    opts_zoom(min = .5, max = 4),
    sizingPolicy(defaultWidth = "100\%", defaultHeight = "300px"),
    opts_hover(css = "fill:red;stroke:orange;r:5pt;")
)

if(interactive()){
    print(x)
}
```

guide_colourbar_interactive

Create interactive continuous colour bar guide

Description

The guide is based on `guide_colourbar`. See the documentation for that function for more details.

Usage

```r
guide_colourbar_interactive(...)
guide_colorbar_interactive(...)
```

Arguments

```
... arguments passed to base function.
```

Value

An interactive guide object.
Details for `scale_*_interactive` and `guide_*_interactive` functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a `guide_legend_interactive` if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as a function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.
- When guide of type ‘colourbar’ is used, it will be converted to a `guide_colourbar_interactive` if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the `element_*_interactive` section for more details.

See Also

`interactive_parameters`
`girafe`

Examples

```r
# add interactive colourbar guide to a ggplot -------
library(ggplot2)
library(ggiraph)

df <- expand.grid(x = 0:5, y = 0:5)
df$z <- runif(nrow(df))

p <- ggplot(df, aes(x, y, fill = z, tooltip = "tooltip")) +
  geom_raster_interactive()

# add an interactive scale (guide is colourbar)
p1 <- p + scale_fill_gradient_interactive(data_id = "colourbar",
                                           onclick = "alert("colourbar")",
                                           tooltip = "colourbar")

x <- girafe(ggobj = p1)
if (interactive()) print(x)
```

# make the legend title interactive
p2 <- p + scale_fill_gradient_interactive(
  data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar",
  name = label_interactive("z",
  data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar"
)
)
x <- girafe(ggobj = p2)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# make the legend labels interactive
p3 <- p + scale_fill_gradient_interactive(
  data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar",
  name = label_interactive("z",
  data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar"
),
  labels = function(breaks) {
    lapply(breaks, function(br) {
      label_interactive(
        as.character(br),
        data_id = "colourbar",
        onclick = "alert("colourbar")",
        tooltip = "colourbar"
      )
    })
  }
)
x <- girafe(ggobj = p3)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# also via the guide
p4 <- p + scale_fill_gradient_interactive(
  data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar",
  guide = guide_colourbar_interactive(
    title.theme = element_text_interactive(
      size = 8,
      data_id = "colourbar",
      onclick = "alert("colourbar")",
    ),
  ))
guide_legend_interactive

Create interactive legend guide

Description

The guide is based on `guide_legend`. See the documentation for that function for more details.

Usage

`guide_legend_interactive(...)`

Arguments

`...`: arguments passed to base function.

Value

An interactive guide object.
Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a `guide_legend_interactive` if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.
- When guide of type ‘colourbar’ is used, it will be converted to a `guide_colourbar_interactive` if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.

See Also

`interactive_parameters`
`girafe`

Examples

```r
# add interactive discrete legend guide to a ggplot -------
# add interactive continuous legend guide to a ggplot -------
library(ggplot2)
library(ggiraph)

dat <- data.frame(
  name = c( "Guy", "Ginette", "David", "Cedric", "Frederic" ),
  gender = c( "Male", "Female", "Male", "Male", "Male" ),
  height = c(169, 160, 171, 172, 171 ) )
p <- ggplot(dat, aes( x = name, y = height, fill = gender,
  data_id = name ) ) +
  geom_bar_interactive(stat = "identity")

# add interactive scale (guide is legend)
p1 <- p +
  scale_fill_manual_interactive(
    values = c(Male = "#0072B2", Female = "#009E73"),
    data_id = c(Female = "Female", Male = "Male"),
```
guide_legend_interactive

```
tooltip = c(Male = "Male", Female = "Female")
)
x <- girafe(ggobj = p1)
if (interactive()) print(x)

