Package ‘ggiraph’

November 1, 2018

Type Package

Title Make ‘ggplot2’ Graphics Interactive

Description Create interactive ‘ggplot2’ graphics using ‘htmlwidgets’.

Version 0.6.0

License GPL-3

Copyright See file COPYRIGHTS.

Imports grid, ggplot2 (>= 3.0.0), htmlwidgets (>= 0.6), stats, xml2
>= 1.0.0), htmltools, Rcpp (>= 0.12.12), gdtools (>= 0.1.6),

LinkingTo Rcpp, gdtools

Suggests knitr, testthat, rmarkdown, maps, shiny, sf (>= 0.3-4), dplyr

VignetteBuilder knitr

URL https://davidgohel.github.io/ggiraph

BugReports https://github.com/davidgohel/ggiraph/issues

RoxygenNote 6.1.0

NeedsCompilation yes

Author David Gohel [aut, cre],
  Mike Bostock [cph] (d3.js),
  Speros Kokenes [cph] (d3-lasso),
  Eric Shull [cph] (saveSvgAsPng js library),
  Eric Book [ctb] (hline and vline geoms)

Maintainer David Gohel <david.gohel@ardata.fr>

Repository CRAN

Date/Publication 2018-11-01 21:50:03 UTC

R topics documented:

  drawDetails.interactive_path_grob .................................. 3
  drawDetails.interactive_points_grob .................................. 3
  drawDetails.interactive_polygon_grob ................................ 4
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>drawDetails.interactive_polyline_grob</td>
<td>4</td>
</tr>
<tr>
<td>drawDetails.interactive_rect_grob</td>
<td>5</td>
</tr>
<tr>
<td>drawDetails.interactive_segments_grob</td>
<td>5</td>
</tr>
<tr>
<td>drawDetails.interactive_text_grob</td>
<td>6</td>
</tr>
<tr>
<td>dsvg</td>
<td>6</td>
</tr>
<tr>
<td>dsvg_view</td>
<td>7</td>
</tr>
<tr>
<td>GeomInteractiveBoxplot</td>
<td>7</td>
</tr>
<tr>
<td>geom_bar_interactive</td>
<td>8</td>
</tr>
<tr>
<td>geom_boxplot_interactive</td>
<td>9</td>
</tr>
<tr>
<td>geom_histogram_interactive</td>
<td>10</td>
</tr>
<tr>
<td>geom_hline_interactive</td>
<td>11</td>
</tr>
<tr>
<td>geom_map_interactive</td>
<td>13</td>
</tr>
<tr>
<td>geom_path_interactive</td>
<td>14</td>
</tr>
<tr>
<td>geom_point_interactive</td>
<td>16</td>
</tr>
<tr>
<td>geom_polygon_interactive</td>
<td>17</td>
</tr>
<tr>
<td>geom_rect_interactive</td>
<td>19</td>
</tr>
<tr>
<td>geom_segment_interactive</td>
<td>21</td>
</tr>
<tr>
<td>geom_sf_interactive</td>
<td>22</td>
</tr>
<tr>
<td>geom_text_interactive</td>
<td>23</td>
</tr>
<tr>
<td>geom_vline_interactive</td>
<td>25</td>
</tr>
<tr>
<td>ggiraph</td>
<td>26</td>
</tr>
<tr>
<td>ggiraphOutput</td>
<td>27</td>
</tr>
<tr>
<td>girafe</td>
<td>28</td>
</tr>
<tr>
<td>girafeOutput</td>
<td>30</td>
</tr>
<tr>
<td>girafe_options</td>
<td>31</td>
</tr>
<tr>
<td>interactive_path_grob</td>
<td>32</td>
</tr>
<tr>
<td>interactive_points_grob</td>
<td>32</td>
</tr>
<tr>
<td>interactive_polygon_grob</td>
<td>33</td>
</tr>
<tr>
<td>interactive_polyline_grob</td>
<td>34</td>
</tr>
<tr>
<td>interactive_rect_grob</td>
<td>35</td>
</tr>
<tr>
<td>interactive_segments_grob</td>
<td>36</td>
</tr>
<tr>
<td>interactive_text_grob</td>
<td>37</td>
</tr>
<tr>
<td>opts_hover</td>
<td>38</td>
</tr>
<tr>
<td>opts_selection</td>
<td>38</td>
</tr>
<tr>
<td>opts_toolbar</td>
<td>39</td>
</tr>
<tr>
<td>opts_tooltip</td>
<td>40</td>
</tr>
<tr>
<td>opts_zoom</td>
<td>42</td>
</tr>
<tr>
<td>renderggiraph</td>
<td>43</td>
</tr>
<tr>
<td>renderGirafe</td>
<td>43</td>
</tr>
</tbody>
</table>
**Description**

draw an interactive_path_grob

**Usage**

```r
## S3 method for class 'interactive_path_grob'
drawDetails(x, recording)
```

**Arguments**

- `x`: A grid grob.
- `recording`: A logical value indicating whether a grob is being added to the display list or redrawn from the display list.

