Package ‘ggiraph’

August 19, 2022

Type    Package
Title   Make 'ggplot2' Graphics Interactive
Description Create interactive 'ggplot2' graphics using 'htmlwidgets'.
Version  0.8.3
License  GPL-3
Copyright See file COPYRIGHTS.
Encoding UTF-8
SystemRequirements C++11, libpng
Imports  grid, ggplot2 (>= 3.3.5), htmlwidgets (>= 1.5), stats,
        htmltools, Rcpp (>= 1.0), systemfonts, purrr, rlang, uuid
LinkingTo Rcpp, systemfonts
Suggests knitr, tinytest, rmarkdown, maps, hexbin, shiny, sf (>= 1.0),
        ggrepel (>= 0.9.1), quantreg, xml2 (>= 1.0), dplyr
VignetteBuilder knitr
URL     https://davidgohel.github.io/ggiraph/
BugReports https://github.com/davidgohel/ggiraph/issues
RoxygenNote 7.2.1
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_bin_2d_interactive.R' 'geom_boxplot_interactive.R'
       'geom_col_interactive.R' 'geom_contour_interactive.R'
       'geom_count_interactive.R' 'geom_crossbar_interactive.R'
       'geom_curve_interactive.R' 'geom_density_2d_interactive.R'
       'geom_density_interactive.R' 'geom_dotplot_interactive.R'
       'geom_errorbar_interactive.R' 'geom_errorbarh_interactive.R'
       'geom_freqpoly_interactive.R' 'geom_hex_interactive.R'
       'geom_histogram_interactive.R' 'geom_hline_interactive.R'

R topics documented:

'annotate_interactive' ................................................. 4
'annotation_raster_interactive' ................................. 5
dsvg ................................................................. 6
dsvg_view ........................................................... 7
element_interactive .................................................. 8
'font_family_exists' ............................................... 10
'geom_abline_interactive' .......................................... 11
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>geom_bar_interactive</td>
<td>13</td>
</tr>
<tr>
<td>geom_bin_2d_interactive</td>
<td>15</td>
</tr>
<tr>
<td>geom_boxplot_interactive</td>
<td>16</td>
</tr>
<tr>
<td>geom_contour_interactive</td>
<td>17</td>
</tr>
<tr>
<td>geom_count_interactive</td>
<td>18</td>
</tr>
<tr>
<td>geom_crossbar_interactive</td>
<td>19</td>
</tr>
<tr>
<td>geom_curve_interactive</td>
<td>20</td>
</tr>
<tr>
<td>geom_density_2d_interactive</td>
<td>22</td>
</tr>
<tr>
<td>geom_density_interactive</td>
<td>23</td>
</tr>
<tr>
<td>geom_dotplot_interactive</td>
<td>24</td>
</tr>
<tr>
<td>geom_errorbarh_interactive</td>
<td>25</td>
</tr>
<tr>
<td>geom_freqpoly_interactive</td>
<td>27</td>
</tr>
<tr>
<td>geom_hex_interactive</td>
<td>28</td>
</tr>
<tr>
<td>geom_jitter_interactive</td>
<td>29</td>
</tr>
<tr>
<td>geom_label_interactive</td>
<td>30</td>
</tr>
<tr>
<td>geom_map_interactive</td>
<td>31</td>
</tr>
<tr>
<td>geom_path_interactive</td>
<td>33</td>
</tr>
<tr>
<td>geom_point_interactive</td>
<td>35</td>
</tr>
<tr>
<td>geom_polygon_interactive</td>
<td>36</td>
</tr>
<tr>
<td>geom_quantile_interactive</td>
<td>38</td>
</tr>
<tr>
<td>geom_raster_interactive</td>
<td>39</td>
</tr>
<tr>
<td>geom_rect_interactive</td>
<td>40</td>
</tr>
<tr>
<td>geom_ribbon_interactive</td>
<td>42</td>
</tr>
<tr>
<td>geom_sf_interactive</td>
<td>43</td>
</tr>
<tr>
<td>geom_smooth_interactive</td>
<td>45</td>
</tr>
<tr>
<td>geom_spoke_interactive</td>
<td>46</td>
</tr>
<tr>
<td>geom_text_repel_interactive</td>
<td>47</td>
</tr>
<tr>
<td>geom_violin_interactive</td>
<td>49</td>
</tr>
<tr>
<td>ggiraph</td>
<td>50</td>
</tr>
<tr>
<td>ggiraphOutput</td>
<td>52</td>
</tr>
<tr>
<td>girafe</td>
<td>52</td>
</tr>
<tr>
<td>girafeOutput</td>
<td>54</td>
</tr>
<tr>
<td>girafe_css</td>
<td>55</td>
</tr>
<tr>
<td>girafe_options</td>
<td>56</td>
</tr>
<tr>
<td>guide_bins_interactive</td>
<td>57</td>
</tr>
<tr>
<td>guide_colourbar_interactive</td>
<td>59</td>
</tr>
<tr>
<td>guide_coloursteps_interactive</td>
<td>62</td>
</tr>
<tr>
<td>guide_legend_interactive</td>
<td>64</td>
</tr>
<tr>
<td>interactive_circle_grob</td>
<td>68</td>
</tr>
<tr>
<td>interactive_curve_grob</td>
<td>69</td>
</tr>
<tr>
<td>interactive_params</td>
<td>69</td>
</tr>
<tr>
<td>interactive_path_grob</td>
<td>72</td>
</tr>
<tr>
<td>interactive_points_grob</td>
<td>72</td>
</tr>
<tr>
<td>interactive_polygon_grob</td>
<td>73</td>
</tr>
<tr>
<td>interactive_polyline_grob</td>
<td>74</td>
</tr>
<tr>
<td>interactive_raster_grob</td>
<td>74</td>
</tr>
<tr>
<td>interactive_rect_grob</td>
<td>75</td>
</tr>
<tr>
<td>interactive_roundrect_grob</td>
<td>76</td>
</tr>
</tbody>
</table>
annotate_interactive

Create interactive annotations

Description

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

Usage

```r
annotate_interactive(...)```

Arguments

```r
...  # 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

\texttt{girafe()}, \texttt{interactive\_parameters}, \texttt{annotation\_raster\_interactive()}

Examples

\begin{verbatim}
# 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)
\end{verbatim}
Examples

```r
# 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.

In order to generate the output, used fonts must be available on the computer used to create the svg, used fonts must also be available on the computer used to render the svg.

### Usage

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

### Arguments

- `file` the file where output will appear.
- `height, width` Height and width in inches.
### Description

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

### Usage

```
dsvg_view(code, ...)```

```r
dsvg_view <- tempfile(fileext = ".svg")
dsvg(file = dsvg_view)
plot(rnorm(10), main="Simple Example", xlab = "", ylab = "")
dev.off()
```

### See Also

- **Devices**

---

R family | Font on Windows | Font on Unix | Font on Mac OS
---|---|---|---
sans | Arial | DejaVu Sans | Helvetica
serif | Times New Roman | DejaVu serif | Times
mono | Courier | DejaVu mono | Courier
symbol | Symbol | DejaVu Sans | Symbol

As an example, using fonts = list(sans = "Roboto") would make the default font "Roboto" as many ggplot theme are using theme_minimal(base_family="") or theme_minimal(base_family="sans"). You can also use theme_minimal(base_family="Roboto").

---

**b**g 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.

**setdims** If TRUE (the default), the svg node will have attributes width & height set

**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()`.

If fonts are available, the default mapping will use these values:

```r
R family | Font on Windows | Font on Unix | Font on Mac OS
---|---|---|---
sans | Arial | DejaVu Sans | Helvetica
serif | Times New Roman | DejaVu serif | Times
mono | Courier | DejaVu mono | Courier
symbol | Symbol | DejaVu Sans | Symbol
```

### Examples

```r
fileout <- tempfile(fileext = ".svg")
dsvg(file = fileout)
plot(rnorm(10), main="Simple Example", xlab = "", ylab = ")
dev.off()
```
element_interactive

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)

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")

# 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",
      hover_css = "fill:red;stroke:none;"
    ),
    axis.title.y = element_text_interactive(
      data_id = "axis.title.y",
      tooltip = "Description for y axis",
      hover_css = "fill:red;stroke:none;"
    ),
    panel.grid.major = element_line_interactive(
      data_id = "panel.grid",
      tooltip = "Major grid lines",
      hover_css = "fill:none;stroke:red;"
    )
  )
  labs(
    title = "Interactive points example!",
    subtitle = label_interactive(""
"by ggiraph",
tooltip = "Click me!",
onclick = "window.open("https://davidgohel.github.io/ggiraph/")",
hover_css = "fill:magenta;cursor:pointer;"
)

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

---

font_family_exists  Check if font family exists.

