Package ‘leaflet’

February 21, 2017

Type Package
Title Create Interactive Web Maps with the JavaScript 'Leaflet' Library
Version 1.1.0
Date 2017-02-17
Maintainer Joe Cheng <joe@rstudio.com>
Description Create and customize interactive maps using the 'Leaflet' JavaScript library and the 'htmlwidgets' package. These maps can be used directly from the R console, from 'RStudio', in Shiny apps and R Markdown documents.
License GPL-3
URL http://rstudio.github.io/leaflet/

BugReports https://github.com/rstudio/leaflet/issues
Depends R (>= 3.1.0)
Imports base64enc, crosstalk, htmlwidgets, htmltools, magrittr, markdown, methods, png, RColorBrewer, raster, scales (>= 0.2.5), sp, stats, viridis
Suggests knitr, maps, sf, shiny, testit (>= 0.4), rgdal, rgeos, R6, RJSONIO, purrr, testthat
RoxygenNote 5.0.1
LazyData true
NeedsCompilation no
Author Joe Cheng [aut, cre], Bhaskar Karambelkar [aut], Yihui Xie [aut], Hadley Wickham [ctb], Kenton Russell [ctb], Kent Johnson [ctb], jQuery Foundation and contributors [ctb, cph] (jQuery library), Vladimir Agafonkin [ctb, cph] (Leaflet library), CloudMade [cph] (Leaflet library),
Leaflet contributors [ctb] (Leaflet library),
Leaflet Providers contributors [ctb, cph] (Leaflet Providers plugin),
Brandon Copeland [ctb, cph] (leaflet-measure plugin),
Jacob Toye [ctb, cph] (Leaflet.label plugin),
Joerg Dietrich [ctb, cph] (Leaflet.Terminator plugin),
Benjamin Becquet [ctb, cph] (Leaflet.MagnifyingGlass plugin),
Norkart AS [ctb, cph] (Leaflet.MiniMap plugin),
L. Voogdt [ctb, cph] (Leaflet.awesome-markers plugin),
Daniel Montague [ctb, cph] (Leaflet.EasyButton plugin),
Kartena AB [ctb, cph] (Proj4Leaflet plugin),
Robert Kajic [ctb, cph] (leaflet-locationfilter plugin),
Mapbox [ctb, cph] (leaflet-omnivore plugin),
Michael Bostock [ctb, cph] (topojson),
RStudio [cph]

Repository CRAN

Date/Publication  2017-02-21 19:08:12

R topics documented:

addAwesomeMarkers ........................................... 3
addControl ...................................................... 4
addGraticule .................................................... 8
addLayersControl .............................................. 9
addLegend .......................................................10
addMeasure .....................................................12
addMiniMap .....................................................14
addProviderTiles .............................................15
addRasterImage ...............................................16
addScaleBar ....................................................18
addSimpleGraticule ...........................................19
addTerminator ................................................20
atlStorms2005 ................................................20
awesomeIconList .............................................21
awesomeIcons ................................................21
breweries91 ....................................................22
colorNumeric ...................................................23
createLeafletMap ..............................................25
derivePoints ...................................................25
derivePolygons ...............................................26
dispatch .........................................................27
easyButtonState ..............................................27
evalFormula ....................................................29
expandLimits ..................................................29
expandLimitsBbox .............................................30
filterNULL .....................................................30
gadmCHE .......................................................31
getMapData ....................................................31
addAwesomeMarkers

iconList ........................................ 32
icons ........................................ 32
leaflet ........................................ 34
leaflet-imports ................................ 39
leafletDependencies ......................... 40
leafletOutput ................................ 40
leafletProxy .................................. 41
makeAwesomeIcon .............................. 42
makeIcon ...................................... 43
mapOptions .................................... 44
previewColors ................................ 45
providers ..................................... 45
providers.details .............................. 46
removeControl ................................ 46
setView ....................................... 48
showGroup .................................... 49
tileOptions .................................. 49
validateCoords ................................ 52

Index 53

addAwesomeMarkers  Add Awesome Markers

Description

Add Awesome Markers

Usage

addAwesomeMarkers(map, lng = NULL, lat = NULL, layerId = NULL,
group = NULL, icon = NULL, popup = NULL, popupOptions = NULL,
label = NULL, labelOptions = NULL, options = markerOptions(),
clusterOptions = NULL, clusterId = NULL, data = getMapData(map))

Arguments

map  the map to add awesome Markers to.

lng  a numeric vector of longitudes, or a one-sided formula of the form ~x where x is
a variable in data; by default (if not explicitly provided), it will be automatically
inferred from data by looking for a column named lng, long, or longitude
(case-insensitively)

lat  a vector of latitudes or a formula (similar to the lng argument; the names lat
and latitude are used when guessing the latitude column from data)

layerId  the layer id
addControl

Graphics elements and layers

Description

Add graphics elements and layers to the map widget.