**Description**

draw an interactive_points_grob

**Usage**

```r
## S3 method for class 'interactive_points_grob'
drawDetails(x, recording)
```

**Arguments**

- `x`: A grid grob.
- `recording`: A logical value indicating whether a grob is being added to the display list or redrawn from the display list.
**Description**

draw an interactive_polygon_grob

**Usage**

```r
## S3 method for class 'interactive_polygon_grob'
drawDetails(x, recording)
```

**Arguments**

- `x`: A grid grob.
- `recording`: A logical value indicating whether a grob is being added to the display list or redrawn from the display list.
### drawDetails.interactive_rect_grob

#### interactive_rect_grob drawing

**Description**

draw an interactive_rect_grob

**Usage**

```r
## S3 method for class 'interactive_rect_grob'
drawDetails(x, recording)
```

**Arguments**

- `x` A grid grob.
- `recording` A logical value indicating whether a grob is being added to the display list or redrawn from the display list.

### drawDetails.interactive_segments_grob

#### interactive_segments_grob drawing

**Description**

draw an interactive_segments_grob

**Usage**

```r
## S3 method for class 'interactive_segments_grob'
drawDetails(x, recording)
```

**Arguments**

- `x` A grid grob.
- `recording` A logical value indicating whether a grob is being added to the display list or redrawn from the display list.
**drawDetails.interactive_text_grob**

*interactive_text_grob drawing*

**Description**

draw an interactive_text_grob

**Usage**

```r
## S3 method for class 'interactive_text_grob'
drawDetails(x, recording)
```

**Arguments**

- **x**
  - A grid grob.
- **recording**
  - A logical value indicating whether a grob is being added to the display list or redrawn from the display list.

**dsvg**

*SVG Graphics Driver*

**Description**

This function produces SVG files (compliant to the current w3 svg XML standard) where elements can be made interactive.

**Usage**

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

**Arguments**

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

Examples

```r
library(ggiraph)
dsvg_view()
plot(rnorm(10), main="Simple Example", xlab = ",""; ylab = ""
dev.off()
```

---

**GeomInteractiveBoxplot**

*ggproto classes for ggiraph*

---

**Description**

- ggproto classes for ggiraph

**Geoms**

All `geom_*_interactive` functions (like `geom_point_interactive`) return a layer that contains a `GeomInteractive*` object (like `GeomInteractivePoint`). The `Geom*` object is responsible for rendering the data in the plot.

See **Geom** for more information.
Description

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

Usage

```r
geom_bar_interactive(mapping = NULL, data = NULL, stat = "count",
                      position = "stack", ..., width = NULL, na.rm = FALSE,
                      show.legend = NA, inherit.aes = TRUE)
```

Arguments

- `mapping` The aesthetic mapping, see `geom_point`.
- `data` A data frame, see `geom_point`.
- `stat` The statistical transformation to use on the data for this layer, as a string, see `geom_point`.
- `position` Position adjustment, see `geom_point`.
- `...` other arguments passed on to layer. See `geom_point`.
- `width` Bar width.
- `na.rm` See `geom_point`.
- `show.legend` See `geom_point`.
- `inherit.aes` See `geom_point`.

See Also

`ggiraph`

Examples

```r
library(ggplot2)
g <- ggplot(mpg, aes(x = class, tooltip = class,
            data_id = class )) +
geom_bar_interactive()
ggiraph(code = print(g))

dat <- data.frame( name = c("David", "Constance", "Leonie" ),
            gender = c( "Male", "Female", "Female" ),
            height = c(172, 159, 71 ) )
g <- ggplot(dat, aes( x = name, y = height, tooltip = gender,
            data_id = name )) +
geom_bar_interactive(stat = "identity")
ggiraph(code = print(g))
```
**geom_boxplot_interactive**

**interactive boxplot**

**Description**

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

**Usage**

```r
geom_boxplot_interactive(mapping = NULL, data = NULL,
stat = "boxplot", position = "dodge", ..., outlier.colour = NULL,
outlier.color = NULL, outlier.shape = 19, outlier.size = 1.5,
outlier.stroke = 0.5, notch = FALSE, notchwidth = 0.5,
varwidth = FALSE, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)
```

**Arguments**

- `mapping` The aesthetic mapping, see `geom_point`.
- `data` A data frame, see `geom_point`.
- `stat` see `geom_boxplot`.
- `position` Postion adjustment, see `geom_point`.
- `...` other arguments passed on to layer. See `geom_point`.
- `outlier.colour` see `geom_boxplot`.
- `outlier.color` see `geom_boxplot`.
- `outlier.shape` see `geom_boxplot`.
- `outlier.size` see `geom_boxplot`.
- `outlier.stroke` see `geom_boxplot`.
- `notch` see `geom_boxplot`.
- `notchwidth` see `geom_boxplot`.
- `varwidth` see `geom_boxplot`.
- `na.rm` See `geom_point`.
- `show.legend` See `geom_point`.
- `inherit.aes` See `geom_point`.