# make the title interactive too
p2 <- p +
scale_fill_manual_interactive(
  name = label_interactive("gender", tooltip="Gender levels", data_id="legend.title"),
  values = c(Male = "#0072B2", Female = "#009E73"),
  data_id = c(Female = "Female", Male = "Male"),
  tooltip = c(Male = "Male", Female = "Female")
)
x <- girafe(ggobj = p2)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# the interactive params can be functions too
p3 <- p +
scale_fill_manual_interactive(
  name = label_interactive("gender", tooltip="Gender levels", data_id="legend.title"),
  values = c(Male = "#0072B2", Female = "#009E73"),
  data_id = function(breaks) { as.character(breaks)},
  tooltip = function(breaks) { as.character(breaks)},
  onclick = function(breaks) { paste0("alert("", as.character(breaks), ")") }
)
x <- girafe(ggobj = p3)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# also via the guide
p4 <- p + scale_fill_manual_interactive(
  values = c(Male = "#0072B2", Female = "#009E73"),
  data_id = function(breaks) { as.character(breaks)},
  tooltip = function(breaks) { as.character(breaks)},
  onclick = function(breaks) { paste0("alert("", as.character(breaks), ")") },
  guide = guide_legend_interactive(
    title.theme = element_text_interactive(
      size = 8,
      data_id = "legend.title",
      onclick = "alert(""Gender levels"")",
      tooltip = "Gender levels"
    ),
    label.theme = element_text_interactive(
      size = 8
    )
  )
)
x <- girafe(ggobj = p4)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
```
if (interactive()) print(x)

# make the legend labels interactive
p5 <- p +
  scale_fill_manual_interactive(
    name = label_interactive("gender", tooltip="Gender levels", data_id="legend.title"),
    values = c(Male = "#0072B2", Female = "#009E73"),
    data_id = function(breaks) { as.character(breaks)},
    tooltip = function(breaks) { as.character(breaks)},
    onclick = function(breaks) { paste0("alert("", as.character(breaks), ")") },
    labels = function(breaks) {
      lapply(breaks, function(br) {
        label_interactive(
          as.character(br),
          data_id = as.character(br),
          onclick = paste0("alert("", as.character(br), ")") ,
          tooltip = as.character(br)
        )
      })
    })
  )

x <- girafe(ggobj = p5)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)
library(ggplot2)
library(ggiraph)

set.seed(4393)
dsmall <- diamonds[sample(nrow(diamonds), 1000),]
p <- ggplot(dsmall, aes(x, y)) +
  stat_density_2d(aes(
    fill = stat(nlevel),
    tooltip = paste("nlevel:", stat(nlevel))
  ),
  geom = "interactive_polygon") +
  facet_grid(. ~ cut)

# add interactive scale, by default the guide is a colourbar
p1 <- p + scale_fill_viridis_c_interactive(data_id = "nlevel",
                                          tooltip = "nlevel")
x <- girafe(ggobj = p1)
if (interactive()) print(x)

# make it legend
p2 <- p + scale_fill_viridis_c_interactive(data_id = "nlevel",
                                          tooltip = "nlevel",
                                          guide = "legend")
x <- girafe(ggobj = p2)
if (interactive()) print(x)

# set the keys separately
p3 <- p + scale_fill_viridis_c_interactive(
interactive_parameters

Description

Throughout ggiraph there are functions that add interactivity to ggplot plot elements. The user can control the various aspects of interactivity by supplying a special set of parameters to these functions.
Arguments

- **tooltip**: Tooltip text to associate with one or more elements. If this is supplied a tooltip is shown when the element is hovered. Plain text or html is supported.
- **onclick**: Javascript code to associate with one or more elements. This code will be executed when the element is clicked.
- **hover_css**: Individual css style associate with one or more elements. This css style is applied when the element is hovered and overrides the default style, set via `opts_hover` or `opts_hover_key`.
- **selected_css**: Individual css style associate with one or more elements. This css style is applied when the element is selected and overrides the default style, set via `opts_selection` or `opts_selection_key`.
- **data_id**: Identifier to associate with one or more elements. This is mandatory parameter if hover and selection interactivity is desired. Identifiers are available as reactive input values in Shiny applications.

Details for geom_*_interactive functions

The interactive parameters can be supplied with two ways:

- As aesthetics with the mapping argument (via `aes`). In this way they can be mapped to data columns and apply to a set of geometries.
- As plain arguments into the geom_*_interactive function (see `layer`). In this way they can be set to a scalar value.

Details for annotate_*_interactive functions

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a `guide_legend_interactive` if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.
- When guide of type ‘colourbar’ is used, it will be converted to a `guide_colourbar_interactive` if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.
To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as element_text_interactive (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.

**Details for element_*_interactive functions**

The interactive parameters can be supplied as arguments in the relevant function and they should be scalar values.

For theme text elements (element_text_interactive), the interactive parameters can also be supplied while setting a label value, via the labs family of functions or when setting a scale/guide title or key label. Instead of setting a character value for the element, function label_interactive can be used to define interactive parameters to go along with the label. When the parameters are supplied that way, they override the default values that are set at the theme via element_text_interactive or via the guide’s theme parameters.

**Details for interactive_*_grob functions**

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

**See Also**

- girafe_options
- girafe

---

**interactive_path_grob  Create interactive path grob**

**Description**

The grob is based on pathGrob. See the documentation for that function for more details.

**Usage**

interactive_path_grob(...)  

**Arguments**

... arguments passed to base function, plus any of the interactive_parameters.

**Value**

An interactive grob object.
interactive_points_grob

Description

The grob is based on pointsGrob. See the documentation for that function for more details.

Usage

interactive_points_grob(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Value

An interactive grob object.

Details for interactive_*_grob functions

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also

girafe
interactive_polygon_grob

Create interactive polygon grob

Description
The grob is based on polygonGrob. See the documentation for that function for more details.

Usage
interactive_polygon_grob(...)

Arguments
... arguments passed to base function, plus any of the interactive_parameters.

Value
An interactive grob object.

Details for interactive_*_grob functions
The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also
girafe

interactive_polyline_grob

Create interactive polyline grob

Description
These grobs are based on polylineGrob and linesGrob. See the documentation for those functions for more details.

Usage
interactive_polyline_grob(...)
interactive_lines_grob(...)

Arguments
... arguments passed to base function, plus any of the interactive_parameters.
Value

An interactive grob object.

Details for interactive_*_grob functions

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also

girafe
**interactive_rect_grob**  
Create interactive rectangle grob

---

**Description**

The grob is based on `rectGrob`. See the documentation for that function for more details.

**Usage**

```r
interactive_rect_grob(...)```

**Arguments**

```r
... arguments passed to base function, plus any of the interactive_parameters.
```

**Value**

An interactive grob object.

**Details for interactive_*_grob functions**

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

**See Also**

`girafe`

---

**interactive_roundrect_grob**  
Create interactive rectangle grob

---

**Description**

The grob is based on `roundrectGrob`. See the documentation for that function for more details.

**Usage**

```r
interactive_roundrect_grob(...)```

**Arguments**

```r
... arguments passed to base function, plus any of the interactive_parameters.
```

**Value**

An interactive grob object.
Details for interactive_*_grob functions

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also

- girafe

---

**interactive_segments_grob**

*Create interactive segments grob*

**Description**

The grob is based on `segmentsGrob`. See the documentation for that function for more details.

**Usage**

```r
interactive_segments_grob(...)```

**Arguments**

- `...` arguments passed to base function, plus any of the `interactive_parameters`.

**Value**

An interactive grob object.

Details for interactive_*_grob functions

The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also

- girafe
interactive_text_grob  
Create interactive text grob

Description
The grob is based on textGrob. See the documentation for that function for more details.

Usage
interactive_text_grob(...)

Arguments
... arguments passed to base function, plus any of the interactive_parameters.

Value
An interactive grob object.

Details for interactive_*_grob functions
The interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors depending on params on base function.

See Also
girafe

label_interactive  
Create an interactive label

Description
This function returns an object that can be used as a label via the labs family of functions or when setting a scale/guide name/title or key label. It passes the interactive parameters to a theme element created via element_text_interactive or via an interactive guide.