Options to highlight shapes (polylines/polylines/circles/rectangles)

Usage

addControl(map, html, position = c("topleft", "topright", "bottomleft", "bottomright"), layerId = NULL, className = "info legend", data = getMapData(map))

addTiles(map, urlTemplate = "//{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", attribution = NULL, layerId = NULL, group = NULL, options = tileOptions())

addWMSTiles(map, baseUrl, layerId = NULL, group = NULL, options = WMSTileOptions(), attribution = NULL, layers = ")

addPopups(map, lng = NULL, lat = NULL, popup, layerId = NULL, group = NULL, options = popupOptions(), data = getMapData(map))

addMarkers(map, lng = NULL, lat = NULL, layerId = NULL, group = NULL,
addControl

icon = NULL, popup = NULL, popupOptions = NULL, label = NULL, labelOptions = NULL, options = markerOptions(), clusterOptions = NULL, clusterId = NULL, data = getMapData(map))

addLabelOnlyMarkers(map, lng = NULL, lat = NULL, layerId = NULL, group = NULL, icon = NULL, label = NULL, labelOptions = NULL, options = markerOptions(), clusterOptions = NULL, clusterId = NULL, data = getMapData(map))

addCircleMarkers(map, lng = NULL, lat = NULL, radius = 10, layerId = NULL, group = NULL, stroke = TRUE, color = "#03F", weight = 5, opacity = 0.5, fill = TRUE, fillColor = color, fillOpacity = 0.2, dashArray = NULL, popup = NULL, popupOptions = NULL, label = NULL, labelOptions = NULL, options = pathOptions(), clusterOptions = NULL, clusterId = NULL, data = getMapData(map))

highlightOptions(stroke = NULL, color = NULL, weight = NULL, opacity = NULL, fill = NULL, fillColor = NULL, fillOpacity = NULL, dashArray = NULL, bringToFront = NULL, sendToBack = NULL)

addCircles(map, lng = NULL, lat = NULL, radius = 10, layerId = NULL, group = NULL, stroke = TRUE, color = "#03F", weight = 5, opacity = 0.5, fill = TRUE, fillColor = color, fillOpacity = 0.2, dashArray = NULL, popup = NULL, popupOptions = NULL, label = NULL, labelOptions = NULL, options = pathOptions(), highlightOptions = NULL, data = getMapData(map))

addPolylines(map, lng = NULL, lat = NULL, layerId = NULL, group = NULL, stroke = TRUE, color = "#03F", weight = 5, opacity = 0.5, fill = FALSE, fillColor = color, fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE, popup = NULL, popupOptions = NULL, label = NULL, labelOptions = NULL, options = pathOptions(), highlightOptions = NULL, data = getMapData(map))

addRectangles(map, lng1, lat1, lng2, lat2, layerId = NULL, group = NULL, stroke = TRUE, color = "#03F", weight = 5, opacity = 0.5, fill = TRUE, fillColor = color, fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE, popup = NULL, popupOptions = NULL, label = NULL, labelOptions = NULL, options = pathOptions(), highlightOptions = NULL, data = getMapData(map))

addPolygons(map, lng = NULL, lat = NULL, layerId = NULL, group = NULL, stroke = TRUE, color = "#03F", weight = 5, opacity = 0.5, fill = TRUE, fillColor = color, fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE, popup = NULL, popupOptions = NULL, label = NULL, labelOptions = NULL, options = pathOptions(), highlightOptions = NULL, data = getMapData(map))
addGeoJSON(map, geojson, layerId = NULL, group = NULL, stroke = TRUE, 
color = "#03F", weight = 5, opacity = 0.5, fill = TRUE, 
fillColor = color, fillOpacity = 0.2, dashArray = NULL, 
smoothFactor = 1, noClip = FALSE, options = pathOptions())

addTopoJSON(map, topojson, layerId = NULL, group = NULL, stroke = TRUE, 
color = "#03F", weight = 5, opacity = 0.5, fill = TRUE, 
fillColor = color, fillOpacity = 0.2, dashArray = NULL, 
smoothFactor = 1, noClip = FALSE, options = pathOptions())