**See Also**

`ggiraph`
Examples

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

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

ggiraph(code = print(p))

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

girafe(ggobj = p)
```

Description

The geometry is based on `geom_histogram`. 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_histogram_interactive(mapping = NULL, data = NULL, stat = "bin",
   position = "stack", ..., binwidth = NULL, bins = NULL,
   na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

Arguments

- **mapping**: Set of aesthetic mappings created by `aes()` or `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
- **data**: The data to be displayed in this layer. There are three options:
  - If NULL, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data.


**stat**
Use to override the default connection between `geom_histogram()/geom_freqpoly()` and `stat_bin()`.

**position**
Position adjustment, either as a string, or the result of a call to a position adjustment function.

**...**
Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

**binwidth**
The width of the bins. Can be specified as a numeric value, or a function that calculates width from x. The default is to use bins bins that cover the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data.

The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.

**bins**
Number of bins. Overridden by `binwidth`. Defaults to 30.

**na.rm**
If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

**show.legend**
logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

**inherit.aes**
If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

**See Also**

`ggiraph`

---

**geom_hline_interactive**

*Horizontal interactive reference line*

**Description**

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

**Usage**

```
geom_hline_interactive(mapping = NULL, data = NULL, ..., yintercept, na.rm = FALSE, show.legend = NA)
```
Arguments

mapping The aesthetic mapping, see `geom_point`.
data A data frame, see `geom_point`.
... other arguments passed on to layer. See `geom_point`.
yintercept controls the position of the line
na.rm See `geom_point`.
show.legend See `geom_point`.

See Also

`ggiraph`

Examples

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

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)))
  girafe(ggobj = gg_hline1)
}

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), ",""\\n""), 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))

girafe(ggobj = gg_hline2)
```
**geom_map_interactive**

**interactive polygons from a reference map.**

**Description**

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

**Usage**

```r
geom_map_interactive(mapping = NULL, data = NULL, map, stat = "identity", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```

**Arguments**

- `mapping` The aesthetic mapping, see `geom_point`.
- `data` A data frame, see `geom_point`.
- `map` Data frame that contains the map coordinates. See `geom_map`.
- `stat` The statistical transformation to use on the data for this layer, as a string, see `geom_point`.
- `na.rm` See `geom_point`.
- `show.legend` See `geom_point`.
- `inherit.aes` See `geom_point`.
- `...` other arguments passed on to layer. See `geom_point`.

**See Also**

`ggiraph`

**Examples**

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

crimes <- data.frame(state = tolower(rownames(USArests)), USArests)

# create tooltips and onclick events
states_ <- sprintf("<p>%s</p>",
                   as.character(crimes$state))
table_  <- paste0(  
    "<table><tr><td>UrbanPop</td></tr>
    "<tr><td>%0f</td></tr>,
    crimes$UrbanPop),
    "</tr><tr>
    "<td>Assault</td>,
    crimes$Assault),
    "</tr></table>"
```
geom_path_interactive

Description

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

Usage

```r
geom_path_interactive(mapping = NULL, data = NULL, stat = "identity", position = "identity", lineend = "butt", linejoin = "round", linemitre = 1, na.rm = FALSE, arrow = NULL, show.legend = NA, inherit.aes = TRUE, ...)
```

```r
geom_line_interactive(mapping = NULL, data = NULL, stat = "identity", position = "identity", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```
Arguments

- `mapping`: The aesthetic mapping, see `geom_point`.
- `data`: A data frame, see `geom_point`.
- `stat`: The statistical transformation to use on the data for this layer, as a string, see `geom_point`.
- `position`: Position adjustment, see `geom_point`.
- `lineend`: Line end style (round, butt, square)
- `linejoin`: Line join style (round, mitre, bevel)
- `linemitre`: Line mitre limit (number greater than 1)
- `na.rm`: See `geom_point`.
- `arrow`: Arrow specification, as created by `arrow`.
- `show.legend`: See `geom_point`.
- `inherit.aes`: See `geom_point`.
- `...`: Other arguments passed on to layer. See `geom_point`.

See Also

- `ggiraph`

Examples

```r
# add interactive paths to a ggplot
library(ggplot2)
# geom_line_interactive example
if(requireNamespace("dplyr", quietly = TRUE)){
  gg <- ggplot(economics_long,
    aes(date, value, colour = variable, tooltip = variable, data_id = variable)) +
    geom_line_interactive(size = .75)
  ggiraph(code = {print(gg)}, hover_css = "stroke:red;")
}

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

cols = c("orange", "orange1", "orange2", "navajowhite4", "navy")
dataset2 <- data.frame(x = rep(1:20, 5),
y = rnorm(100, 5, .2) + rep(1:5, each=20),
...)```
geom_point_interactive