Description

Check if a font family exists in system fonts.

Usage

font_family_exists(font_family = "sans")

Arguments

font_family  font family name (case sensitive)

Value

A logical value

See Also

Other functions for font management: match_family(), validated_fonts()

Examples

font_family_exists("sans")
font_family_exists("Arial")
font_family_exists("Courier")
geom_abline_interactive

Create interactive reference lines

Description

These geometries are based on `geom_abline()`, `geom_hline()` and `geom_vline()`.

Usage

```r
geom_abline_interactive(...)
geom_hline_interactive(...)
geom_vline_interactive(...)
```

Arguments

```r
...
```

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

```r
girafe()
girafe()
girafe()
```

Examples

```r
# add diagonal interactive reference lines to a ggplot -------
library(ggplot2)
library(ggiraph)

p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
g <- p + geom_abline_interactive(intercept = 20, tooltip = 20)
x <- girafe(ggobj = g)
if (interactive())
  print(x)

l <- coef(lm(mpg ~ wt, data = mtcars))
```
library(ggplot2)
library(ggiraph)

if( requireNamespace("dplyr", quietly = TRUE)){
  g1 <- ggplot(economics, aes(x = date, y = unemploy)) + geom_point() + geom_line()
  gg_hline1 <- g1 + geom_hline_interactive(aes(yintercept = mean(unemploy),
                                              tooltip = round(mean(unemploy), 2)), size = 3)
  x <- girafe(ggobj = gg_hline1)
  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 horizontal interactive reference lines to a ggplot -------
library(ggplot2)
library(ggiraph)

# 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**

...  
arguments passed to base function, plus any of the `interactive_parameters()`.
Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

`girafe()`

Examples

```r
# add interactive bar -------
library(ggplot)
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" ),
gender = c( "Male", "Female", "Female" ),
height = c(172, 159, 71 ) )
p <- ggplot(dat, aes( x = name, y = height, tooltip = gender,
data_id = name )) +
geom_col_interactive()

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

# an example with interactive guide ----
inf <- 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") +
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 = p)
if( interactive() ) print(x)
```
geom_bin_2d_interactive

Create interactive heatmaps of 2d bin counts

Description

The geometry is based on geom_bin_2d(). See the documentation for those functions for more details.

Usage

geom_bin_2d_interactive(...)

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

Examples

# add interactive bin2d heatmap to a ggplot ----
library(ggplot2)
library(ggiraph)

p <- ggplot(diamonds, aes(x, y, fill=cut)) + xlim(4, 10) + ylim(4, 10)+ geom_bin2d_interactive(aes(tooltip = cut), bins = 30)

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

Create interactive boxplot

Description
The geometry is based on \texttt{geom_boxplot()}. See the documentation for those functions for more details.

Usage

\begin{verbatim}
geom_boxplot_interactive(...)
\end{verbatim}

Arguments

\begin{verbatim}
...
\end{verbatim}

arguments passed to base function, plus any of the \texttt{interactive_parameters()}. 

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

Examples

\begin{verbatim}
# 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)
\end{verbatim}
geom_contour_interactive

Create interactive 2d contours of a 3d surface

Description
These geometries are based on `geom_contour()` and `geom_contour_filled()`. See the documentation for those functions for more details.

Usage

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

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

`girafe()`

Examples

```r
# 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)  

if (packageVersion("grid") >= numeric_version("3.6")) {  
p <- v + geom_contour_filled_interactive(aes(  
  colour = stat(level),  
  fill = stat(level)  
))  
x <- girafe(ggobj = p)  
if (interactive()) print(x)  
```
geom_count_interactive

Create interactive point counts

Description

The geometry is based on geom_bin2d(). See the documentation for those functions for more details.

Usage

geom_count_interactive(...)

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

description

Examples

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

p <- ggplot(mpg, aes(cty, hwy)) +
  geom_count_interactive(aes(tooltip=after_stat(n)))
x <- girafe(ggobj = p)
if( interactive() ) print(x)

p2 <- ggplot(diamonds, aes(x = cut, y = clarity)) +
geom_crossbar_interactive

```r
gem_count_interactive(aes(size = after_stat(prop),
      tooltip = after_stat(round(prop, 3), group = 1)) +
  scale_size_area(max_size = 10)
x <- girafe(ggobj = p2)
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**

```r
geom_crossbar_interactive(...)  
geom_errorbar_interactive(...)  
geom_linerange_interactive(...)  
geom_pointrange_interactive(...)  
```

**Arguments**

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

**Details for interactive geom 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. 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_curve_interactive

Create interactive line segments and curves

Description

The geometries are based on geom_segment() and geom_curve(). See the documentation for those functions for more details.

Usage

geom_curve_interactive(...)

geom_segment_interactive(...)

Arguments

... arguments passed to base function, plus any of the interactive_parameters().
Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

Examples

```r
# add interactive segments and curves 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),
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)

df <- data.frame(x1 = 2.62, x2 = 3.57, y1 = 21.0, y2 = 15.0)
p <- ggplot(df, aes(x = x1, y = y1, xend = x2, yend = y2)) +
  geom_curve_interactive(aes(colour = "curve", tooltip=I("curve"))) +
  geom_segment_interactive(aes(colour = "segment", tooltip=I("segment")))

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

geom_density_2d_interactive

Create interactive contours of a 2d density estimate

Description

The geometries are based on geom_density_2d() and geom_density_2d_filled(). See the documentation for those functions for more details.

Usage

geom_density_2d_interactive(...)

geom_density_2d_filled_interactive(...)

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

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, "\neruptions:", 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)
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 interactive geom 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. In this way they can be set to a scalar value.
geom_dotplot_interactive

Create interactive dot plots

Description

This geometry is based on `geom_dotplot()`. See the documentation for those functions for more details.

Usage

`geom_dotplot_interactive(…)`
Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

Examples

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

p <- ggplot(mtcars, aes(x = mpg, fill = factor(cyl))) +
  geom_dotplot_interactive(
    aes(tooltip = row.names(mtcars)),
    stackgroups = TRUE, binwidth = 1, method = "histodot"
  )

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

gg_point = ggplot(
  data = mtcars,
  mapping = aes(
    x = factor(vs), fill = factor(cyl), y = mpg,
    tooltip = row.names(mtcars)) +
    geom_dotplot_interactive(binaxis = "y",
                             stackdir = "center", position = "dodge")
  )

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

---

**geom_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

gem_errorbarh_interactive(...)

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

Examples

# 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)

# Define the top and bottom of the errorbars

# set the height of the errorbars to 0.2

# Define the top and bottom of the errorbars

# set the height of the errorbars to 0.2
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

```r
geom_freqpoly_interactive(...)  
geom_histogram_interactive(...)  
```

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

`girafe()`

Examples

```r
# 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)
```
geom_hex_interactive

`geom_hex_interactive(...)`

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

`girafe()`

Examples

```r
# add interactive hexagonal heatmaps to a ggplot
library(ggplot2)
library(ggiraph)

p <- ggplot(diamonds, aes(carat, price)) +
  geom_hex_interactive(aes(tooltip = after_stat(count)), bins = 10)
x <- girafe(ggobj = p)
if( interactive() ) print(x)
```
**geom_jitter_interactive**

*Create interactive jittered points*

---

**Description**

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

**Usage**

```r
geom_jitter_interactive(...)```

**Arguments**

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

**Details for interactive geom 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. 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(...)

geom_text_interactive(...)
```

Arguments

```r
...
```

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

`girafe()`

Examples

```r
# add interactive labels to a ggplot -------
library(ggplot2)
library(ggit)

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)

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",
    )
```
geom_map_interactive

    fontface = "bold")

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

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

## 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(check_overlap = TRUE) +
    coord_cartesian(xlim = c(0,50))

## 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)

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 interactive geom 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. In this way they can be set to a scalar value.