Arguments

map a map widget object created from leaflet()
html the content of the control. May be provided as string or as HTML generated 
with Shiny/htmltools tags
position position of control: 'topleft', 'topright', 'bottomleft', or 'bottomright'
layerId the layer id
className extra CSS classes to append to the control, space separated
data the data object from which the argument values are derived; by default, it is the 
data object provided to leaflet() initially, but can be overridden
urlTemplate a character string as the URL template
attribution the attribution text of the tile layer (HTML)
group the name of the group the newly created layers should belong to (for clearGroup 
and addLayersControl purposes). Human-friendly group names are permitted–
they need not be short, identifier-style names. Any number of layers and even 
different types of layers (e.g. markers and polygons) can share the same group 
name.
options a list of extra options for tile layers, popups, paths (circles, rectangles, polygons, 
...), or other map elements
baseUrl a base URL of the WMS service
layers comma-separated list of WMS layers to show
lng a numeric vector of longitudes, or a one-sided formula of the form ~x where x is a variable in data; by default (if not explicitly provided), it will be automatically inferred from data by looking for a column named lng, long, or longitude (case-insensitively)
lat a vector of latitudes or a formula (similar to the lng argument; the names lat and latitude are used when guessing the latitude column from data)
popup a character vector of the HTML content for the popups (you are recommended to escape the text using htmlEscape())
icon the icon(s) for markers; an icon is represented by an R list of the form list(iconUrl = '?', iconSize = 
and you can use icons() to create multiple icons; note when you use an R list 
that contains images as local files, these local image files will be base64 encoded into the HTML page so the icon images will still be available even when you publish the map elsewhere
PopupOptions: A Vector of `popupOptions` to provide popups for security reasons.

Label: a character vector of the HTML content for the labels.

LabelOptions: A Vector of `labelOptions` to provide label options for each label. Default NULL.

ClusterOptions: if not NULL, markers will be clustered using `Leaflet.markerCluster`; you can use `markerClusterOptions()` to specify marker cluster options.

ClusterId: the id for the marker cluster layer.

Radius: a numeric vector of radii for the circles; it can also be a one-sided formula, in which case the radius values are derived from the data (units in meters for circles, and pixels for circle markers).

Stroke: whether to draw stroke along the path (e.g. the borders of polygons or circles).

Color: stroke color.

Weight: stroke width in pixels.

Opacity: stroke opacity (or layer opacity for tile layers).

Fill: whether to fill the path with color (e.g. filling on polygons or circles).

FillColor: fill color.

FillOpacity: fill opacity.

DashArray: a string that defines the stroke dash pattern.

BringToFront: Whether the shape should be brought to front on hover.

SendToBack: whether the shape should be sent to back on mouse out.

HighlightOptions: Options for highlighting the shape on mouse over.

SmoothFactor: how much to simplify the polyline on each zoom level (more means better performance and less accurate representation).

NoClip: whether to disable polyline clipping.

Lng1, Lat1, Lng2, Lat2: latitudes and longitudes of the south-west and north-east corners of rectangles.

GeoJSON: a GeoJSON list, or character vector of length 1.

TopoJSON: a TopoJSON list, or character vector of length 1.

Value: the new map object.

Functions:

- `addControl`: Add arbitrary HTML controls to the map.
- `addTiles`: Add a tile layer to the map.
- `addWMSTiles`: Add a WMS tile layer to the map.
- `addPopups`: Add popups to the map.
- `addMarkers`: Add markers to the map.
- `addLabelOnlyMarkers`: Add Label only markers to the map.
• addCircleMarkers: Add circle markers to the map
• highlightOptions: Options to highlight a shape on hover
• addCircles: Add circles to the map
• addPolylines: Add polylines to the map
• addRectangles: Add rectangles to the map
• addPolygons: Add polygons to the map
• addGeoJSON: Add GeoJSON layers to the map
• addTopoJSON: Add TopoJSON layers to the map

References
The Leaflet API documentation: http://leafletjs.com/reference.html

See Also
tileOptions, WMSTileOptions, popupOptions, markerOptions, pathOptions

---

addGraticule          Add a Graticule on the map see https://github.com/turban/Leaflet.Graticule

Description
Add a Graticule on the map see https://github.com/turban/Leaflet.Graticule

Usage
addGraticule(map, interval = 20, sphere = FALSE, style = list(color = 
"#333", weight = 1), layerId = NULL, group = NULL,
options = pathOptions(pointerEvents = "none", clickable = FALSE))

Arguments
map                 a map widget object
interval            The spacing in map units between horizontal and vertical lines.
sphere              boolean. Default FALSE
style               path options for the generated lines. See http://leafletjs.com/reference.
options             the path options for the graticule layer
layerId             the layer id
group               the name of the group this layer belongs to.
addLayersControl

Examples

```r
library(leaflet)

leaf <- leaflet()
  addTiles()
  addGraticule()
```

addLayersControl | Add UI controls to switch layers on and off

Description

Uses Leaflet's built-in layers control feature to allow users to choose one of several base layers, and to choose any number of overlay layers to view.