**interactive points**

**Description**

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

**Usage**

```
geom_point_interactive(mapping = NULL, data = NULL, 
  stat = "identity", position = "identity", na.rm = FALSE, 
  show.legend = NA, inherit.aes = TRUE, ...)
```

**Arguments**

- `mapping` The aesthetic mapping, see `geom_point`.
- `data` A data frame, see `geom_point`. 

```r
z = rep(1:20, 5),
grp = factor(rep(1:5, each=20)),
color = factor(rep(1:20, each=20)),
label = rep(paste0("id ", 1:5 ), each=20),
onclick = paste0("alert("", 
sample(letters, 100, replace = TRUE), 
"\") ")
```

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

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, 
color = color, tooltip = label, onclick = onclick), size = 1 )

# ggiraph widgets ---
ggiraph(code = (print(gg_path_1)), hover_css = "stroke-width:3px;")
ggiraph(code = (print(gg_path_2)), hover_css = "stroke:orange;stroke-width:3px;")
ggiraph(code = (print(gg_path_3)), hover_css = "stroke-width:10px;")
```
The geometry is based on `geom_polygon`. See the documentation for those functions for more details.
Usage

```r
gem_polygon_interactive(mapping = NULL, data = NULL,
  stat = "identity", position = "identity", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, ...)
```

Arguments

- **mapping**: The aesthetic mapping, see `geom_point`.
- **data**: A data frame, see `geom_point`.
- **stat**: The statistical transformation to use on the data for this layer, as a string, see `geom_point`.
- **position**: Position adjustment, see `geom_point`.
- **na.rm**: See `geom_point`.
- **show.legend**: See `geom_point`.
- **inherit.aes**: See `geom_point`.
- **...**: other arguments passed on to layer. See `geom_point`.

See Also

- `ggiraph`

Examples

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

# 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

gg_poly_1 <- ggplot(datapoly, aes( x = x, y = y ) ) +
  geom_polygon_interactive(aes(fill = value, group = id,
    tooltip = value, data_id = value, onclick = oc))

# display
```
geom_rect_interactive

```r
ggiraph(code = (print(gg_poly_1)))
```

---

**geom_rect_interactive**  *interactive rectangles*

**Description**

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

**Usage**

```r
geom_rect_interactive(mapping = NULL, data = NULL, stat = "identity",
position = "identity", na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE, ...)
```

```r
geom_tile_interactive(mapping = NULL, data = NULL, stat = "identity",
position = "identity", ..., na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)
```

**Arguments**

- `mapping`  
  The aesthetic mapping, see `geom_point`.
- `data`  
  A data frame, see `geom_point`.
- `stat`  
  The statistical transformation to use on the data for this layer, as a string, see `geom_point`.
- `position`  
  Position adjustment, see `geom_point`.
- `na.rm`  
  See `geom_point`.
- `show.legend`  
  See `geom_point`.
- `inherit.aes`  
  See `geom_point`.
- `...`  
  other arguments passed on to layer. See `geom_point`.

**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**

`ggiraph`
Examples

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

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 )

ggiiraph(code = {print(gg_rect)})
library(ggplot2)
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) )

ggiiraph( code = {
  print(
    ggplot(df, aes(x, y, tooltip = id)) + geom_tile_interactive(aes(fill = z))
  )
})

# 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("%s" vs %s: 
  var1, var2, cor)

# ggplot creation and ggiraph printing ----
p <- ggplot(data = cor_mat, aes(x = var1, y = var2 ) ) +
  geom_tile_interactive(aes(fill = cor, tooltip = tooltip), colour = "white") +
  scale_fill_gradient(low = "#BC120A", mid = "white", high = "#BC120A", limits = c(-1, 1)) +
  coord_equal()
ggiraph( code = print(p))

---

**geom_segment_interactive**

*Line interactive segments*

**Description**

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

**Usage**

```r
geom_segment_interactive(mapping = NULL, data = NULL, 
stat = "identity", position = "identity", arrow = NULL, 
lineend = "butt", na.rm = FALSE, show.legend = NA, 
inherit.aes = TRUE, ...)
```

**Arguments**

- `mapping` The aesthetic mapping, see `geom_point`.
- `data` A data frame, see `geom_point`.
- `stat` The statistical transformation to use on the data for this layer, as a string, see `geom_point`.
- `position` Position adjustment, see `geom_point`.
- `arrow` Arrow specification, as created by `grid::arrow`.
- `lineend` Line end style (round, butt, square)
- `na.rm` See `geom_point`.
- `show.legend` See `geom_point`.
- `inherit.aes` See `geom_point`.
- `...` other arguments passed on to layer. See `geom_point`.