See Also
girafe()

Examples

# add interactive maps to a ggplot -------
library(ggplot2)
library(ggiraph)
crimes <- data.frame(state = tolower(rownames(USArrests)), USArrests)

# create tooltips and onclick events
states_ <- sprintf("<p>%s</p>",
  as.character(crimes$state) )
table_ <- paste0(
  "<table><tr><td>UrbanPop</td>
  , sprintf("<td>%.0f</td>", crimes$UrbanPop),
  "</tr></tr>",
  "<td>Assault</td>
  , sprintf("<td>%.0f</td>", crimes$Assault),
  "</tr></table>"
)
onclick <- sprintf(
  "window.open("%s%s")",
  http://en.wikipedia.org/wiki/",
  as.character(crimes$state)
)

if (require("maps") ) {
  states_map <- map_data("state")
gg_map <- ggplot(crimes, aes(map_id = state))
gg_map <- gg_map + geom_map_interactive(aes(
    fill = Murder,
    tooltip = labs,
    data_id = state,
    onclick = onclick
  ),
  map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat)
x <- girafe(ggobj = gg_map)
if( interactive() ) print(x)
}
Description

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

Usage

```r
geom_path_interactive(...)
geom_line_interactive(...)
geom_step_interactive(...)
```

Arguments

```r
...
```

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

Details for interactive geom 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. 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 = "Unemployed stairstep line", data_id = 1))
  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(quotesingle.Var
line
quotesingle.Var, data$id)
data$onclick = paste0("alert("data$id", ")") 
cols = c("orange", "orange1", "orange2", "navajowhite4", "navy")
}
data2 <- 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("label")") 
)

# 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)

gg_path_2 = ggplot(data, aes(variable, value, group = id, data_id = id,
 tooltip = tooltip)) +
 geom_path_interactive(alpha = 0.5) +
 facet_wrap( ~ group )

gg_path_3 = ggplot(dataset2) +
 geom_path_interactive(aes(x, y, group=grp, data_id = label,
## geom_point_interactive

Create interactive points

### Description

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

### Usage

```r
geom_point_interactive(...)```

### Arguments

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

### Details for interactive geom 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. 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_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)
```

---

**geom_polygon_interactive**

Create interactive polygons

---

Description

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

Usage

`geom_polygon_interactive(...)`

Arguments

... arguments passed to base function, plus any of the `interactive_parameters()`.
Details for interactive geom 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 \texttt{geom*_interactive} function. 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(
id = rep(ids, each = 4),
x = c(2, 1, 1.1, 2.2, 1, 0, 0.3, 1.1, 2.2, 1.1, 1.2, 2.5, 1.1, 0.3,
0.5, 1.2, 2.5, 1.2, 1.3, 2.7, 1.2, 0.5, 0.6, 1.3),
y = c(-0.5, 0, 1, 0.5, 0, 0.5, 1.5, 1, 0.5, 1, 2.1, 1.7, 1, 1.5,
2.2, 2.1, 1.7, 2.1, 3.2, 2.8, 2.1, 2.2, 3.3, 3.2) )
datapoly <- merge(values, positions, by=c("id"))
datapoly$oc = "alert(this.getAttribute("data-id"))"

# create a ggplot -----
# As of R version 3.6 geom_polygon() supports polygons with holes
# Use the subgroup aesthetic to differentiate holes from the main polygon
holes <- do.call(rbind, lapply(split(datapoly, datapoly$id), function(df) {
df$x <- df$x + 0.5 * (mean(df$x) - df$x)
df$y <- df$y + 0.5 * (mean(df$y) - df$y)
df
}))

# display ------
x <- girafe(ggobj = gg_poly_1)
if( interactive() ) print(x)
```

if (packageVersion("grid") >= "3.6") {
    # As of R version 3.6 geom_polygon() supports polygons with holes
    # Use the subgroup aesthetic to differentiate holes from the main polygon
    holes <- do.call(rbind, lapply(split(datapoly, datapoly$id), function(df) {
        df$x <- df$x + 0.5 * (mean(df$x) - df$x)
        df$y <- df$y + 0.5 * (mean(df$y) - df$y)
        df
    }))
```
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

geom_quantile_interactive(...)

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()
Examples

```r
# 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.

Usage

`geom_raster_interactive(...)`

Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.
See Also

girafe()
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)

Description

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

Usage

geom_rect_interactive(...)

geom_tile_interactive(...)

Arguments

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

Details for interactive geom 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. 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,
                                       tooltip = tooltip, onclick = oc, data_id = uid ),
                         color="black", alpha=0.5) +
  geom_text(data=dataset,
            aes(x = x1 + ( x2 - x1 ) / 2, y = y1 + ( y2 - y1 ) / 2,
                 label = r ),
            size = 4 )

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

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

df <- data.frame(
    id = rep(c("a", "b", "c", "d", "e"), 2),
    x = rep(c(2, 5, 7, 9, 12), 2),
    y = rep(c(1, 2), each = 5),
    z = factor(rep(1:5, each = 2)),
    w = rep(diff(c(0, 4, 6, 8, 10, 14)), 2) )
```
p <- ggplot(df, aes(x, y, tooltip = id)) + geom_tile_interactive(aes(fill = z))
x <- girafe(ggobj = p)
if( interactive() ) print(x)

# correlation dataset ----
cor_mat <- cor(mtcars)
diag( cor_mat ) <- NA
var1 <- rep( row.names(cor_mat), ncol(cor_mat) )
var2 <- rep( colnames(cor_mat), each = nrow(cor_mat) )
cor <- as.numeric(cor_mat)
cor_mat <- data.frame( var1 = var1, var2 = var2,
                      cor = cor, stringsAsFactors = FALSE )
cor_mat["tooltip"] <-
  sprintf("<i>/grave.%s</i> vs <i>/grave.%s</i>:</br><code>%.03f</code>",
             var1, var2, cor)
p <- ggplot(data = cor_mat, aes(x = var1, y = var2) ) +
  geom_tile_interactive(aes(fill = cor, tooltip = tooltip), colour = "white") +
  scale_fill_gradient2_interactive(low = "#BC120A", mid = "white", high = "#BC120A",
                                   limits = c(-1, 1), data_id = "cormat", tooltip = "cormat") +
  coord_equal()
x <- girafe(ggobj = p)
if( interactive() ) print(x)

---

**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**

`geom_ribbon_interactive(...)`

`geom_area_interactive(...)`

**Arguments**

...  

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

**Details for interactive geom 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. 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))
g <- h +
  geom_ribbon_interactive(aes(ymin = level - 1, ymax = level + 1),
                           fill = "grey70", tooltip = "ribbon1", data_id="ribbon1",
                           outline.type = "both",
                           hover_css = "stroke:red;stroke-width:inherit;") +
  geom_line_interactive(aes(y = level), tooltip = "level", data_id="line1",
                        hover_css = "stroke:orange;fill:none;")
x <- girafe(ggobj = g)
x <- girafe_options(x = x,
                    opts_hover(css = girafe_css(
                                  css = "stroke:orange;stroke-width:3px;",
                                  area = "fill:blue;"))))
if( interactive() ) print(x)

g <- h + geom_area_interactive(aes(y = level), tooltip = "area1")
x <- girafe(ggobj = g)
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 interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

Examples

# 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 = ">=", 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 interactive geom 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. 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(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)
```

---

**geom_spoke_interactive**

Create interactive line segments parameterised by location, direction and distance

---

Description

The geometry is based on `geom_spoke()`. See the documentation for those functions for more details.

Usage

`geom_spoke_interactive(...)`

Arguments

... arguments passed to base function, plus any of the `interactive_parameters()`.
Details for interactive geom 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. In this way they can be set to a scalar value.

See Also

girafe()

Examples

# add interactive line segments parameterised by location, direction and distance to a ggplot -------
library(ggplot2)
library(ggiraph)

df <- expand.grid(x = 1:10, y=1:10)
df$angle <- runif(100, 0, 2*pi)
df$speed <- runif(100, 0, sqrt(0.1 * df$x))

p <- ggplot(df, aes(x, y)) +
  geom_point() +
  geom_spoke_interactive(aes(angle = angle, tooltip=round(angle, 2)), radius = 0.5)
x <- girafe(ggobj = p)
if( interactive() ) print(x)

p2 <- ggplot(df, aes(x, y)) +
  geom_point() +
  geom_spoke_interactive(aes(angle = angle, radius = speed, tooltip=paste(round(angle, 2), round(speed, 2), sep="\n")))

x2 <- girafe(ggobj = p2)
if( interactive() ) print(x2)

---

Geom_text_repel_interactive

Create interactive repulsive textual annotations

Description

The geometries are based on `ggrepel::geom_text_repel()` and `ggrepel::geom_label_repel()`. See the documentation for those functions for more details.

Usage

geom_text_repel_interactive(...)

geom_label_repel_interactive(...)
Arguments

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

Details for interactive geom 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. In this way they can be set to a scalar value.

Note

The `ggrepel` package is required for these geometries

See Also

girafe()

Examples

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

# geom_text_repel_interactive
if (requireNamespace("ggrepel", quietly = TRUE)) {
  dataset = mtcars
dataset$label = row.names(mtcars)
dataset$tooltip = paste0(dataset$label, "<br/>", "cyl: ", dataset$cyl, "<br/>",
  "gear: ", dataset$gear, "<br/>",
  "carb: ", dataset$carb)
p <- ggplot(dataset, aes(wt, mpg, color = qsec )) +
  geom_point_interactive(aes(tooltip = tooltip, data_id = label))

  gg_text = p +
  geom_text_repel_interactive(
    aes(label = label, tooltip = tooltip, data_id = label),
    size = 3
  )
}

x <- girafe(ggobj = gg_text)
x <- girafe_options(x = x,
  opts_hover(css = "fill:#FF4C3B;") )
if (interactive()) print(x)
}

# geom_label_repel_interactive
if (requireNamespace("ggrepel", quietly = TRUE)) {
  gg_label = p +
```
Create interactive violin plot

**Description**

The geometry is based on `geom_violin()`. See the documentation for those functions for more details.