Usage

```r
addLayersControl(map, baseGroups = character(),
  overlayGroups = character(), position = c("topright", "bottomright",
  "bottomleft", "topleft"), options = layersControlOptions())
```

```r
layersControlOptions(collapsed = TRUE, autoZIndex = TRUE, ...)
```

```r
removeLayersControl(map)
```

Arguments

- **map**
  - the map to add the layers control to

- **baseGroups**
  - character vector where each element is the name of a group. The user will be able to choose one base group (only) at a time. This is most commonly used for mostly-opaque tile layers.

- **overlayGroups**
  - character vector where each element is the name of a group. The user can turn each overlay group on or off independently.

- **position**
  - position of control: 'topleft', 'topright', 'bottomleft', or 'bottomright'

- **options**
  - a list of additional options, intended to be provided by a call to layersControlOptions

- **collapsed**
  - if TRUE (the default), the layers control will be rendered as an icon that expands when hovered over. Set to FALSE to have the layers control always appear in its expanded state.

- **autoZIndex**
  - if TRUE, the control will automatically maintain the z-order of its various groups as overlays are switched on and off.

- **...**
  - other options for layersControlOptions()
addLegend

Add a color legend to a map

Description

When a color palette function is used in a map (e.g. colorNumeric), a color legend can be automatically derived from the palette function. You can also manually specify the colors and labels for the legend.

Usage

addLegend(map, position = c("topright", "bottomright", "bottomleft", "topleft"), pal, values, na.label = "NA", bins = 7, colors, opacity = 0.5, labels, labFormat = labelFormat(), title = NULL, className = "info legend", layerId = NULL)

labelFormat(prefix = ",", suffix = ",", between = " &ndash; ", digits = 3, big.mark = ",", transform = identity)

Arguments

map a map widget object created from leaflet()
position the position of the legend
pal the color palette function, generated from colorNumeric(), colorBin(), colorQuantile(), or colorFactor()
values the values used to generate colors from the palette function
na.label the legend label for NAs in values
bins an approximate number of tick-marks on the color gradient for the colorNumeric palette if it is of length one; you can also provide a numeric vector as the predefined breaks (equally spaced)
colors a vector of (HTML) colors to be used in the legend if pal is not provided
opacity the opacity of colors
addLegend

labels a vector of text labels in the legend corresponding to colors
labFormat a function to format the labels derived from pal and values (see Details below to know what labFormat() returns by default; you can either use the helper function labFormat(), or write your own function)
title the legend title
className extra CSS classes to append to the control, space separated
layerId the ID of the legend; subsequent calls to addLegend or addControl with the same layerId will replace this legend. The ID can also be used with removeControl.
prefix a prefix of legend labels
suffix a suffix of legend labels
between a separator between x[i] and x[i + 1] in legend labels (by default, it is a dash)
digits the number of digits of numeric values in labels
big.mark the thousand separator
transform a function to transform the label value

details

The labFormat argument is a function that takes the argument type = c("numeric", "bin", "quantile", "factor"), plus, arguments for different types of color palettes. For the colorNumeric() palette, labFormat takes a single argument, which is the breaks of the numeric vector, and returns a character vector of the same length. For colorBin(), labFormat also takes a vector of breaks of length n but should return a character vector of length n - 1, with the i-th element representing the interval c(x[i], x[i + 1]). For colorQuantile, labFormat takes two arguments, the quantiles and the associated probabilities (each of length n), and should return a character vector of length n - 1 (similar to the colorBin() palette). For colorFactor(), labFormat takes one argument, the unique values of the factor, and should return a character vector of the same length.