**See Also**

- `ggiraph`
Examples

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

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))

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),
.yy=c(0.2,1.3,1.7,0.8,1.4),
labs = paste0("Lab", 1:5))
dataset$clickjs = paste0("alert\"", dataset$lab, "\")


gg_segment_2 = ggplot() +
geom_segment_interactive(data=dataset, mapping=aes(x=x, y=y,
exend=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")

ggiraph(code = {print(gg_segment_1)})
ggiraph(code = {print(gg_segment_2)})

description

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

Usage

gg_segment_interactive(mapping = aes(), data = NULL, stat = "sf",
position = "identity", na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE, ...)

Arguments

mapping The aesthetic mapping, see geom_point.
data A data frame, see geom_point.
stat The statistical transformation to use on the data for this layer, as a string, see geom_point.
geom_text_interactive

Position adjustment, see geom_point.

See Also
ggiraph

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)) {
  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))
ggiraph( ggobj = gg)

  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 ggplot 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")
girafe( ggobj = gg)
}

Description

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

geom_text_interactive(mapping = NULL, data = NULL, stat = "identity",
position = "identity", parse = FALSE, ..., nudge_x = 0,
nudge_y = 0, check_overlap = FALSE, na.rm = FALSE,
show.legend = NA, inherit.aes = TRUE)

Arguments

mapping The aesthetic mapping, see geom_point.
data A data frame, see geom_point.
stat The statistical transformation to use on the data for this layer, as a string, see geom_point.
position Position adjustment, see geom_point.
parse See geom_point.
... other arguments passed on to layer. See geom_point.
nudge_x, nudge_y See geom_point.
check_overlap See geom_point.
a.rm See geom_point.
show.legend See geom_point.
inherit.aes See geom_point.

See Also

ggiraph

Examples

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

## 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 ) ) +
gem_text_interactive() +
coord_cartesian(xlim = c(0,50))
## geom_vline_interactive

### Vertical interactive reference line

**Description**

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

**Usage**

```r
geom_vline_interactive(mapping = NULL, data = NULL, ..., xintercept, na.rm = FALSE, show.legend = NA)
```

**Arguments**

- `mapping`: The aesthetic mapping, see `geom_point`.
- `data`: A data frame, see `geom_point`.
- `...`: other arguments passed on to layer. See `geom_point`.
- `xintercept`: controls the position of the line
- `na.rm`: See `geom_point`.
- `show.legend`: See `geom_point`.

**See Also**

`ggiraph`

**Examples**

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

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))
  ggiraph(code = print(gg_vline1))
}

dataset <- data.frame(x = rnorm(100))
```
```r
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")

ggiraph(code = print(gg_vline2),
  hover_css = "cursor:pointer;fill:orange;stroke:orange;")
```

---

### 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**

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

**Arguments**

- `code`  
  Plotting code to execute

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

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

- `width`  
  Widget width ratio (0 < width <= 1). See below in section.

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

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

- `tooltip_extra_css`  
  Extra css (added to position: absolute;pointer-events: none;) used to customize tooltip area.
### ggiraphOutput

**Create a ggiraph output element**

**Description**

Render a ggiraph within an application page.

- **hover_css**: CSS to apply when mouse is hover and element with a data-id attribute.
- **tooltip_opacity**: Tooltip opacity
- **tooltip_offx**: Tooltip x offset
- **tooltip_offy**: Tooltip y offset
- **tooltip_zindex**: Tooltip CSS z-index, default to 999.
- **zoom_max**: Maximum zoom factor
- **selection_type**: Row selection mode ("single", "multiple", "none") when widget is in a Shiny application.
- **selected_css**: CSS to apply when element is selected (shiny only).
- **dep_dir**: Deprecated; the path where the output files are stored. If NULL, the current path for temporary files is used.
- **xml_reader_options**: Read XML additional arguments to be used when parsing the SVG result. This feature can be used to parse huge SVG files by using `read_xml(options = "HUGE")` but this is not recommended.

**Examples**

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

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,
mapping = aes(x = wt, y = qsec, color = disp,
               tooltip = carname, data_id = carname) ) +
geom_point_interactive() + theme_minimal()
girafe(ggobj = gg_point, width = .7)
```
Usage

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

Arguments

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

Examples

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

## End(Not run)

girafe create a girafe object

Description

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

Usage

girafe(code, ggobj = NULL, width = 0.9, pointsize = 12,
width_svg = 6, height_svg = 5, xml_reader_options = list(), ...)

Arguments

code Plotting code to execute
ggobj ggplot objet to print. argument code will be ignored if this argument is supplied.
width widget width ratio (0 < width <= 1). See below in section .
pointsize the default pointsize of plotted text in pixels, default to 12.
width_svg, height_svg The width and height of the graphics region in inches. The default values are 6
and 5 inches. This will define the aspect ratio of the graphic as it will be used to
define viewbox attribute of the SVG result.
xml_reader_options

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

... arguments passed on to dsvg

Details

Use geom_zzz_interactive to create interactive graphical elements.