**Usage**

`geom_violin_interactive(...)`

**Arguments**

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

**Details for interactive geom 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. In this way they can be set to a scalar value.

**See Also**

`girafe()`
ggiraph

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,
  ...
)
Arguments

code             Plotting code to execute

```r
library(ggplot2)
library(ggiraph)
```

ggobj            ggplot object to print. Argument code will be ignored if this argument is supplied.

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

```r
pointsize <- 12
```    

width            widget width ratio (0 < width <= 1).

```r
width <- 0.5
```    

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.

tooltip_extra_css extra css (added to position: absolute; pointer-events: none;) used to customize tooltip area.

```r
tooltip_extra_css <- "position: absolute; pointer-events: none;"
```    

``` r
hover_css <- "color: blue; opacity: 0.5;"
```    

tooltip_opacity

``` r
tooltip_opacity <- 0.8
```    

tooltip_offx
tooltip x offset

``` r
tooltip_offx <- 10
```    

tooltip_offy
tooltip y offset

``` r
tooltip_offy <- 20
```    

tooltip_zindex
 tooltip css z-index, default to 999.

``` r
tooltip_zindex <- 999
```    

zoom_max         maximum zoom factor

``` r
zoom_max <- 4
```    

selection_type   row selection mode ("single", "multiple", "none") when widget is in a Shiny application.

``` r
selection_type <- "single"
```    

selected_css     css to apply when element is selected (shiny only).

``` r
selected_css <- "color: red;"
```    

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

``` r
dep_dir <- ""
```    

... arguments passed on to dsvg

Examples

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


dataset

``` r
# 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**

```r
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.
girafe

Usage

girafe(
  code,
  ggobj = NULL,
  pointsize = 12,
  width_svg = 6,
  height_svg = 5,
  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_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.

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() method.

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 deactivate" and "zoom init" buttons are available in a toolbar.

When selection type is set to 'multiple' (in Shiny applications), lasso selection and lasso anti-selection 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\% 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 beeing 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(), validated_fonts(), dsvg()

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**

```r
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**

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

**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:polyline, svg:path)
- **image**: Override style for image elements (svg:image)

**Value**

- css as scalar character
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

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

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()
guide_bins_interactive

```r
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_bins_interactive**

Create interactive bins guide

---

**Description**

The guide is based on `guide_bins()`. See the documentation for that function for more details.

**Usage**

`guide_bins_interactive(...)`

**Arguments**

`...` arguments passed to base function.

**Value**

An interactive guide object.

**Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a `guide_legend_interactive()` or `guide_bins_interactive()` respectively, 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 or coloursteps is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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 bins guide 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 binned scale and guide
p1 <- p + scale_fill_viridis_b_interactive(data_id = "nlevel",
  tooltip = "nlevel",
  guide = "bins")
x <- girafe(ggobj = p1)
if (interactive()) print(x)

# set the keys separately
p2 <- p + scale_fill_viridis_b_interactive(
  data_id = function(breaks) {
    as.character(breaks)
  },
  tooltip = function(breaks) {
    as.character(breaks)
  },
  guide = "bins"
)
x <- girafe(ggobj = p2)
if (interactive()) print(x)

# make the title and labels interactive
p3 <- p + scale_fill_viridis_c_interactive(
  data_id = function(breaks) {
    as.character(breaks)
  },
  title = "interactive title",
  label = "interactive label"
)
```
guide_colourbar_interactive

```r
tooltip = function(breaks) {
  as.character(breaks)
},
guide = "bins",
name = label_interactive("nlevel", data_id = "nlevel",
tooltip = "nlevel"),
labels = function(breaks) {
  label_interactive(
    as.character(breaks),
    data_id = as.character(breaks),
onclick = paste0("alert("", as.character(breaks), ")"),
tooltip = 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)
```

---

**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 interactive scale and interactive guide 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 or bins is used, it will be converted to a `guide_legend_interactive()` or `guide_bins_interactive()` respectively, 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 or coloursteps is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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")(x)) + 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("x",
data_id = "colourbar",
onclick = "alert("colourbar")",
tooltip = "colourbar"
),
labels = function(breaks) {
    br <- na.omit(breaks)
    label_interactive(
        as.character(breaks),
data_id = paste0("colourbar", br),
onclick = "alert("colourbar")",
tooltip = paste0("colourbar", br)
    )
}
)
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"
)
)}
guide_coloursteps_interactive

Create interactive colorsteps guide

Description

The guide is based on `guide_coloursteps()`. See the documentation for that function for more details.

Usage

`guide_coloursteps_interactive(...)`

`guide_colorsteps_interactive(...)`

Arguments

... arguments passed to base function.

Value

An interactive guide object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a \texttt{guide_legend_interactive()} or \texttt{guide_bins_interactive()} respectively, 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 or coloursteps is used, it will be converted to a \texttt{guide_colourbar_interactive()} or \texttt{guide_coloursteps_interactive()} respectively, 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 \texttt{title.theme} and \texttt{label.theme} can be defined as \texttt{element_text_interactive} (in fact, they will be converted to that if they are not already), either directly or via the theme. See the \texttt{element_*_interactive} section for more details.

\textbf{See Also}

\texttt{interactive_parameters()}, \texttt{girafe()}

\textbf{Examples}

```r
# add interactive coloursteps guide 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 binned scale, by default the guide is colorsteps
p1 <- p + scale_fill_viridis_b_interactive(data_id = "nlevel",
                                          tooltip = "nlevel")
x <- girafe(ggobj = p1)
if (interactive()) print(x)

# make the title and labels interactive
p2 <- p + scale_fill_viridis_b_interactive(
    data_id = "nlevel",
tooltip = "nlevel",
    name = label_interactive("nlevel", data_id = "nlevel",
                              tooltip = "nlevel"),
```

labels = function(breaks) {
  1 <- lapply(breaks, function(br) {
    label_interactive(
      as.character(br),
      data_id = as.character(br),
      onclick = paste0("alert("", as.character(br), ")"),
      tooltip = as.character(br)
    )
  })
  1
}

x <- girafe(ggobj = p2)
x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)

---

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 interactive scale and interactive guide 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 or bins is used, it will be converted to a guide_legend_interactive()
or guide_bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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 -------
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 = 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),
                    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 = p5)
x <- girafe_options(x,
    opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red")))
if (interactive()) print(x)
guide_legend_interactive

```r

data_id = as.character(br),
onclick = paste0("alert(\"", as.character(br), "\\\")
```

**Example Usage:**

```r
guide_legend_interactive
```

---

```r
# add interactive continuous legend guide 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",
                                         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(
  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
```

---

This code demonstrates how to create an interactive continuous legend guide for a `ggplot2` visualization. It leverages the `girafe` package to enable dynamic interactions with the legend, such as clicking on legend entries to highlight data points on the plot. The example uses a synthetic dataset of diamonds to illustrate the functionality, adding a 2D density plot with interactive color scaling and a custom legend guide.
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) {
    label_interactive(
      as.character(breaks),
      data_id = as.character(breaks),
      onclick = paste0("alert("", as.character(breaks), ", \")\")
    )
  }
)
)x <- girafe(ggobj = p4)
)x <- girafe_options(x,
  opts_hover_key(girafe_css("stroke:red", text="stroke:none;fill:red"))
)if (interactive()) print(x)

---

**interactive_circle_grob**

Create interactive circles grob

**Description**

The grob is based on `circleGrob()`. See the documentation for that function for more details.

**Usage**

`interactive_circle_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.
**interactive_curve_grob**

Create interactive curve grob

---

**Description**

The grob is based on `curveGrob()`. See the documentation for that function for more details.

**Usage**

```r
interactive_curve_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_parameters**

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. To use html markup it is advised to use `htmltools::HTML()` function in order to mark the text as html markup. If the text is not marked as html and no opening/closing tags were detected, then any existing newline characters (`\r\n`, `\r` and `\n`) are replaced with the `<br/>` tag.

**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()`, `opts_hover_key()` or `opts_hover_theme()`. It can also be constructed with `girafe_css`, to give more control over the css for different element types (see `opts_hover()` note).

**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()`, `opts_selection_key()` or `opts_selection_theme()`. It can also be constructed with `girafe_css`, to give more control over the css for different element types (see `opts_selection()` note).