Usage
label_interactive(label, ...)

Arguments
label  The text for the label (scalar character)
...   any of the interactive_parameters.
Value

an interactive label object

---

**opts_hover**  
*Hover effect settings*

Description

Allows customization of the rendering of graphic elements when the user hovers over them with the cursor (mouse pointer). Use `opts_hover` for interactive geometries in panels, `opts_hover_key` for interactive scales-guides and `opts_hover_theme` for interactive theme elements.

Usage

```r
opts_hover(css = NULL)
```  
```r
opts_hover_key(css = NULL)
```  
```r
opts_hover_theme(css = NULL)
```  
Arguments

- **css** CSS to associate with elements when they are hovered. It must be a scalar character. It can also be constructed with `girafe_css`, to give more control over the CSS for different element types.

See Also

Other girafe animation options: `girafe_options`, `opts_selection`, `opts_sizing`, `opts_toolbar`, `opts_tooltip`, `opts_zoom`

Examples

```r
library(ggplot2)

dataset <- mtcars
dataset$carname = row.names(mtcars)

gg <- ggplot(
data = dataset,
mapping = aes(x = wt, y = qsec, color = disp,
              tooltip = carname, data_id = carname) ) +
            geom_point_interactive() + theme_minimal()

x <- girafe(ggobj = gg)
x <- girafe_options(x,
                     opts_hover(css = "fill:wheat;stroke:orange;r:5pt;") )
if(interactive()) print(x)
```
opts_selection  

Selection effect settings

Description

Allows customization of the rendering of selected graphic elements. Use opts_selection for interactive geometries in panels, opts_selection_key for interactive scales/guides and opts_selection_theme for interactive theme elements.

Usage

opts_selection(css = NULL, type = "multiple", only_shiny = TRUE, selected = character(0))

opts_selection_key(css = NULL, type = "single", only_shiny = TRUE, selected = character(0))

opts_selection_theme(css = NULL, type = "single", only_shiny = TRUE, selected = character(0))

Arguments

- **css**: css to associate with elements when they are selected. It must be a scalar character. It can also be constructed with girafe_css, to give more control over the css for different element types.
- **type**: selection mode ("single", "multiple", "none") when widget is in a Shiny application.
- **only_shiny**: disable selections if not in a shiny context.
- **selected**: character vector, id to be selected when the graph will be initialized.

See Also

Other girafe animation options: girafe_options, opts_hover, opts_sizing, opts_toolbar, opts_tooltip, opts_zoom

Examples

```r
library(ggplot2)

dataset <- mtcars
dataset$carname = row.names(mtcars)

gg <- ggplot(
  data = dataset,
  mapping = aes(x = wt, y = qsec, color = disp,
                tooltip = carname, data_id = carname) ) +
geom_point_interactive() + theme_minimal()
```
x <- girafe(ggobj = gg)
x <- girafe_options(x,
    opts_selection(type = "multiple",
    css = "fill:red;stroke:gray;r:5pt;")
if( interactive() ) print(x)

---

**opts_sizing**  
*Girafe sizing settings*

**Description**

Allows customization of the svg style sizing

**Usage**

```r
opts_sizing(rescale = TRUE, width = 1)
```

**Arguments**

- `rescale`  
  if FALSE, graphic will not be resized and the dimensions are exactly those of the container.
- `width`  
  widget width ratio (0 < width <= 1).

**See Also**

Other girafe animation options: `girafe_options, opts_hover, opts_selection, opts_toolbar, opts_tooltip, opts_zoom`

**Examples**

```r
library(ggplot2)

dataset <- mtcars
dataset$carname = row.names(mtcars)

gg <- ggplot(
  data = dataset,
  mapping = aes(x = wt, y = qsec, color = disp, 
  tooltip = carname, data_id = carname) ) +
  geom_point_interactive() + theme_minimal()

x <- girafe(ggobj = gg)
x <- girafe_options(x,
    opts_sizing(rescale = FALSE) )
if( interactive() ) print(x)
```
**Description**

Allows customization of the toolbar

**Usage**

```r
opts_toolbar(position = "topright", saveaspng = TRUE)
```

**Arguments**

- `position` one of 'top', 'bottom', 'topleft', 'topright', 'bottomleft', 'bottomright'
- `saveaspng` set to TRUE to propose the 'save as png' button.

**Note**

`saveaspng` relies on JavaScript promises, so any browsers that don’t natively support the standard Promise object will need to have a polyfill (e.g. Internet Explorer with version less than 11 will need it).

**See Also**

Other girafe animation options: `girafe_options`, `opts_hover`, `opts_selection`, `opts_sizing`, `opts_tooltip`, `opts_zoom`

**Examples**

```r
library(ggplot2)

dataset <- mtcars
dataset$carnname = row.names(mtcars)

gg <- ggplot(
data = dataset,
mapping = aes(x = wt, y = qsec, color = disp,
              tooltip = carname, data_id = carname) ) +
  geom_point_interactive() + theme_minimal()

x <- girafe(ggobj = gg)
x <- girafe_options(x,
  opts_toolbar(position = "top") )
if( interactive() ) print(x)
```
Description

Settings to be used with `girafe` for tooltip customisation.

Usage

```r
opts_tooltip(css = NULL, offx = 10, offy = 0,
             use_cursor_pos = TRUE, opacity = 0.9, use_fill = FALSE,
             use_stroke = FALSE, delay_Mouseover = 200, delay_Mouseout = 500,
             zindex = 999)
```

Arguments

- **css** extra css (added to `position: absolute;pointer-events: none;`) used to customize tooltip area.
- **offx, offy** tooltip x and y offset
- **use_cursor_pos** should the cursor position be used to position tooltip (in addition to offx and offy). Setting to TRUE will have no effect in the RStudio browser windows.
- **opacity** tooltip background opacity
- **use_fill, use_stroke** logical, use fill and stroke properties to color tooltip.
- **delay_Mouseover** The duration in milliseconds of the transition associated with tooltip display.
- **delay_Mouseout** The duration in milliseconds of the transition associated with tooltip end of display.
- **zindex** tooltip css z-index, default to 999.