By default, labFormat is basically format(scientific = FALSE, big.mark = ', ') for the numeric palette, as.character() for the factor palette, and a function to return labels of the form 'x[i] - x[i + 1]' for bin and quantile palettes (in the case of quantile palettes, x is the probabilities instead of the values of breaks).

examples

# !formatR
library(leaflet)
# a manual legend
leaflet() %>% addTiles() %>% addLegend(
    position = 'bottomright',
    colors = rgb(t(col2rgb(palette()))) / 255,
    labels = palette(), opacity = 1,
    title = 'An Obvious Legend'
)

# an automatic legend derived from the color palette
df = local({
    n = 300; x = rnorm(n); y = rnorm(n)
    z = sqrt(x^2 + y^2); z[sample(n, 10)] = NA
})
```r

data.frame(x, y, z)
})
pal = colorNumeric('OrRd', df$z)
leaflet(df) %>%
  addCircleMarkers(~x, ~y, color = ~pal(z)) %>%
  addLegend(pal = pal, values = ~z)

# format legend labels
df = data.frame(x = rnorm(100), y = rexp(100, 2), z = runif(100))
pal = colorBin('PuOr', df$z, bins = c(0, .1, .4, .9, 1))
leaflet(df) %>%
  addCircleMarkers(~x, ~y, color = ~pal(z)) %>%
  addLegend(pal = pal, values = ~z)

leaflet(df) %>%
  addCircleMarkers(~x, ~y, color = ~pal(z)) %>%
  addLegend(pal = pal, values = ~z, labFormat = labelFormat(
    prefix = '(', suffix = ')', between = ',',
    transform = function(x) 100 * x
  ))
```

---

addMeasure

Add a measure control to the map.

**Description**

Add a measure control to the map.

**Usage**

```r
addMeasure(map, position = "topright", primaryLengthUnit = "feet",
  secondaryLengthUnit = NULL, primaryAreaUnit = "acres",
  secondaryAreaUnit = NULL, activeColor = "#ABE67E",
  completedColor = "#CF8F2BE", popupOptions = list(className =
    "leaflet-measure-resultpopup", autoPanPadding = c(10, 10)),
  captureZIndex = 10000, localization = "en", decPoint = ".",
  thousandsSep = "",)
```

**Arguments**

- `map` a map widget object
- `position` standard Leaflet control position options.
- `primaryLengthUnit`, `secondaryLengthUnit` units used to display length results. `secondaryLengthUnit` is optional. Valid values are "feet", "meters", "miles", and "kilometers".
- `primaryAreaUnit`, `secondaryAreaUnit` units used to display area results. `secondaryAreaUnit` is optional. Valid values are "acres", "hectares", "sqmeters", and "sqmiles".
**activeColor**  
Base color to use for map features rendered while actively performing a measurement. Value should be a color represented as a hexadecimal string.

**completedColor**  
Base color to use for features generated from a completed measurement. Value should be a color represented as a hexadecimal string.

**popupOptions**  
List of options applied to the popup of the resulting measure feature. Properties may be any standard Leaflet popup options.

**captureZIndex**  
Z-index of the marker used to capture measure clicks. Set this value higher than the z-index of all other map layers to disable click events on other layers while a measurement is active.

**localization**  
Locale to translate displayed text. Available locales include en (default), cn, de, es, fr, it, nl, pt, pt_BR, pt_PT, ru, and tr.

**decPoint**  
Decimal point used when displaying measurements. If not specified, values are defined by the localization.

**thousandsSep**  
Thousands separator used when displaying measurements. If not specified, values are defined by the localization.

---

### Value

Modified map

### Examples

```r
library(leaflet)

leaf <- leaflet() %>%
  addTiles() %>%
  # central park
  fitBounds( -73.9, 40.75, -73.95, 40.8 ) %>%
  addMeasure()

leaf

# customizing
leaf %>% addMeasure(
  position = "bottomleft",
  primaryLengthUnit = "meters",
  primaryAreaUnit = "sqmeters",
  activeColor = "#3D535D",
  completedColor = "#7D4479",
  localization = 'de'
)
```
**addMiniMap**

Add a minimap to the Map [https://github.com/Norkart/Leaflet-MiniMap](https://github.com/Norkart/Leaflet-MiniMap)

## Description

Add a minimap to the Map [https://github.com/Norkart/Leaflet-MiniMap](https://github.com/Norkart/Leaflet-MiniMap)

## Usage

```
addMiniMap(map, position = "bottomright", width = 150, height = 150,
            collapsedWidth = 19, collapsedHeight = 19, zoomLevelOffset = -5,
            zoomLevelFixed = FALSE, centerFixed = FALSE, zoomAnimation = FALSE,
            toggleDisplay = FALSE, autoToggleDisplay = FALSE, minimized = FALSE,
            aimingRectOptions = list(color = "#ff7800", weight = 1, clickable = FALSE),
            shadowRectOptions = list(color = "#000000", weight = 1, clickable = FALSE,
                                    opacity = 0, fillOpacity = 0), strings = list(hideText = "Hide MiniMap",
                                                   showText = "Show MiniMap"), tiles = NULL, mapOptions = list())
```

## Arguments

- **map**
  a map widget object

- **position**
  The standard Leaflet.Control position parameter, used like all the other controls. Defaults to 'bottomright'.