**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.

**tooltip_fill**

Color to use for tooltip background when `opts_tooltip()` use_fill is TRUE. Useful for setting the tooltip background color in `geom_text_interactive()` or `geom_label_interactive()`, when the geom text color may be the same as the tooltip text color.

### Details for interactive geom 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. 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 interactive scale and interactive guide 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.
interactive_parameters

- When guide of type legend or bins is used, it will be converted to a `guide_legend_interactive()` or `guide_bins_interactive()` respectively, 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 or coloursteps is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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.

Custom interactive parameters

The argument `extra_interactive_params` can be passed to any of the `*_interactive` functions (geoms, grobs, scales, labeller, labels and theme elements), It should be a character vector of additional names to be treated as interactive parameters when evaluating the aesthetics. The values will eventually end up as attributes in the SVG elements of the output.

Intended only for expert use.

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.

**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_points_grob**  
*Create 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_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

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

interactive_parameters(), 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

interactive_rect_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_roundrect_grob

Create interactive rectangle grob

Description

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

Usage

interactive_roundrect_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

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()
labeller_interactive  

Construct interactive labelling specification for facet strips

Description
This function is a wrapper around labeller() that allows the user to turn facet strip labels into interactive labels via label_interactive().

It requires that the theme()'s strip.text elements are defined as interactive theme elements via element_text_interactive(), see details.

Usage
labeller_interactive(.mapping = NULL, ...)

Arguments
.mapping set of aesthetic mappings created by aes() or aes_(). It should provide mappings for any of the interactive_parameters(). In addition it understands a label parameter for creating a new label text.

... arguments passed to base function labeller()

Details
The aesthetics set provided via .mapping is evaluated against the data provided by the ggplot2 facet. This means that the variables for each facet are available for using inside the aesthetic mappings. In addition the .label variable provides access to the produced label. See the examples.

The plot's theme is required to have the strip texts as interactive text elements. This involves strip.text or individually strip.text.x and strip.text.y: theme(strip.text.x = element_text_interactive())

theme(strip.text.y = element_text_interactive())

See Also
labeller(),label_interactive(),labellers

Examples
# use interactive labeller
library(ggplot2)
library(ggiraph)

p1 <- ggplot(mtcars, aes(x = mpg, y = wt)) +
  geom_point_interactive(aes(tooltip = row.names(mtcars)))

# Always remember to set the theme's strip texts as interactive
# no need to set any interactive parameters, they'll be assigned from the labels
p1 <- p1 +
  theme(
    strip.text.x = element_text_interactive(),
  )
strip.text.y = element_text_interactive()

# simple facet
p <- p1 + facet_wrap(
  vars(gear),
  labeller = labeller_interactive(aes(tooltip = paste("Gear: ", gear)))
)
x <- girafe(ggobj = p)
if (interactive()) print(x)

# With two vars. When the .multi_line labeller argument is TRUE (default),
# supply a different labeller for each var
p <- p1 + facet_wrap(
  vars(gear, vs),
  labeller = labeller_interactive(
    gear = labeller_interactive(aes(tooltip = paste("Gear: ", gear))),
    vs = labeller_interactive(aes(tooltip = paste("VS: ", vs)))
  )
)
x <- girafe(ggobj = p)
if (interactive()) print(x)

# When the .multi_line argument is FALSE, the labels are joined and
# the same happens with the data, so we can refer to both variables in the aesthetics!

p <- p1 + facet_wrap(
  vars(gear, vs),
  labeller = labeller_interactive(
    aes(tooltip = paste0("Gear: ", gear, "\nVS: ", vs)),
    .multi_line = FALSE
  )
)
x <- girafe(ggobj = p)
if (interactive()) print(x)

# Example with facet_grid:
p <- p1 + facet_grid(
  vs + am ~ gear,
  labeller = labeller(
    gear = labeller_interactive(aes(
      tooltip = paste("gear: ", gear), data_id = paste0("gear_", gear)),
    ),
    vs = labeller_interactive(aes(
      tooltip = paste("VS: ", vs), data_id = paste0("vs_", vs)),
    ),
    am = labeller_interactive(aes(
      tooltip = paste("AM: ", am), data_id = paste0("am_", am)),
    ))
  )
x <- girafe(ggobj = p)
if (interactive()) print(x)
# Same with .rows and .cols and .multi_line = FALSE
p <- p1 + facet_grid(
  vs + am ~ gear,
  labeller = labeller(
    .cols = labeller_interactive(
      .mapping = aes(tooltip = paste("gear:", gear))
    ),
    .rows = labeller_interactive(
      aes(tooltip = paste0("VS: ", vs, "\nAM: ", am)),
      .multi_line = FALSE
    )
  )
)

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

# a more complex example
p2 <- ggplot(msleep, aes(x = sleep_total, y = awake)) +
  geom_point_interactive(aes(tooltip = name)) +
  theme(
    strip.text.x = element_text_interactive(),
    strip.text.y = element_text_interactive()
  )

# character vector as lookup table
conservation_status <- c(
  cd = "Conservation Dependent",
  en = "Endangered",
  lc = "Least concern",
  nt = "Near Threatened",
  vu = "Vulnerable",
  domesticated = "Domesticated"
)

# function to capitalize a string
capitalize <- function(x) {
  substr(x, 1, 1) <- toupper(substr(x, 1, 1))
  x
}

# function to cut a string and append an ellipsis
cut_str <- function(x, width = 10) {
  ind <- !is.na(x) & nchar(x) > width
  x[ind] <- paste0(substr(x[ind], 1, width), "...")
  x
}

replace_nas <- function(x) {
  ifelse(is.na(x), "Not available", x)
}

# in this example we use the `.label` variable to access the produced label
# and we set the `label` aesthetic to modify the label
p <- p2 + facet_grid(
  vore ~ conservation,
  labeller = labeller(
    vore = labeller_interactive(
      aes(tooltip = paste("Vore:", replace_nas(.label)),
          .default = capitalize
    ),
    conservation = labeller_interactive(
      aes(
        tooltip = paste("Conservation:\n", replace_nas(.label)),
        label = cut_str(.label, 3)
      ),
      .default = conservation_status
    )
  )
)

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

---

**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**

```r
label_interactive(label, ...)
```

**Arguments**

- `label`  
  - The text for the label (scalar character)
- `...`  
  - any of the `interactive_parameters()`.

**Value**

- an interactive label object

**See Also**

- `interactive_parameters`, `labeller_interactive()`
match_family

Examples

```r
library(ggplot2)
library(ggiraph)

gg_jitter <- ggplot(
  mpg, aes(cyl, hwy, group = cyl)) +
  geom_boxplot() +
  labs(title =
    label_interactive(
      "title",
      data_id = "id_title",
      onclick = "alert("title")",
      tooltip = "title"
    )
  ) +
  theme(plot.title = element_text_interactive())

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

match_family

Find best family match with systemfonts

Description

match_family() returns the best font family match.

Usage

```r
match_family(font = "sans", bold = TRUE, italic = TRUE, debug = NULL)
```

Arguments

- `font` family or face to match.
- `bold` Wheter to match a font featuring a bold face.
- `italic` Wheter to match a font featuring an italic face.
- `debug` deprecated

See Also

Other functions for font management: `font_family_exists()`, `validated_fonts()`

Examples

```r
match_family("sans")
match_family("serif")
```
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. Use `opts_hover_inv` for the effect on the rest of the geometries, while one is hovered (inverted operation).

Usage

```r
opts_hover(css = NULL, reactive = FALSE)
```

```r
opts_hover_inv(css = NULL)
```

```r
opts_hover_key(css = NULL, reactive = FALSE)
```

```r
opts_hover_theme(css = NULL, reactive = FALSE)
```

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.
- **reactive**: if TRUE, in Shiny context, hovering will set Shiny input values.

Note

**IMPORTANT**: When applying a `fill` style with the `css` argument, be aware that the browser’s CSS engine will apply it also to line elements, if there are any that use the hovering feature. This will cause an undesired effect.

To overcome this, supply the argument `css` using `girafe_css`, in order to set the `fill` style only for the desired elements.

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$cname = row.names(mtcars)
```
```r
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.

Note

IMPORTANT: When applying a fill style with the css argument, be aware that the browser’s CSS engine will apply it also to line elements, if there are any that use the selection feature. This will cause an undesired effect.

To overcome this, supply the argument css using girafe_css, in order to set the fill style only for the desired elements.

See Also

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

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_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

opts_sizing(rescale = TRUE, width = 1)
Arguments

rescale  If FALSE, graphic will not be resized and the dimensions are exactly those of
          the svg. If TRUE the graphic will be resize to fit its 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

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)

opts_toolbar  Toolbar settings

Description

Allows customization of the toolbar

Usage

opts_toolbar(position = "topright", saveaspng = TRUE, pngname = "diagram")

Arguments

position  one of 'top', 'bottom', 'topleft', 'topright', 'bottomleft', 'bottomright'
saveaspng set to TRUE to propose the 'save as png' button.
pngname   the default basename (without .png extension) to use for the png file.
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$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_toolbar(position = "top") )
if( interactive() ) print(x)
```