See Also

Other girafe animation options: `girafe_options`, `opts_hover`, `opts_selection`, `opts_sizing`, `opts_toolbar`, `opts_zoom`

Examples

```r
library(ggplot2)

dataset <- mtcars
dataset$carname = row.names(mtcars)

gg <- ggplot(
  data = dataset,
  mapping = aes(x = wt, y = qsec, color = disp,
                tooltip = carname, data_id = carname) ) +
```
opts_zoom

Legend

geom_point_interactive() + theme_minimal()

x <- girafe(ggobj = gg)
x <- girafe_options(x,
   opts_tooltip(opacity = .7,
       offx = 20, offy = -10,
       use_fill = TRUE, use_stroke = TRUE,
       delaymouseout = 1000) )
if( interactive() ) print(x)

opts_zoom

Zoom settings

Description

Allows customization of the zoom.

Usage

opts_zoom(min = 1, max = 1)

Arguments

min minimum zoom factor
max maximum zoom factor

See Also

Other girafe animation options: girafe_options, opts_hover, opts_selection, opts_sizing, opts_toolbar, opts_tooltip

Examples

library(ggplot2)

dataset <- mtcars
dataset$carname = row.names(mtcars)

gg <- ggplot(
   data = dataset,
   mapping = aes(x = wt, y = qsec, color = disp,
                   tooltip = carname, data_id = carname) ) +
   geom_point_interactive() + theme_minimal()

x <- girafe(ggobj = gg)
x <- girafe_options(x,
   opts_zoom(min = .7, max = 2) )
if( interactive() ) print(x)
**renderGirafe**

*Reactive version of girafe object*

**Description**

Makes a reactive version of a girafe object for use in Shiny.

**Usage**

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

**Arguments**

- `expr`: An expression that returns a `girafe` object.
- `env`: The environment in which to evaluate `expr`.
- `quoted`: Is `expr` a quoted expression

**Examples**

```r
## Not run:
if( require(shiny) && interactive() ){
  app_dir <- file.path( system.file(package = "ggiraph"), "examples/shiny" )
  shinyAppDir(appDir = app_dir )
}
## End(Not run)
```

---

**renderGgiraph**

*Reactive version of ggiraph object*

**Description**

Makes a reactive version of a ggiraph object for use in Shiny.

**Usage**

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

**Arguments**

- `expr`: An expression that returns a `ggiraph` object.
- `env`: The environment in which to evaluate `expr`.
- `quoted`: Is `expr` a quoted expression

**Examples**

```r
## Not run:
if( require(shiny) && interactive() ){
  app_dir <- file.path( system.file(package = "ggiraph"), "examples/shiny" )
  shinyAppDir(appDir = app_dir )
}
## End(Not run)
```
run_girafe_example

---

**run_girafe_example**  *Run shiny examples and see corresponding code*

---

**Description**

Run shiny examples and see corresponding code

**Usage**

```r
run_girafe_example(name = "crimes")
```

**Arguments**

- `name`  
  an application name, one of cars, click_scale, crimes, DT, dynamic_ui, iris, maps and modal.

---

**scale_alpha_interactive**

*Create interactive scales for alpha transparency*

---

**Description**

These scales are based on `scale_alpha`, `scale_alpha_continuous`, `scale_alpha_discrete`, `scale_alpha_ordinal`, `scale_alpha_date` and `scale_alpha_datetime`. See the documentation for those functions for more details.

**Usage**

```r
scale_alpha_interactive(...)  
scale_alpha_continuous_interactive(...)  
scale_alpha_discrete_interactive(...)  
scale_alpha_ordinal_interactive(...)  
scale_alpha_date_interactive(...)  
scale_alpha_datetime_interactive(...)  
```

**Arguments**

- `...`  
  arguments passed to base function, plus any of the `interactive_parameters`.  

**Value**

An interactive scale object.

**Details for scale_*_interactive and guide_*_interactive functions**

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a `guide_legend_interactive` if it’s not already.  
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.  
  The interactive parameters here, give interactivity only to the key elements of the guide.

- When guide of type ‘colourbar’ is used, it will be converted to a `guide_colourbar_interactive` if it’s not already.  
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The `guide` arguments ‘title.theme’ and ‘label.theme’ can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the `element_*_interactive` section for more details.

**See Also**

`girafe`

Other interactive scale: `scale_colour_brewer_interactive, scale_colour_interactive, scale_gradient_interactive, scale_linetype_interactive, scale_manual_interactive, scale_shape_interactive, scale_size_interactive, scale_viridis_interactive`

---

`scale_colour_brewer_interactive`  
*Create interactive colorbrewer scales*

**Description**

These scales are based on `scale_colour_brewer, scale_fill_brewer, scale_colour_distiller` and `scale_fill_distiller`. See the documentation for those functions for more details.
Usage

scale_colour_brewer_interactive(...)
scale_color_brewer_interactive(...)
scale_fill_brewer_interactive(...)
scale_colour_distiller_interactive(...)
scale_color_distiller_interactive(...)
scale_fill_distiller_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Value

An interactive scale object.

Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a guide_legend_interactive if it's not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.

- When guide of type ‘colourbar’ is used, it will be converted to a guide_colourbar_interactive if it's not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as element_text_interactive (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.
See Also

- `girafe`

Other interactive scale: `scale_alpha_interactive`, `scale_colour_interactive`, `scale_gradient_interactive`, `scale_linetype_interactive`, `scale_manual_interactive`, `scale_shape_interactive`, `scale_size_interactive`, `scale_viridis_interactive`

---

**scale_colour_interactive**

*Create interactive colour scales*

---

**Description**

These scales are based on `scale_colour_continuous`, `scale_fill_continuous`, `scale_colour_grey`, `scale_fill_grey`, `scale_colour_hue` and `scale_fill_hue`. See the documentation for those functions for more details.