- **width**
  The width of the minimap in pixels. Defaults to 150.

- **height**
  The height of the minimap in pixels. Defaults to 150.

- **collapsedWidth**
  The width of the toggle marker and the minimap when collapsed, in pixels. Defaults to 19.

- **collapsedHeight**
  The height of the toggle marker and the minimap when collapsed, in pixels. Defaults to 19.

- **zoomLevelOffset**
  The offset applied to the zoom in the minimap compared to the zoom of the main map. Can be positive or negative, defaults to -5.

- **zoomLevelFixed**
  Overrides the offset to apply a fixed zoom level to the minimap regardless of the main map zoom. Set it to any valid zoom level, if unset zoomLevelOffset is used instead.

- **centerFixed**
  Applies a fixed position to the minimap regardless of the main map’s view / position. Prevents panning the minimap, but does allow zooming (both in the minimap and the main map). If the minimap is zoomed, it will always zoom around the centerFixed point. You can pass in a Lat/Lng-equivalent object. Defaults to false.

- **zoomAnimation**
  Sets whether the minimap should have an animated zoom. (Will cause it to lag a bit after the movement of the main map.) Defaults to false.
toggleDisplay
Sets whether the minimap should have a button to minimise it. Defaults to false.

autoToggleDisplay
Sets whether the minimap should hide automatically, if the parent map bounds does not fit within the minimap bounds. Especially useful when 'zoomLevelFixed' is set.

minimized
Sets whether the minimap should start in a minimized position.

aimingRectOptions
Sets the style of the aiming rectangle by passing in a Path.Options (http://leafletjs.com/reference.html#path-options) object. (Clickable will always be overridden and set to false.)

shadowRectOptions
Sets the style of the aiming shadow rectangle by passing in a Path.Options (http://leafletjs.com/reference.html#path-options) object. (Clickable will always be overridden and set to false.)

strings
Overrides the default strings allowing for translation.

tiles
URL for tiles or one of the pre-defined providers.

mapOptions
Sets Leaflet options for the MiniMap map. It does not override the MiniMap default map options but extends them.

See Also
providers

Examples

library(leaflet)

leaf <- leaflet() %>%
  addTiles() %>%
  addMiniMap()

Description
Add a tile layer from a known map provider

Usage

addProviderTiles(map, provider, layerId = NULL, group = NULL,
                 options = providerTileOptions())

providerTileOptions(errorTileUrl = "", noWrap = FALSE, opacity = NULL,
                     zIndex = NULL, unloadInvisibleTiles = NULL, updateWhenIdle = NULL,
                     detectRetina = FALSE, reuseTiles = FALSE, ...)
Arguments

map the map to add the tile layer to

provider the name of the provider (see http://leaflet-extras.github.io/leaflet-providers/preview/ and https://github.com/leaflet-extras/leaflet-providers)

layerId the layer id to assign

group the name of the group the newly created layers should belong to (for clearGroup and addLayersControl purposes). Human-friendly group names are permitted—they need not be short, identifier-style names.

options tile options

errorTileUrl, nowrap, opacity, zIndex, unloadInvisibleTiles, updateWhenIdle, detectRetina, reuseTiles the tile layer options; see http://leafletjs.com/reference.html#tilelayer

... named parameters to add to the options

Value

modified map object

Examples

```r
leaflet() %>%
  addProviderTiles("Stamen.Watercolor") %>%
  addProviderTiles("Stamen.TonerHybrid")
```

---

addRasterImage Add a raster image as a layer

Description

Create an image overlay from a RasterLayer object. This is only suitable for small to medium sized rasters, as the entire image will be embedded into the HTML page (or passed over the websocket in a Shiny context).

Usage

```r
addRasterImage(map, x, colors = "Spectral", opacity = 1,
               attribution = NULL, layerId = NULL, group = NULL, project = TRUE,
               maxBytes = 4 * 1024 * 1024)

projectRasterForLeaflet(x)
```
Arguments

- **map**: a map widget object
- **x**: a RasterLayer object—see `raster`
- **colors**: the color palette (see `colorNumeric`) or function to use to color the raster values (hint: if providing a function, set `na.color` to "#00000000" to make NA areas transparent)
- **opacity**: the base opacity of the raster, expressed from 0 to 1
- **attribution**: the HTML string to show as the attribution for this layer
- **layerId**: the layer id
- **group**: the name of the group this raster image should belong to (see the same parameter under `addTiles`)
- **project**: if TRUE, automatically project x to the map projection expected by Leaflet (EPSG:3857); if FALSE, it’s the caller’s responsibility to ensure that x is already projected, and that `extent(x)` is expressed in WGS84 latitude/longitude coordinates
- **maxBytes**: the maximum number of bytes to allow for the projected image (before base64 encoding); defaults to 4MB.