### opts_tooltip

**Tooltip settings**

**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,
placement = "auto",
```
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.
placement Defines the container used for the tooltip element. It can be one of "auto" (default), "doc" or "container".
  • doc: the host document's body is used as tooltip container. The tooltip may cover areas outside of the svg graphic.
  • container: the svg container is used as tooltip container. In this case the tooltip content may wrap to fit inside the svg bounds. It will also inherit the CSS styles and transforms applied to the parent containers (like scaling in a slide presentation).
  • auto: This is the default, ggiraph chooses the best option according to use cases. Usually it redirects to "doc", however in a xaringan context, it redirects to "container".

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

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_tooltip(opacity = .7,
    offx = 20, offy = -10,
    use_fill = TRUE, use_stroke = TRUE,
    delay_mouseout = 1000) )
if( interactive() ) print(x)

opts_zoom

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows customization of the zoom.</td>
</tr>
</tbody>
</table>

| 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) |
renderggiraph

Reactively version of ggmaph object

Description

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

Usage

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

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

renderGirafe

Reactively version of girafe

Description

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

Usage

renderGirafe(expr, env = parent.frame(), quoted = FALSE, outputArgs = list())

Arguments

expr An expression that returns a girafe() object.
env The environment in which to evaluate expr.
quoted Is expr a quoted expression
outputArgs A list of arguments to be passed through to the implicit call to girafeOutput() when renderGirafe is used in an interactive R Markdown document.
**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_binned()`, `scale_alpha_ordinal()`, `scale_alpha_date()`, `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_binned_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 interactive scale and interactive guide 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 or bins is used, it will be converted to a `guide_legend_interactive()` or `guide_bins_interactive()` respectively, 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 or coloursteps is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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_colour_steps_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()`, `scale_fill_distiller()`, `scale_colour_fermenter()`, `scale_fill_fermenter()`. See the documentation for those functions for more details.
**scale_colour_brewer_interactive**

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(...)  
scale_colour_fermenter_interactive(...)  
scale_color_fermenter_interactive(...)  
scale_fill_fermenter_interactive(...)

Arguments

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

Value

An interactive scale object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a guide_legend_interactive() or guide_bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a guide_colourbar_interactive() or guide_coloursteps_interactive() respectively, 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_colour_steps_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()`, `scale_fill_hue()`, `scale_colour_binned()`, `scale_fill_binned()`, `scale_colour_discrete()`, `scale_fill_discrete()`, `scale_colour_date()`, `scale_fill_date()`, `scale_colour_datetime()` and `scale_fill_datetime()`. 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_binned_interactive(...)
scale_color_binned_interactive(...)

---
scale.fill_binned_interactive(...)  
scale.colour_discrete_interactive(...)  
scale.color_discrete_interactive(...)  
scale.fill_discrete_interactive(...)  
scale.colour.date_interactive(...)  
scale.color.date_interactive(...)  
scale.fill.date_interactive(...)  
scale.colour.datetime_interactive(...)  
scale.color.datetime_interactive(...)  
scale.fill.datetime_interactive(...)  

Arguments  

...  

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

Value  

An interactive scale object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a guide.legend_interactive() or guide.bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a guide.colourbar_interactive() or guide.coloursteps_interactive() respectively, 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_steps_interactive(), scale_gradient_interactive, scale_linetype_interactive(),
scale_manual_interactive, scale_shape_interactive(), scale_size_interactive(), scale_viridis_interactive

scale_colour_steps_interactive

Create interactive binned gradient colour scales

Description
These scales are based on scale_colour_steps(), scale_fill_steps(), scale_colour_steps2(),
scale_fill_steps2(), scale_colour_stepsn() and scale_fill_stepsn(). See the documentation for those functions for more details.

Usage

scale_colour_steps_interactive(...)
scale_color_steps_interactive(...)
scale_fill_steps_interactive(...)
scale_colour_steps2_interactive(...)
scale_color_steps2_interactive(...)
scale_fill_steps2_interactive(...)
scale_colour_stepsn_interactive(...)
scale_color_stepsn_interactive(...)
scale_fill_stepsn_interactive(...)  

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

Value
An interactive scale object.
Details for interactive scale and interactive guide 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` or `bins` is used, it will be converted to a `guide_legend_interactive()` or `guide_bins_interactive()` respectively, 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` or `coloursteps` is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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()`, `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.
scale_gradient_interactive

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 interactive scale and interactive guide 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 or bins is used, it will be converted to a guide_legend_interactive() or guide_bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a guide_colourbar_interactive() or guide_coloursteps_interactive() respectively, 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_colour_manual_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",
                                             labels = label_interactive(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)
```
name = label_interactive("z",
data_id = "colourbar",
onclick = "alert('colourbar')",
tooltip = "colourbar"),
labels = function(breaks) {
br <- na.omit(breaks)
label_interactive(as.character(breaks),
data_id = paste0("colourbar", br),
onclick = "alert('colourbar')",
tooltip = paste0("colourbar", br)
}
}
x <- girafe(ggobj = p3)
x <- girafe_options(x, options_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, options_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(), scale_linetype_discrete() and scale_linetype_binned(). See the documentation for those functions for more details.

Usage

scale_linetype_interactive(...)  
scale_linetype_continuous_interactive(...)  
scale_linetype_discrete_interactive(...)  
scale_linetype_binned_interactive(...)

Arguments

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

Value

An interactive scale object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a guide_legend_interactive() or guide_bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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_interactive`, `scale_colour_steps_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

`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(...)`
Arguments

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

Value

An interactive scale object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a guide_legend_interactive() or guide_bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a guide_colourbar_interactive() or guide_coloursteps_interactive() respectively, 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_colour_steps_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 = 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")",
  )
)
scale_shape_interactive

Create interactive scales for shapes

Description

These scales are based on scale_shape(), scale_shape_continuous(), scale_shape_discrete(), scale_shape_binned() and scale_shape_ordinal(). See the documentation for those functions for more details.

Usage

scale_shape_interactive(...)
scale_shape_continuous_interactive(...)

scale_shape_discrete_interactive(...)

scale_shape_binned_interactive(...)

scale_shape_ordinal_interactive(...)

Arguments

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

Value

An interactive scale object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a guide_legend_interactive() or guide_bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a guide_colourbar_interactive() or guide_coloursteps_interactive() respectively, 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_colour_steps_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_area()`, `scale_size_continuous()`, `scale_size_discrete()`, `scale_size_binned()`, `scale_size_binned_area()`, `scale_size_date()`, `scale_size_datetime()`, `scale_size_ordinal()` and `scale_radius()`. See the documentation for those functions for more details.

Usage

```
scale_size_interactive(...)  
scale_size_area_interactive(...)  
scale_size_continuous_interactive(...)  
scale_size_discrete_interactive(...)  
scale_size_binned_interactive(...)  
scale_size_binned_area_interactive(...)  
scale_size_date_interactive(...)  
scale_size_datetime_interactive(...)  
scale_size_ordinal_interactive(...)  
scale_radius_interactive(...)  
```

Arguments

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

Value

An interactive scale object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a `guide_legend_interactive()` or `guide_bins_interactive()` respectively, 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 or coloursteps is used, it will be converted to a `guide_colourbar_interactive()` or `guide_coloursteps_interactive()` respectively, 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_colour_steps_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()`, `scale_fill_viridis_c()`, `scale_colour_viridis_b()`, `scale_fill_viridis_b()`, `scaleColour_ordinal()`, `scale_fill_ordinal()`. See the documentation for those functions for more details.

**Usage**

`scale_colour_viridis_d_interactive(...)`

`scale_color_viridis_d_interactive(...)`

`scale_fill_viridis_d_interactive(...)`

`scale_colour_viridis_c_interactive(...)`
scale_viridis_interactive

scale_color_viridis_c_interactive(...)  
scale_fill_viridis_c_interactive(...)  
scale_colour_viridis_b_interactive(...)  
scale_color_viridis_b_interactive(...)  
scale_fill_viridis_b_interactive(...)  
scale_colour_ordinal_interactive(...)  
scale_color_ordinal_interactive(...)  
scale_fill_ordinal_interactive(...)

Arguments

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

Value

An interactive scale object.