Difference from original functions is that the following aesthetics are understood: tooltip, onclick and data_id.

Tooltips can be displayed when mouse is over graphical elements.

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

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

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

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

Widget options

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

Widget sizing

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

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

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

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

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

Examples

```r
library(ggplot2)

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

gg_point = ggplot( data = dataset,
                   mapping = aes(x = wt, y = qsec, color = disp,
                                  tooltip = carname, data_id = carname ) ) +
geo_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.
- `width`  
  widget width
- `height`  
  widget height
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

opts_tooltip, opts_hover, opts_selection, opts_zoom, opts_toolbar, sizingPolicy

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()

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)
}
interactive_path_grob  Generate interactive grob paths

Description
This function can be used to generate interactive grob paths.

Usage

interactive_path_grob(x, y, id = NULL, id.lengths = NULL,
rule = "winding", tooltip = NULL, onclick = NULL, data_id = NULL,
default.units = "npc", name = NULL, gp = gpar(), vp = NULL)

Arguments

x  A numeric vector or unit object specifying x-locations.
y  A numeric vector or unit object specifying y-locations.
id  A numeric vector used to separate locations in x and y into sub-paths. All locations with the same id belong to the same sub-path.
id.lengths  A numeric vector used to separate locations in x and y into sub-paths. Specifies consecutive blocks of locations which make up separate sub-paths.
rule  A character value specifying the fill rule: either "winding" or "evenodd".
tooltip  tooltip associated with polygons
onclick  javascript action to execute when polygon is clicked
data_id  identifiers to associate with polygons
default.units  A string indicating the default units to use if x or y are only given as numeric vectors.
name  A character identifier.
gp  An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
vp  A Grid viewport object (or NULL).

interactive_points_grob

Generate interactive grob points

Description
This function can be used to generate interactive grob points.
interactive_polygon_grob

Usage

interactive_polygon_grob(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
tooltip = NULL, onclick = NULL, data_id = NULL, pch = 1,
size = unit(1, "char"), default.units = "native", name = NULL,
gp = gpar(), vp = NULL)

Arguments

x       numeric vector or unit object specifying x-values.
y       numeric vector or unit object specifying y-values.
tooltip tooltip associated with points
onclick javascript action to execute when point is clicked
data_id identifiers to associate with points
pch      numeric or character vector indicating what sort of plotting symbol to use. See points for the interpretation of these values, and note fill below.
size     unit object specifying the size of the plotting symbols.
default.units string indicating the default units to use if x or y are only given as numeric vectors.
name     character identifier.
gp       an R object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings; note that fill (and not bg as in package graphics points) is used to “fill”, i.e., color the background of symbols with pch = 21:25.
vp       A Grid viewport object (or NULL).

interactive_polygon_grob

Generate interactive grob polygons

Description

This function can be used to generate interactive grob polygons.

Usage

interactive_polygon_grob(x = unit(c(0, 1), "npc"), y = unit(c(0, 1),
"npc"), id = NULL, id.lengths = NULL, tooltip = NULL,
onclick = NULL, data_id = NULL, default.units = "npc",
name = NULL, gp = gpar(), vp = NULL)
**interactive_polyline_grob**

*Generate an Interactive Grob Path*

**Arguments**

- **x**  
  A numeric vector or unit object specifying x-locations.

- **y**  
  A numeric vector or unit object specifying y-locations.

- **id**  
  A numeric vector used to separate locations in x and y into multiple lines. All locations with the same id belong to the same line.

- **id.lengths**  
  A numeric vector used to separate locations in x and y into multiple lines. Specifies consecutive blocks of locations which make up separate lines.

- **tooltip**  
  Tooltip associated with polylines

- **onclick**  
  Javascript action to execute when polyline is clicked

- **data_id**  
  Identifiers to associate with polylines

- **default.units**  
  A string indicating the default units to use if x, y, width, or height are only given as numeric vectors.

- **name**  
  A character identifier.

- **gp**  
  An object of class `gpar`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.

- **vp**  
  A Grid viewport object (or NULL).

**Description**

This function can be used to generate an interactive grob path.

**Usage**

```r
interactive_polyline_grob(x = unit(c(0, 1), "npc"), y = unit(c(0, 1), "npc"), id = NULL, id.lengths = NULL, tooltip = NULL, onclick = NULL, data_id = NULL, default.units = "npc", arrow = NULL, name = NULL, gp = gpar(), vp = NULL)
```

**Arguments**

- **x**  
  A numeric vector or unit object specifying x-values.

- **y**  
  A numeric vector or unit object specifying y-values.

- **id**  
  A numeric vector used to separate locations in x and y into multiple lines. All locations with the same id belong to the same line.

- **id.lengths**  
  A numeric vector used to separate locations in x and y into multiple lines. Specifies consecutive blocks of locations which make up separate lines.

- **tooltip**  
  Tooltip associated with polylines

- **onclick**  
  Javascript action to execute when polyline is clicked
interactive_rect_grob

**Description**

This function can be used to generate interactive grob rectangles.

**Usage**