Details

The `maxBytes` parameter serves to prevent you from accidentally embedding an excessively large amount of data into your htmlwidget. This value is compared to the size of the final compressed image (after the raster has been projected, colored, and PNG encoded, but before base64 encoding is applied). Set `maxBytes` to `Inf` to disable this check, but be aware that very large rasters may not only make your map a large download but also may cause the browser to become slow or unresponsive.

By default, the `addRasterImage` function will project the RasterLayer `x` to EPSG:3857 using the `raster` package’s `projectRaster` function. This can be a time-consuming operation for even moderately sized rasters. Upgrading the raster package to 2.4 or later will provide a large speedup versus previous versions. If you are repeatedly adding a particular raster to your Leaflet maps, you can perform the projection ahead of time using `projectRasterForLeaflet()`, and call `addRasterImage` with `project=FALSE`.

Examples

```
library(raster)

r <- raster(xmn=-2.8, xmx=-2.79, ymn=54.04, ymx=54.05, nrows=30, ncols=30)
values(r) <- matrix(1:900, nrow(r), ncol(r), byrow = TRUE)
crs(r) <- CRS("+init=epsg:4326")

leaflet() %>% addTiles() %>%
  addRasterImage(r, colors = "Spectral", opacity = 0.8)
```
addScaleBar  

*Add or remove a scale bar*

**Description**

Uses Leaflet’s built-in scale bar feature to add a scale bar.

**Usage**

```r
addScaleBar(map, position = c("topright", "bottomright", "bottomleft", "topleft"), options = scaleBarOptions())

scaleBarOptions(maxWidth = 100, metric = TRUE, imperial = TRUE, updateWhenIdle = TRUE)
```

```r
removeScaleBar(map)
```

**Arguments**

- `map`  
  the map to add the scale bar to

- `position`  
  position of control: 'topleft', 'topright', 'bottomleft', or 'bottomright'

- `options`  
  a list of additional options, intended to be provided by a call to `scaleBarOptions`

- `maxWidth`  
  maximum width of the control in pixels (default 100)

- `metric`  
  if TRUE (the default), show a scale bar in metric units (m/km)

- `imperial`  
  if TRUE (the default), show a scale bar in imperial units (ft/mi)

- `updateWhenIdle`  
  if FALSE (the default), the scale bar is always up-to-date (updated on move). If TRUE, the control is updated on moveend.

**Examples**

```r
leaflet() %>%
  addTiles() %>%
  addScaleBar()
```
addSimpleGraticule

Add a simple Graticule on the map see https://github.com/ablakey/Leaflet.SimpleGraticule

Description

Add a simple Graticule on the map see https://github.com/ablakey/Leaflet.SimpleGraticule

Usage

addSimpleGraticule(map, interval = 20, showOriginLabel = TRUE,
                  redraw = "move", hidden = FALSE, zoomIntervals = list(),
                  layerId = NULL, group = NULL)

Arguments

map a map widget object
interval The spacing in map units between horizontal and vertical lines.
showOriginLabel true Whether or not to show `(0,0)` at the origin.
redraw on which map event to redraw the graticule. On move is default but moveend can be smoother.
hidden hide on start
zoomIntervals use different intervals in different zoom levels. If not specified, all zoom levels use value in interval option.
layerId the layer id
group the name of the group this layer belongs to.

Examples

library(leaflet)

leaflet() %>%
  addTiles() %>%
  addSimpleGraticule()
addTerminator  Add a daylight layer on top of the map

Description

See https://github.com/joergdietrich/Leaflet.Terminator

Usage

addTerminator(map, resolution = 2, time = NULL, layerId = NULL,
group = NULL, options = pathOptions(pointerEvents = "none", clickable =
FALSE))

Arguments

map  a map widget object
resolution  the step size at which the terminator points are computed. The step size is 1
degree/resolution, i.e. higher resolution values have smaller step sizes and more
points in the polygon. The default value is 2.
time  Time
layerId  the layer id
group  the name of the group this layer belongs to.
options  the path options for the daynight layer

Examples

library(leaflet)

leaf <- leaflet() %>%
  addTiles() %>%
  addTerminator()

atlStorms2005  Atlantic Ocean storms 2005

Description

Atlantic Ocean storms 2005

Format

sp::SpatialLinesDataFrame

Details

This dataset contains storm tracks for selected storms in the Atlantic Ocean basin for the year 2005
awesomeIconList  Make awesome-icon set

Description
Make awesome-icon set

Usage
awesomeIconList(...)