Details for interactive scale and interactive guide 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 or bins is used, it will be converted to a guide_legend_interactive() or guide_bins_interactive() respectively, 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 or coloursteps is used, it will be converted to a guide_colourbar_interactive() or guide_coloursteps_interactive() respectively, 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_colour_steps_interactive(), scale_gradient_interactive, scale_linetype_interactive(), scale_manual_interactive, scale_shape_interactive(), scale_size_interactive()

Examples

# add interactive viridis scale to a ggplot -------
library(ggplot2)
library(ggitap)

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(
    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
validated_fonts

List of validated default fonts

Description

Validates and possibly modifies the fonts to be used as default value in a graphic according to the fonts available on the machine. It process elements named "sans", "serif", "mono" and "symbol".

Usage

validated_fonts(fonts = list())

Arguments

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(). If fonts are available, the default mapping will use these values:

<table>
<thead>
<tr>
<th>R family</th>
<th>Font on Windows</th>
<th>Font on Unix</th>
<th>Font on Mac OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>sans</td>
<td>Arial</td>
<td>DejaVu Sans</td>
<td>Helvetica</td>
</tr>
<tr>
<td>serif</td>
<td>Times New Roman</td>
<td>DejaVu serif</td>
<td>Times</td>
</tr>
<tr>
<td>mono</td>
<td>Courier</td>
<td>DejaVu mono</td>
<td>Courier</td>
</tr>
<tr>
<td>symbol</td>
<td>Symbol</td>
<td>DejaVu Sans</td>
<td>Symbol</td>
</tr>
</tbody>
</table>
Value

a named list of validated font family names

See Also

girafe(), dsvg()

Other functions for font management: font_family_exists(), match_family()

Examples

validated_fonts()
Index

* device
dsvg, 6

* functions for font management  
  font_family_exists, 10  
  match_family, 82  
  validated_fonts, 111

* girafe animation options  
  girafe_options, 56  
  opts_hover, 83  
  opts_selection, 84  
  opts_sizing, 85  
  opts_toolbar, 86  
  opts_tooltip, 87  
  opts_zoom, 89

* interactive scale  
  scale_alpha_interactive, 91  
  scale_colour_brewer_interactive, 92  
  scale_colour_interactive, 94  
  scale_colour_steps_interactive, 96  
  scale_gradient_interactive, 97  
  scale_linetype_interactive, 101  
  scale_manual_interactive, 102  
  scale_shape_interactive, 105  
  scale_size_interactive, 107  
  scale_viridis_interactive, 108

  aes(), 11, 14–19, 21–23, 25–31, 33, 35, 37–40, 43–45, 47–49, 70, 78
  aes_(), 78
  annotate(), 4  
  annotate_interactive, 4  
  annotation_raster(), 5  
  annotation_raster_interactive, 5  
  annotation_raster_interactive(), 5

  circleGrob(), 68  
  curveGrob(), 69

  Devices, 7  
  dsvg, 6, 8, 51, 53  
  dsvg(), 54, 112  
  dsvg_view, 7  
  element_interactive, 8  
  element_line(), 8  
  element_line_interactive  
    (element_interactive), 8  
  element_rect(), 8  
  element_rect_interactive  
    (element_interactive), 8  
  element_text(), 8  
  element_text_interactive, 81  
  element_text_interactive  
    (element_interactive), 8  
  element_text_interactive(), 8, 71, 78  
  font_family_exists, 10, 82, 112

  geom_abline(), 11  
  geom_abline_interactive, 11  
  geom_area(), 42  
  geom_area_interactive  
    (geom_ribbon_interactive), 42  
  geom_bar(), 13  
  geom_bar_interactive, 13  
  geom_bin2d(), 18  
  geom_bin2d_interactive  
    (geom_bin_2d_interactive), 15  
  geom_bin_2d(), 15  
  geom_bin_2d_interactive, 15  
  geom_boxplot(), 16  
  geom_boxplot_interactive, 16  
  geom_col(), 13  
  geom_col_interactive  
    (geom_bar_interactive), 13  
  geom_contour(), 17  
  geom_contour_filled(), 17  
  geom_contour_filled_interactive  
    (geom_contour_interactive), 17
geom_contour_interactive, 17
geom_count_interactive, 18
geom_crossbar(), 19
geom_crossbar_interactive, 19
geom_curve(), 20
geom_curve_interactive, 20
geom_density(), 23
geom_density2d_filled_interactive
  (geom_density_2d_interactive), 22
geom_density2d_interactive
  (geom_density_2d_interactive), 22
geom_density_2d(), 22
geom_density_2d_filled(), 22
geom_density_2d_filled_interactive
  (geom_density_2d_interactive), 22
geom_density_2d_interactive, 22
geom_density_interactive, 23
geom_dotplot(), 24
geom_dotplot_interactive, 24
geom_errorbar(), 19
geom_errorbar_interactive
  (geom_crossbar_interactive), 19
geom_errorbarh(), 25
geom_errorbarh_interactive, 25
geom_freqpoly(), 27
geom_freqpoly_interactive, 27
geom_hex(), 28
geom_hex_interactive, 28
geom_histogram(), 27
geom_histogram_interactive
  (geom_freqpoly_interactive), 27
geom_hline(), 11
geom_hline_interactive
  (geom_abline_interactive), 11
geom_jitter(), 29
geom_jitter_interactive, 29
geom_label(), 30
geom_label_interactive, 30
geom_label_interactive(), 70
geom_label_repel_interactive
  (geom_text_repel_interactive), 47
geom_line(), 33
geom_line_interactive
  (geom_path_interactive), 33
geom_linerange(), 19
geom_linerange_interactive
  (geom_crossbar_interactive), 19
geom_map(), 31
geom_map_interactive, 31
geom_path(), 33
geom_path_interactive, 33
geom_point(), 35
geom_point_interactive, 35
geom_pointrange(), 19
geom_pointrange_interactive
  (geom_crossbar_interactive), 19
geom_polygon(), 36
geom_polygon_interactive, 36
geom_quantile(), 38
geom_quantile_interactive, 38
geom_raster(), 39
geom_raster_interactive, 39
geom_rect(), 40
geom_rect_interactive, 40
geom.ribbon(), 42
geom.ribbon_interactive, 42
geom_segment(), 20
geom_segment_interactive
  (geom_curve_interactive), 20
geom.sf(), 43
geom_sf_interactive, 43
geom_sf_label(), 43
geom_sf_label_interactive
  (geom_sf_interactive), 43
geom_sf_text(), 43
geom_sf_text_interactive
  (geom_sf_interactive), 43
geom_smooth(), 45
geom_smooth_interactive, 45
geom.spoke(), 46
geom_spoke_interactive, 46
geom_step(), 33
geom_step_interactive
  (geom_path_interactive), 33
geom.text(), 30
geom_text_interactive
  (geom_label_interactive), 30
geom_text_interactive(), 70
geom_text_repel_interactive, 47
geom_tile(), 40
geom_tile_interactive
  (geom_rect_interactive), 40
geom_vline(), 49
geom_vline_interactive, 49
geom_vline(), 11
geom_vline_interactive
  (geom_abline_interactive), 11
ggiraph, 50, 90
ggiraphOutput, 52
ggrepel::geom_label_repel(), 47
ggrepel::geom_text_repel(), 47
girafe, 52

girafe(). 5, 8, 11, 14–19, 21, 22, 24–30, 32,
  33, 36–38, 40, 41, 43–45, 47–50, 56,
  58, 60, 63, 65, 69, 71–77, 87, 90, 92,
  94, 96, 97, 99, 102, 103, 106, 108,
  110, 112
girafe_css, 55, 70, 83–85
girafe_options, 50, 53, 56, 83, 85–89
girafe_options(), 54, 71
girafeOutput, 54
girafeOutput(). 90
guide_bins(), 57
guide_bins_interactive, 57
guide_bins_interactive(), 57, 60, 63, 64,
  71, 92, 93, 95, 97, 98, 101, 103, 106,
  108, 109
guide_colorbar_interactive
  (guide_colourbar_interactive), 59
guide_colorsteps_interactive
  (guide_coloursteps_interactive), 62
guide_colorbar(), 59
guide_colorbar_interactive, 59
guide_colorbar_interactive(), 57, 60,
  63, 65, 71, 92, 93, 95, 97, 98, 102,
  103, 106, 108, 109
guide_colorsteps(), 62
guide_colorsteps_interactive, 62
guide_colorsteps_interactive(), 57, 60,
  63, 65, 71, 92, 93, 95, 97, 98, 102,
  103, 106, 108, 109
guide_legend(), 64
guide_legend_interactive, 64
guide_legend_interactive(), 57, 60, 63,
  64, 71, 92, 93, 95, 97, 98, 101, 103,
  106, 108, 109