**Usage**

- `scale_colour_continuous_interactive(...)`
- `scale_color_continuous_interactive(...)`
- `scale_fill_continuous_interactive(...)`
- `scale_colour_grey_interactive(...)`
- `scale_color_grey_interactive(...)`
- `scale_fill_grey_interactive(...)`
- `scale_colour_hue_interactive(...)`
- `scale_color_hue_interactive(...)`
- `scale_fill_hue_interactive(...)`
- `scale_colour_hue_interactive(...)`
- `scale_color_hue_interactive(...)`
- `scale_fill_hue_interactive(...)`

**Arguments**

- `...` arguments passed to base function, plus any of the `interactive_parameters`.

**Value**

An interactive scale object.
Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a `guide_legend_interactive` if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.

- When guide of type ‘colourbar’ is used, it will be converted to a `guide_colourbar_interactive` if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.

See Also

- `girafe`

Other interactive scale: `scale_alpha_interactive, scale_colour_brewer_interactive, scale_gradient_interactive, scale_linetype_interactive, scale_manual_interactive, scale_shape_interactive, scale_size_interactive, scale_viridis_interactive`

---

**scale_gradient_interactive**

Create interactive gradient colour scales

**Description**

These scales are based on `scale_colour_gradient, scale_fill_gradient, scale_colour_gradient2, scale_fill_gradient2, scale_colour_gradientn` and `scale_fill_gradientn`. See the documentation for those functions for more details.
Usage

scale_colour_gradient_interactive(...)
scale_color_gradient_interactive(...)
scale_fill_gradient_interactive(...)
scale_colour_gradient2_interactive(...)
scale_color_gradient2_interactive(...)
scale_fill_gradient2_interactive(...)
scale_colour_gradientn_interactive(...)
scale_color_gradientn_interactive(...)
scale_fill_gradientn_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive parameters.

Value

An interactive scale object.

Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a guide_legend_interactive if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.

- When guide of type ‘colourbar’ is used, it will be converted to a guide_colourbar_interactive if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’
and ‘label.theme’ can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the `element_*_interactive` section for more details.

**See Also**

girafe

Other interactive scale: `scale_alpha_interactive`, `scale_colour_brewer_interactive`, `scale_colour_interactive`, `scale_linetype_interactive`, `scale_manual_interactive`, `scale_shape_interactive`, `scale_size_interactive`, `scale_viridis_interactive`

**Examples**

```r
# add interactive gradient colour scale to a ggplot -------
library(ggplot2)
library(ggiraph)

df <- expand.grid(x = 0:5, y = 0:5)
df$z <- runif(nrow(df))

p <- ggplot(df, aes(x, y, fill = z, tooltip = "tooltip")) +
  geom_raster_interactive()

# add an interactive scale (guide is colourbar)

p1 <- p + scale_fill_gradient_interactive(data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar")
x <- girafe(ggobj = p1)
if (interactive()) print(x)

# make the legend title interactive

p2 <- p + scale_fill_gradient_interactive(data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar",
  name = label_interactive(
    "z",
    data_id = "colourbar",
    onclick = "alert("colourbar")",
    tooltip = "colourbar"
  )
)
x <- girafe(ggobj = p2)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# make the legend labels interactive

p3 <- p + scale_fill_gradient_interactive(data_id = "colourbar",
  onclick = "alert("colourbar")",
  tooltip = "colourbar",

name = label_interactive("z",
  data_id = "colourbar",
  onclick = "alert('colourbar')",
  tooltip = "colourbar"
),
labels = function(breaks) {
lapply(breaks, function(br) {
  label_interactive(
    as.character(br),
    data_id = "colourbar",
    onclick = "alert('colourbar')",
    tooltip = "colourbar"
  )
})
}

x <- girafe(ggobj = p3)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# also via the guide
p4 <- p + scale_fill_gradient_interactive(
  data_id = "colourbar",
  onclick = "alert('colourbar')",
  tooltip = "colourbar",
  guide = guide_colourbar_interactive(
    title.theme = element_text_interactive(
      size = 8,
      data_id = "colourbar",
      onclick = "alert('colourbar')",
      tooltip = "colourbar"
    ),
    label.theme = element_text_interactive(
      size = 8,
      data_id = "colourbar",
      onclick = "alert('colourbar')",
      tooltip = "colourbar"
    )
  )
)

x <- girafe(ggobj = p4)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# make the legend background interactive
p5 <- p4 + theme(
  legend.background = element_rect_interactive(
    data_id = "colourbar",
    onclick = "alert('colourbar')",
    tooltip = "colourbar"
scale_linetype_interactive

Create interactive scales for line patterns

Description

These scales are based on scale_linetype, scale_linetype_continuous and scale_linetype_discrete. See the documentation for those functions for more details.

Usage

scale_linetype_interactive(...)

scale_linetype_continuous_interactive(...)

scale_linetype_discrete_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Value

An interactive scale object.

Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

• When guide of type ‘legend’ is used, it will be converted to a guide_legend_interactive if it’s not already.

The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.

The interactive parameters here, give interactivity only to the key elements of the guide.
When guide of type ‘colourbar’ is used, it will be converted to a `guide_colourbar_interactive` if it’s not already. The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.

**See Also**

`girafe`

Other interactive scale: `scale_alpha_interactive`, `scale_colour_brewer_interactive`, `scale_colour_interactive`, `scale_gradient_interactive`, `scale_manual_interactive`, `scale_shape_interactive`, `scale_size_interactive`, `scale_viridis_interactive`

---

**scale_manual_interactive**

*Create your own interactive discrete scale*

---

**Description**

These scales are based on `scale_colour_manual`, `scale_fill_manual`, `scale_size_manual`, `scale_shape_manual`, `scale_linetype_manual`, `scale_alpha_manual` and `scale_discrete_manual`. See the documentation for those functions for more details.

**Usage**

```r
scale_colour_manual_interactive(...)  
scale_color_manual_interactive(...)  
scale_fill_manual_interactive(...)  
scale_size_manual_interactive(...)  
scale_shape_manual_interactive(...)  
scale_linetype_manual_interactive(...)  
scale_alpha_manual_interactive(...)  
scale_discrete_manual_interactive(...)```

scale_manual_interactive

Arguments

... arguments passed to base function, plus any of the interactive parameters.