```r
interactive_rect_grob(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
width = unit(1, "npc"), height = unit(1, "npc"), tooltip = NULL,
onclick = NULL, data_id = NULL, just = "centre", hjust = NULL,
vjust = NULL, default.units = "npc", name = NULL, gp = gpar(),
vp = NULL)
```

**Arguments**

- `x` A numeric vector or unit object specifying x-location.
- `y` A numeric vector or unit object specifying y-location.
- `width` A numeric vector or unit object specifying width.
- `height` A numeric vector or unit object specifying height.
- `tooltip` tooltip associated with rectangles
- `onclick` javascript action to execute when rectangle is clicked
- `data_id` identifiers to associate with rectangles
- `just` The justification of the rectangle relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left alignment and 1 means right alignment.
- `hjust` A numeric vector specifying horizontal justification. If specified, overrides the just setting.
- `vjust` A numeric vector specifying vertical justification. If specified, overrides the just setting.
interactive_segments_grob

Generate interactive grob segments

Description

This function can be used to generate interactive grob segments.

Usage

interactive_segments_grob(x0 = unit(0, "npc"), y0 = unit(0, "npc"),
                          x1 = unit(1, "npc"), y1 = unit(1, "npc"), tooltip = NULL,
                          onclick = NULL, data_id = NULL, default.units = "npc",
                          arrow = NULL, name = NULL, gp = gpar(), vp = NULL)

Arguments

- x0: Numeric indicating the starting x-values of the line segments.
- y0: Numeric indicating the starting y-values of the line segments.
- x1: Numeric indicating the stopping x-values of the line segments.
- y1: Numeric indicating the stopping y-values of the line segments.
- tooltip: Tooltip associated with segments
- onclick: Javascript action to execute when segment is clicked
- data_id: Identifiers to associate with segments
- default.units: A string.
- arrow: A list describing arrow heads to place at either end of the line segments, as produced by the arrow function.
- name: A character identifier.
- gp: An object of class gpar.
- vp: A Grid viewport object (or NULL).
**interactive_text_grob**  Generate interactive grob text

**Description**
This function can be used to generate interactive grob text.

**Usage**
```r
interactive_text_grob(label, x = unit(0.5, "npc"), y = unit(0.5, "npc"), tooltip = NULL, onclick = NULL, data_id = NULL,
  just = "centre", hjust = NULL, vjust = NULL, rot = 0,
  check.overlap = FALSE, default.units = "npc", name = NULL,
  gp = gpar(), vp = NULL)
```

**Arguments**
- **label**: A character or expression vector. Other objects are coerced by `as.graphicsAnnot`.
- **x**: A numeric vector or unit object specifying x-values.
- **y**: A numeric vector or unit object specifying y-values.
- **tooltip**: Tooltip associated with rectangles.
- **onclick**: Javascript action to execute when rectangle is clicked.
- **data_id**: Identifiers to associate with rectangles.
- **just**: The justification of the text relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left (bottom) alignment and 1 means right (top) alignment.
- **hjust**: A numeric vector specifying horizontal justification. If specified, overrides the just setting.
- **vjust**: A numeric vector specifying vertical justification. If specified, overrides the just setting.
- **rot**: The angle to rotate the text.
- **check.overlap**: A logical value to indicate whether to check for and omit overlapping text.
- **default.units**: A string indicating the default units to use if x or y are only given as numeric vectors.
- **name**: A character identifier.
- **gp**: An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
- **vp**: A Grid viewport object (or NULL).
opts_hover  

**hover effect settings**

**Description**

Allows customization of the animation of graphic elements on which the mouse is positioned.

**Usage**

```r
opts_hover(css = NULL)
```

**Arguments**

- `css`: css to associate with elements to be animated when mouse is hover them.

**See Also**

- set options with `girafe_options`
- Other girafe animation options: `opts_selection, opts_toolbar, opts_tooltip, opts_zoom`

**Examples**

```r
library(ggplot2)

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

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

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

---

opts_selection  

**selection effect settings**

**Description**

Allows customization of the rendering of selected graphic elements.
Usage

```r
opts_selection(css = NULL, type = "multiple", only_shiny = TRUE)
```

Arguments

css
css to associate with elements when they are selected.

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

only_shiny
disable selections if not in a shiny context.