htmltools::HTML(), 70

interactive_circle_grob, 68
interactive_curve_grob, 69

interaction_lines_grob
  (interactive_polyline_grob), 74

interactive_parameters, 5, 69, 81
interactive_parameters(), 4, 5, 11, 13,
  15–20, 22, 23, 25–31, 33, 35, 36,
  38–40, 42, 44–46, 48, 49, 53, 58, 60,
  63, 65, 68, 69, 72–78, 81, 91, 93, 95,
  96, 98, 101, 103, 106, 107, 109

interactive_path_grob, 72
interactive_points_grob, 72
interactive_polygon_grob, 73
interactive_polyline_grob, 74
interactive_raster_grob, 74
interactive_rect_grob, 75
interactive_roundrect_grob, 76
interactive_segments_grob, 76
interactive_text_grob, 77

label_interactive, 81
label_interactive(), 8, 71, 78
labeller(), 78
labeller_interactive(), 78
labellers, 78
labs(), 8, 71, 81
linesGrob(), 74

match_family, 10, 82, 112
match_family(), 7, 111

opts_hover, 53, 56, 83, 85–89
opts_hover(), 70
opts_hover_inv (opts_hover), 83
opts_hover_key (opts_hover), 83
opts_hover_key(), 70
opts_hover_theme (opts_hover), 83
opts_hover_theme(), 70
opts_selection, 53, 56, 83, 84, 86–89
opts_selection(), 70
opts_selection_key (opts_selection), 84
opts_selection_key(), 70
opts_selection_theme (opts_selection), 84

opts_selection_theme(), 70

effects, 56, 83, 85, 87–89

effects_popup, 56, 83, 85, 86, 88, 89

effects_tooltip, 53, 56, 83, 85–87, 87, 89
scale_colour_interactive(), 94
scale_colour_brewer(), 92
scale_colour_brewer_interactive(), 92, 96, 97, 99, 102, 103, 106, 108, 110
scale_colour_continuous(), 94
scale_colour_continuous_interactive(scale_colour_interactive), 94
scale_colour_date(), 94
scale_colour_date_interactive(scale_colour_interactive), 94
scale_colour_datetime(), 94
scale_colour_datetime_interactive(scale_colour_interactive), 94
scale_colour_discrete(), 94
scale_colour_discrete_interactive(scale_colour_interactive), 94
scale_colour_distiller(), 92
scale_colour_distiller_interactive(scale_colour_brewer_interactive), 92
scale_colour_fermenter(), 92
scale_colour_fermenter_interactive(scale_colour_brewer_interactive), 92
scale_colour_gradient(), 97
scale_colour_gradient2(), 97
scale_colour_gradient2_interactive(scale_gradient_interactive), 97
scale_colour_gradient_interactive(scale_gradient_interactive), 97
scale_colour_gradientn(), 97
scale_colour_gradientn_interactive(scale_gradient_interactive), 97
scale_colour_grey(), 94
scale_colour_grey_interactive(scale_colour_interactive), 94
scale_colour_hue(), 94
scale_colour_hue_interactive(scale_colour_interactive), 94
scale_colour_interactive(), 92, 94, 96, 97, 99, 102, 103, 106, 108, 110
scale_colour_manual(), 102
scale_colour_manual_interactive(scale_manual_interactive), 102
scale_colour_ordinal(), 108
scale_colour_ordinal_interactive(scale_viridis_interactive), 108
scale_colour_steps(), 96
scale_colour_steps2(), 96
scale_colour_steps2_interactive(scale_colour_steps_interactive), 96
scale_colour_steps_interactive(), 92, 94, 96, 97, 99, 102, 103, 106, 108, 110
scale_colour_stepsn(), 96
scale_colour_stepsn_interactive(scale_colour_steps_interactive), 96
scale_colour_viridis_b(), 108
scale_colour_viridis_b_interactive(scale_viridis_interactive), 108
scale_colour_viridis_c(), 108
scale_colour_viridis_c_interactive(scale_viridis_interactive), 108
scale_colour_viridis_d(), 108
scale_colour_viridis_d_interactive(scale_viridis_interactive), 108
scale_discrete_manual(), 102
scale_discrete_manual_interactive(scale_manual_interactive), 102
scale_fill_binned(), 94
scale_fill_binned_interactive(scale_colour_interactive), 94
scale_fill_brewer(), 92
scale_fill_brewer_interactive(scale_viridis_interactive), 92
scale_fill_continuous(), 94
scale_fill_continuous_interactive(scale_colour_interactive), 94
scale_fill_date(), 94
scale_fill_date_interactive(scale_colour_interactive), 94
scale_fill_datetime(), 94
scale_fill_datetime_interactive(scale_colour_interactive), 94
scale_fill_discrete(), 94
scale_fill_discrete_interactive(scale_colour_interactive), 94
scale_fill_distiller(), 92
scale_fill_distiller_interactive
  (scale_colour_brewer_interactive), 92
scale_fill_fermenter(), 92
scale_fill_fermenter_interactive
  (scale_colour_brewer_interactive), 92
scale_fill_gradient(), 97
scale_fill_gradient2(), 97
scale_fill_gradient2_interactive
  (scale_colour_brewer_interactive), 97
scale_fill_gradient_interactive
  (scale_viridis_interactive), 108
scale_fill_gradientn(), 97
scale_fill_gradientn_interactive
  (scale_colour_brewer_interactive), 97
scale_fill_grey(), 94
scale_fill_grey_interactive
  (scale_colour_interactive), 94
scale_fill_hue(), 94
scale_fill_hue_interactive
  (scale_colour_interactive), 94
scale_fill_manual(), 102
scale_fill_manual_interactive
  (scale_manual_interactive), 102
scale_fill_ordinal(), 108
scale_fill_ordinal_interactive
  (scale_viridis_interactive), 108
scale_fill_steps(), 96
scale_fill_steps2(), 96
scale_fill_steps2_interactive
  (scale_colour_steps_interactive), 96
scale_fill_steps_interactive
  (scale_colour_steps_interactive), 96
scale_fill_stepsn(), 96
scale_fill_stepsn_interactive
  (scale_colour_steps_interactive), 96
scale_fill_viridis_b(), 108
scale_fill_viridis_b_interactive
  (scale_viridis_interactive), 108
scale_fill_viridis_c(), 108
scale_fill_viridis_c_interactive
  (scale_viridis_interactive), 108
scale_fill_viridis_d(), 108
scale_fill_viridis_d_interactive
  (scale_viridis_interactive), 108
scale_gradient_interactive, 92, 94, 96, 97, 102, 103, 106, 108, 110
scale_linetype(), 101
scale_linetype_binned(), 101
scale_linetype_binned_interactive
  (scale_linetype_interactive), 101
scale_linetype_continuous(), 101
scale_linetype_continuous_interactive
  (scale_linetype_interactive), 101
scale_linetype_discrete(), 101
scale_linetype_discrete_interactive
  (scale_linetype_interactive), 101
scale_linetype_manual(), 102
scale_linetype_manual_interactive
  (scale_manual_interactive), 102
scale_radius(), 107
scale_radius_interactive
  (scale_size_interactive), 107
scale_shape(), 105
scale_shape_binned(), 105
scale_shape_binned_interactive
  (scale_shape_interactive), 105
scale_shape_continuous(), 105
scale_shape_continuous_interactive
  (scale_shape_interactive), 105
scale_shape_discrete(), 105
scale_shape_discrete_interactive
  (scale_shape_interactive), 105
scale_shape_interactive, 92, 94, 96, 97, 99, 102, 103, 105, 108, 110
scale_shape_manual(), 102
scale_shape_manual_interactive
(scale_manual_interactive), 102
scale_shape_ordinal(), 105
scale_shape_ordinal_interactive
(scale_shape_interactive), 105
scale_size(), 107
scale_size_area(), 107
scale_size_area_interactive
(scale_size_interactive), 107
scale_size_binned(), 107
scale_size_binned_area(), 107
scale_size_binned_area_interactive
(scale_size_interactive), 107
scale_size_binned_interactive
(scale_size_interactive), 107
scale_size_continuous(), 107
scale_size_continuous_interactive
(scale_size_interactive), 107
scale_size_date(), 107
scale_size_date_interactive
(scale_size_interactive), 107
scale_size_datetime(), 107
scale_size_datetime_interactive
(scale_size_interactive), 107
scale_size_discrete(), 107
scale_size_discrete_interactive
(scale_size_interactive), 107
scale_size_interactive, 92, 94, 96, 97, 99, 102, 103, 106, 107, 110
scale_size_manual(), 102
scale_size_manual_interactive
(scale_manual_interactive), 102
scale_size_ordinal(), 107
scale_size_ordinal_interactive
(scale_size_interactive), 107
scale_viridis_interactive, 92, 94, 96, 97, 99, 102, 103, 106, 108, 108
segmentsGrob, 76
textGrob, 77
theme, 8
theme(), 78
validated_fonts, 10, 82, 111
validated_fonts(), 54