Value

An interactive scale object.

Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a guide_legend_interactive if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.

  The interactive parameters here, give interactivity only to the key elements of the guide.

- When guide of type ‘colourbar’ is used, it will be converted to a guide_colourbar_interactive if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as element_text_interactive (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.

See Also

girafe

Other interactive scale: scale_alpha_interactive, scale_colour_brewer_interactive, scale_colour_interactive, scale_gradient_interactive, scale_linetype_interactive, scale_shape_interactive, scale_size_interactive, scale_viridis_interactive

Examples

# add interactive manual fill scale to a ggplot -------
library(ggplot2)
library(ggiraph)

dat <- data.frame(
  name = c( "Guy", "Ginette", "David", "Cedric", "Frederic" ),
  gender = c( "Male", "Female", "Male", "Male", "Male" ),
  height = c(169, 160, 171, 172, 171 )
)
p <- ggplot(dat, aes(x = name, y = height, fill = gender,
        data_id = name)) +
        geom_bar_interactive(stat = "identity")

# add interactive scale (guide is legend)
p1 <- p +
        scale_fill_manual_interactive(
            values = c(Male = "#0072B2", Female = "#009E73"),
            data_id = c(Female = "Female", Male = "Male"),
            tooltip = c(Male = "Male", Female = "Female")
        )
x <- girafe(ggobj = p1)
if (interactive()) print(x)

# make the title interactive too
p2 <- p +
        scale_fill_manual_interactive(
            name = label_interactive("gender", tooltip="Gender levels", data_id="legend.title"),
            values = c(Male = "#0072B2", Female = "#009E73"),
            data_id = function(breaks) { as.character(breaks)},
            tooltip = function(breaks) { as.character(breaks)},
            onclick = function(breaks) { paste0("alert("", as.character(breaks), "")") }
        )
x <- girafe(ggobj = p2)
x <- girafe_options(x,
    opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# the interactive params can be functions too
p3 <- p +
        scale_fill_manual_interactive(
            name = label_interactive("gender", tooltip="Gender levels", data_id="legend.title"),
            values = c(Male = "#0072B2", Female = "#009E73"),
            data_id = function(breaks) { as.character(breaks)},
            tooltip = function(breaks) { as.character(breaks)},
            onclick = function(breaks) { paste0("alert("", as.character(breaks), "")") }
        )
x <- girafe(ggobj = p3)
x <- girafe_options(x,
    opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# also via the guide
p4 <- p + scale_fill_manual_interactive(
    values = c(Male = "#0072B2", Female = "#009E73"),
    data_id = function(breaks) { as.character(breaks)},
    tooltip = function(breaks) { as.character(breaks)},
    onclick = function(breaks) { paste0("alert("", as.character(breaks), "")") },
    guide = guide_legend_interactive(
        title.theme = element_text_interactive(
            size = 8,
            data_id = "legend.title",
            onclick = "alert("Gender levels")",
            tooltip = "Gender levels"
scale_shape_interactive

Create interactive scales for shapes

Description

These scales are based on `scale_shape, scale_shape_continuous, scale_shape_discrete` and `scale_shapeOrdinal`. See the documentation for those functions for more details.

Usage

scale_shape_interactive(...)

scale_shape_continuous_interactive(...)

scale_shape_discrete_interactive(...)

scale_shapeOrdinal_interactive(...)

x <- girafe(ggobj = p4)
x <- girafe_options(x,
                   opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

# make the legend labels interactive
p5 <- p +
  scale_fill_manual_interactive(
    name = label_interactive("gender", tooltip="Gender levels", data_id="legend.title"),
    values = c(Male = "#0072B2", Female = "#009E73"),
    data_id = function(breaks) { as.character(breaks)},
    tooltip = function(breaks) { as.character(breaks)},
    onclick = function(breaks) { paste0("alert("", as.character(breaks), ")") },
    labels = function(breaks) {
      lapply(breaks, function(br) {
        label_interactive(
          as.character(br),
          data_id = as.character(br),
          onclick = paste0("alert("", as.character(br), ")")
        )
      })
    })
  )
x <- girafe(ggobj = p5)
x <- girafe_options(x,
                   opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)
scale_shape_discrete_interactive(…)

scale_shape_ordinal_interactive(…)

Arguments

... arguments passed to base function, plus any of the interactive_parameters.

Value

An interactive scale object.

Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a guide_legend_interactive if it's not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.

- When guide of type ‘colourbar’ is used, it will be converted to a guide_colourbar_interactive if it's not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as element_text_interactive (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.

See Also

girafe

Other interactive scale: scale_alpha_interactive, scale_colour_brewer_interactive, scale_colour_interactive, scale_gradient_interactive, scale_linetype_interactive, scale_manual_interactive, scale_size_interactive, scale_viridis_interactive
**scale_size_interactive**

*Create interactive scales for area or radius*

**Description**

These scales are based on `scale_size, scale_size_continuous, scale_size_discrete, scale_size_ordinal` and `scale_radius`. See the documentation for those functions for more details.

**Usage**

```r
scale_size_interactive(...)
```

```r
scale_size_continuous_interactive(...)
```

```r
scale_size_discrete_interactive(...)
```

```r
scale_size_ordinal_interactive(...)
```

```r
scale_radius_interactive(...)
```

**Arguments**

...  

Arguments passed to base function, plus any of the `interactive_parameters`.

**Value**

An interactive scale object.