See Also

set options with `girafe_options`

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

Examples

```r
library(ggplot2)

dataset <- mtcars
dataset$carnames <- 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)
```
**Arguments**

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

**Note**

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

**See Also**

- set options with `girafe_options`
- Other girafe animation options: `opts_hover`, `opts_selection`, `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)
```

**Description**

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

**Usage**

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

**Arguments**

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

- **offx, offy**
  
  Tooltip x and y offset

- **use_cursor_pos**
  
  Should the cursor position be used to position tooltip (in addition to offx and offy). Setting to TRUE will have no effect in the RStudio browser windows.

- **opacity**
  
  Tooltip background opacity

- **use_fill, use_stroke**
  
  Logical, use fill and stroke properties to color tooltip.

- **delay_mouseover**
  
  The duration in milliseconds of the transition associated with tooltip display.

- **delay_mouseout**
  
  The duration in milliseconds of the transition associated with tooltip end of display.

- **zindex**
  
  Tooltip css z-index, default to 999.

**See Also**

- Set options with `girafe_options`

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

**Examples**

```r
library(ggplot2)

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

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

Allows customization of the zoom.

Usage

opts_zoom(min = 1, max = 1)

Arguments

- min: minimum zoom factor
- max: maximum zoom factor

See Also

set options with \texttt{girafe_options}

Other girafe animation options: \texttt{opts_hover, opts_selection, opts_toolbar, opts_tooltip}

Examples

```r
library(ggplot2)

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

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

# Use girafe function to create interactive object
x <- girafe(ggobj = gg)
x <- girafe_options(x,
  opts_zoom(min = .7, max = 2))
if( interactive() ) print(x)
```
renderggiraph

**Reactive version of ggiraph object**

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

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

renderGirafe

**Reactive version of girafe**

**Description**

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

**Usage**

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

**Arguments**

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

*Topic **datasets**
  GeomInteractiveBoxplot, 7

*Topic **device**
  dsvg, 6

aes(), 10
aes_(), 10
arrow, 15
as.graphicsAnnot, 37
borders(), 11

Devices, 7
drawDetails.interactive_path_grob, 3
drawDetails.interactive_points_grob, 3
drawDetails.interactive_polygon_grob, 4
drawDetails.interactive_polyline_grob, 4
drawDetails.interactive_rect_grob, 5
drawDetails.interactive_segments_grob, 5
drawDetails.interactive_text_grob, 6
dsvg, 6, 7, 27, 29
dsvg_view, 7
expression, 37
fortify(), 10

Geom, 7
geom_bar, 8
geom_bar_interactive, 8
geom_boxplot, 9
geom_boxplot_interactive, 9
geom_histogram, 10
geom_histogram_interactive, 10
geom_hline, 11
geom_hline_interactive, 11
geom_line, 14
geom_line_interactive
  (geom_path_interactive), 14
geom_map, 13
geom_map_interactive, 13
geom_path, 14
geom_path_interactive, 14
gem_point, 8, 9, 12, 13, 15–19, 21–25
gem_point_interactive, 16
gem_polygon, 17
gem_polygon_interactive, 17
gem_rect, 19
gem_rect_interactive, 19
gem_segment, 21
gem_segment_interactive, 21
gem_sf_interactive, 22
gem_text, 23
gem_text_interactive, 23
gem_tile, 19
gem_tile_interactive
  (geom_rect_interactive), 19
gem_vline, 25
gem_vline_interactive, 25
GeomInteractive
  (GeomInteractiveBoxplot), 7
GeomInteractiveBar
  (GeomInteractiveBoxplot), 7
GeomInteractiveBoxplot, 7
GeomInteractiveHLine
  (GeomInteractiveBoxplot), 7
GeomInteractiveLine
  (GeomInteractiveBoxplot), 7
GeomInteractiveMap
  (GeomInteractiveBoxplot), 7
GeomInteractivePoint
  (GeomInteractiveBoxplot), 7
GeomInteractivePolygon
  (GeomInteractiveBoxplot), 7
GeomInteractiveRect
  (GeomInteractiveBoxplot), 7
INDEX

GeomInteractiveSegment
   (GeomInteractiveBoxplot), 7
GeomInteractiveText
   (GeomInteractiveBoxplot), 7
GeomInteractiveTile
   (GeomInteractiveBoxplot), 7
GeomInteractiveVline
   (GeomInteractiveBoxplot), 7
ggiraph, 8, 9, 11–13, 15, 17–19, 21, 23–25, 26, 43
ggiraphOutput, 27
ggplot(), 10
ggsf, 22
girafe, 26, 28, 31, 40, 43
girafe_options, 26, 29, 30, 31, 38–42
girafeOutput, 30

interactive_path_grob, 32
interactive_points_grob, 32
interactive_polygon_grob, 33
interactive_polyline_grob, 34
interactive_rect_grob, 35
interactive_segments_grob, 36
interactive_text_grob, 37

layer(), 11

match_family, 6

opts_hover, 31, 38, 39–42
opts_selection, 31, 38, 38, 40–42
opts_toolbar, 31, 38, 39, 41, 42
opts_tooltip, 31, 38–40, 40, 42
opts_zoom, 31, 38–41, 42

points, 33

renderggiraph, 43
renderGirafe, 43

sizingPolicy, 31