**Details for scale_*_interactive and guide_*_interactive functions**

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type ‘legend’ is used, it will be converted to a `guide_legend_interactive` if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.
- When guide of type ‘colourbar’ is used, it will be converted to a `guide_colourbar_interactive` if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.
To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments `title.theme` and `label.theme` can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the `element_*_interactive` section for more details.

**See Also**

girafe

Other interactive scale: `scale_alpha_interactive, scale_colour_brewer_interactive, scale_colour_interactive`, `scale_gradient_interactive, scale_linetype_interactive, scale_manual_interactive, scale_shape_interactive, scale_viridis_interactive`

---

**scale_viridis_interactive**

*Create interactive viridis colour scales*

**Description**

These scales are based on `scale_colour_viridis_d, scale_fill_viridis_d, scale_colour_viridis_c` and `scale_fill_viridis_c`. See the documentation for those functions for more details.

**Usage**

```r
scale_colour_viridis_d_interactive(...)  
scale_color_viridis_d_interactive(...)  
scale_fill_viridis_d_interactive(...)  
scale_colour_viridis_c_interactive(...)  
scale_color_viridis_c_interactive(...)  
scale_fill_viridis_c_interactive(...)  
```

**Arguments**

`...` arguments passed to base function, plus any of the `interactive_parameters`.

**Value**

An interactive scale object.
Details for scale_*_interactive and guide_*_interactive functions

For scales, the interactive parameters can be supplied as arguments in the relevant function and they can be scalar values or vectors, depending on the number of breaks (levels) and the type of the guide used. The guides do not accept any interactive parameter directly, they receive them from the scales.

- When guide of type `legend` is used, it will be converted to a `guide_legend_interactive` if it’s not already.
  The length of each scale interactive parameter vector should match the length of the breaks. It can also be a named vector, where each name should correspond to the same break name. It can also be defined as function that takes the breaks as input and returns a named or unnamed vector of values as output.
  The interactive parameters here, give interactivity only to the key elements of the guide.

- When guide of type `colourbar` is used, it will be converted to a `guide_colourbar_interactive` if it’s not already.
  The scale interactive parameters in this case should be scalar values and give interactivity to the colorbar only.

To provide interactivity to the rest of the elements of a guide, (title, labels, background, etc), the relevant theme elements or relevant guide arguments can be used. The guide arguments ‘title.theme’ and ‘label.theme’ can be defined as `element_text_interactive` (in fact, they will be converted to that if they are not already), either directly or via the theme. See the element_*_interactive section for more details.

See Also

girafe

Other interactive scale: `scale_alpha_interactive`, `scale_colour_brewer_interactive`, `scale_colour_interactive`, `scale_gradient_interactive`, `scale_linetype_interactive`, `scale_manual_interactive`, `scale_shape_interactive`, `scale_size_interactive`

Examples

```r
# add interactive viridis scale to a ggplot -----
library(ggplot2)
library(ggiraph)

set.seed(4393)
dsmall <- diamonds[sample(nrow(diamonds), 1000),]
p <- ggplot(dsmall, aes(x, y)) +
  stat_density_2d(aes(
    fill = stat(nlevel),
    tooltip = paste("nlevel:", stat(nlevel))
  ),
  geom = "interactive_polygon") +
  facet_grid(. ~ cut)

# add interactive scale, by default the guide is a colourbar
p1 <- p + scale_fill_viridis_c_interactive(data_id = "nlevel",
```
x <- girafe(ggobj = p1)
if (interactive()) print(x)

# make it legend
p2 <- p + scale_fill_viridis_c_interactive(data_id = "nlevel",
                                          tooltip = "nlevel",
                                          guide = "legend")

x <- girafe(ggobj = p2)
if (interactive()) print(x)

# set the keys separately
p3 <- p + scale_fill_viridis_c_interactive(
    data_id = function(breaks) {
        as.character(breaks)
    },
    tooltip = function(breaks) {
        as.character(breaks)
    },
    guide = "legend"
)

x <- girafe(ggobj = p3)
if (interactive()) print(x)

# make the title and labels interactive
p4 <- p + scale_fill_viridis_c_interactive(
    data_id = function(breaks) {
        as.character(breaks)
    },
    tooltip = function(breaks) {
        as.character(breaks)
    },
    guide = "legend",
    name = label_interactive("nlevel", data_id = "nlevel",
                             tooltip = "nlevel"),
    labels = function(breaks) {
        l <- lapply(breaks, function(br) {
            label_interactive(
                as.character(br),
                data_id = as.character(br),
                onclick = paste0("alert("", as.character(br), "")"),
                tooltip = as.character(br)
            )
        })
        l
    }
)

x <- girafe(ggobj = p4)

x <- girafe_options(x,
                    opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)
Index

*Topic device
  dsvg, 6

aes, 10, 12–15, 17–19, 21–24, 26, 28, 29, 31–33, 35–37, 39, 54
annotate, 4
annotate_interactive, 4
annotation_raster, 5
annotation_raster_interactive, 5

Devices, 6
dsvg, 6, 7, 41, 42
dsvg_view, 7

element_interactive, 7
element_line, 7
element_line_interactive
  (element_interactive), 7
element_rect, 7
element_rect_interactive
  (element_interactive), 7
element_text, 7
element_text_interactive, 8, 55, 61
element_text_interactive
  (element_interactive), 7

geom_abline, 9
geom_abline_interactive, 9
geom_area, 35
geom_area_interactive
  (geom_ribbon_interactive), 34
geom_bar, 12
geom_bar_interactive, 12
gem_boxplot, 13
gem_boxplot_interactive, 13
gem_col, 12
gem_col_interactive
  (geom_bar_interactive), 12
geom_contour, 14
geom_contour_interactive, 14

geom_crossbar, 15
geom_crossbar_interactive, 15
geom_density, 18
geom_density2d_interactive
  (geom_density_2d_interactive), 16
geom_density_2d, 16
geom_density2d_interactive, 16
geom_density_interactive, 18
geom_errorbar, 15
geom_errorbar_interactive
  (geom_crossbar_interactive), 15
geom_errorbarh, 19
geom_errorbarh_interactive, 19
geom_freqpoly, 20
geom_freqpoly_interactive, 20
geom_histogram, 20
geom_histogram_interactive
  (geom_freqpoly_interactive), 20
geom_hline, 9
geom_hline_interactive
  (geom_abline_interactive), 9
gem_jitter, 21
gem_jitter_interactive, 21
gem_label, 22
gem_label_interactive, 22
gem_line, 25
gem_line_interactive
  (geom_path_interactive), 25
gem_linerange, 15
gem_linerange_interactive
  (geom_crossbar_interactive), 15
gem_map, 24
gem_map_interactive, 24
gem_path, 25
gem_path_interactive, 25
gem_point, 28
gem_point_interactive, 28
gem_pointrange, 15
rasterGrob, 58
rectGrob, 59
renderggiraph, 68
renderGirafe, 68
roundrectGrob, 59
run_girafe_example, 69

scale_alpha, 69
scale_alpha_continuous, 69
  (scale_alpha_continuous_interactive), 69
scale_alpha_date, 69
  (scale_alpha_date_interactive), 69
scale_alpha_datetime, 69
  (scale_alpha_datetime_interactive), 69

scale_alpha_manual_interactive
  (scale_manual_interactive), 78

scale_alpha_manual, 78

scale_alpha_ordinal, 69
  (scale_alpha_ordinal_interactive), 69
scale_alpha_discrete, 69
  (scale_alpha_discrete_interactive), 69
scale_alpha_alpha, 69
  (scale_alpha_alpha_interactive), 69

scale_alpha_manual_interactive
  (scale_manual_interactive), 78

scale_color_manual_interactive
  (scale_manual_interactive), 78
scale_color_viridis_c_interactive
  (scale_viridis_interactive), 84
scale_color_viridis_d_interactive
  (scale_viridis_interactive), 84
scale_colour_brewer, 70

scale_colour_brewer_interactive, 70, 70,
  73, 75, 78, 79, 82, 84, 85
scale_colour_continuous, 72

scale_colour_continuous_interactive
  (scale_colour_interactive), 72

scale_colour_distiller, 70

scale_colour_distiller_interactive
  (scale_colour_brewer_interactive), 70
scale_colour_distiller, 70

scale_colour_gradient, 73

scale_colour_gradient_interactive
  (scale_gradient_interactive), 73

scale_colour_gradientn, 73

scale_colour_gradientn_interactive
  (scale_gradient_interactive), 73

scale_colour_grey, 72

scale_colour_grey_interactive
  (scale_colour_interactive), 78

scale_colour_hue, 72

scale_colour_hue_interactive
  (scale_colour_interactive), 78

scale_colour_interactive
  (scale_viridis_interactive), 84

scale_colour_manual, 78

scale_colour_manual_interactive
  (scale_manual_interactive), 78

scale_colour_viridis_c, 84

scale_colour_viridis_c_interactive
  (scale_viridis_interactive), 84

scale_colour_viridis_d, 84

scale_colour_viridis_d_interactive
  (scale_viridis_interactive), 84

scale_discrete_manual, 78

scale_discrete_manual_interactive
  (scale_manual_interactive), 78
scale_fill_brewer, 70
scale_fill_brewer_interactive
  (scale_colour_brewer_interactive), 70
scale_fill_continuous, 72
scale_fill_continuous_interactive
  (scale_colour_interactive), 72
scale_fill_distiller, 70
scale_fill_distiller_interactive
  (scale_colour_brewer_interactive), 70
scale_fill_gradient, 73
scale_fill_gradient2, 73
scale_fill_gradient2_interactive
  (scale_gradient_interactive), 73
scale_fill_gradient_interactive
  (scale_gradient_interactive), 73
scale_fill_gradientn, 73
scale_fill_gradientn_interactive
  (scale_gradient_interactive), 73
scale_fill_grey, 72
scale_fill_grey_interactive
  (scale_colour_interactive), 72
scale_fill_hue, 72
scale_fill_hue_interactive
  (scale_colour_interactive), 72
scale_fill_manual, 78
scale_fill_manual_interactive
  (scale_manual_interactive), 78
scale_fill_viridis_c, 84
scale_fill_viridis_c_interactive
  (scale_viridis_interactive), 84
scale_fill_viridis_d, 84
scale_fill_viridis_d_interactive
  (scale_viridis_interactive), 84
scale_gradients_interactive, 70, 72, 73, 75, 77, 79, 82, 84, 85
scale_linetype, 77
scale_linetype_continuous, 77
scale_linetype_continuous_interactive
  (scale_linetype_interactive), 77
scale_linetype_discrete, 77
scale_linetype_discrete_interactive
  (scale_linetype_interactive), 77
scale_linetype_manual, 78
scale_linetype_manual_interactive
  (scale_manual_interactive), 78
scale_linetype_manual_interactive
  (scale_manual_interactive), 78
scale_radius, 83
scale_radius_interactive
  (scale_size_interactive), 83
scale_shape, 81
scale_shape_continuous, 81
scale_shape_continuous_interactive
  (scale_shape_interactive), 81
scale_shape_discrete, 81
scale_shape_discrete_interactive
  (scale_shape_interactive), 81
scale_shape_manual, 81
scale_shape_manual_interactive
  (scale_manual_interactive), 81
scale_shape_manual_interactive
  (scale_manual_interactive), 81
scale_size, 83
scale_size_continuous, 83
scale_size_continuous_interactive
  (scale_size_interactive), 83
scale_size_discrete, 83
scale_size_discrete_interactive
  (scale_size_interactive), 83
scale_size_manual, 83
scale_size_manual_interactive
  (scale_manual_interactive), 83
scale_size_manual_interactive
  (scale_manual_interactive), 83
scale_size_ordinal, 83
scale_size_ordinal_interactive
  (scale_size_interactive), 83
scale_viridis_interactive
  (scale_size_interactive), 83
segmentsGrob, 60
textGrob, 61
theme, 7