Arguments
...  icons created from makeAwesomeIcon()

Examples
iconSet = awesomeIconList(
    home = makeAwesomeIcon(icon='Home', library='fa'),
    flag = makeAwesomeIcon(icon='Flag', library='fa')
)
iconSet[c('home', 'flag')]

awesomeIcons  Create a list of awesome icon data see https://github.com/lvoogdt/Leaflet.awesome-markers

Description
An icon can be represented as a list of the form list(icon, library,...). This function is vectorized over its arguments to create a list of icon data. Shorter argument values will be re-cycled. NULL values for these arguments will be ignored.

Usage
awesomeIcons(icon = "home", library = "glyphicon", markerColor = "blue",
iconColor = "white", spin = FALSE, extraClasses = NULL,
squareMarker = FALSE, iconRotate = 0, fontFamily = "monospace",
text = NULL)
Arguments

icon Name of the icon
library Which icon library. Default 'glyphicon', other possible values are 'fa' (fontawesome) or 'ion' (ionicons).
markerColor Possible values are 'red', 'darkred', 'lightred', 'orange', 'beige', 'green', 'darkgreen', 'lightgreen', 'blue', 'darkblue', 'lightblue', 'purple', 'darkpurple', 'pink', 'cadetblue', 'white', 'gray', 'lightgray', 'black'
iconColor The color to use for the icon itself. Use any CSS-valid color (hex, rgba, etc.) or a named web color.
spin If TRUE, make the icon spin (only works when library = 'fa')
extraClasses Additional css classes to include on the icon.
squareMarker Whether to use a square marker.
iconRotate Rotate the icon by a given angle.
fontFamily Used when text option is specified.
text Use this text string instead of an icon. argument of addAwesomeMarkers().

Value

A list of awesome-icon data that can be passed to the icon

breweries91

Selected breweries in Franconia

Description

Selected breweries in Franconia (zip code starting with 91...)

Format

sp::SpatialPointsDataFrame

Details

This dataset contains selected breweries in Franconia. It is a subset of a larger database that was compiled by students at the University of Marburg for a seminar called "The Geography of Beer, sustainability in the food industry"
**colorNumeric**  

**Color mapping**

**Description**
Conveniently maps data values (numeric or factor/character) to colors according to a given palette, which can be provided in a variety of formats.

**Usage**

```
colorNumeric(palette, domain, na.color = "#808080", alpha = FALSE, reverse = FALSE)
```

```
colorBin(palette, domain, bins = 7, pretty = TRUE, na.color = "#808080", alpha = FALSE, reverse = FALSE)
```

```
colorQuantile(palette, domain, n = 4, probs = seq(0, 1, length.out = n + 1), na.color = "#808080", alpha = FALSE, reverse = FALSE)
```

```
colorFactor(palette, domain, levels = NULL, ordered = FALSE, na.color = "#808080", alpha = FALSE, reverse = FALSE)
```

**Arguments**

- **palette**
  The colors or color function that values will be mapped to

- **domain**
  The possible values that can be mapped.
  For `colorNumeric` and `colorBin`, this can be a simple numeric range (e.g. `c(0, 100)`); `colorQuantile` needs representative numeric data; and `colorFactor` needs categorical data.
  If `NULL`, then whenever the resulting color function is called, the x value will represent the domain. This implies that if the function is invoked multiple times, the encoding between values and colors may not be consistent; if consistency is needed, you must provide a non-`NULL` domain.

- **na.color**
  The color to return for NA values. Note that `na.color=NA` is valid.

- **alpha**
  Whether alpha channels should be respected or ignored. If `TRUE` then colors without explicit alpha information will be treated as fully opaque.

- **reverse**
  Whether the colors (or color function) in `palette` should be used in reverse order. For example, if the default order of a palette goes from blue to green, then `reverse = TRUE` will result in the colors going from green to blue.

- **bins**
  Either a numeric vector of two or more unique cut points or a single number (greater than or equal to 2) giving the number of intervals into which the domain values are to be cut.

- **pretty**
  Whether to use the function `pretty()` to generate the bins when the argument `bins` is a single number. When `pretty = TRUE`, the actual number of bins may not be the number of bins you specified. When `pretty = FALSE`, `seq()` is used to generate the bins and the breaks may not be "pretty".
n
Number of equal-size quantiles desired. For more precise control, use the `probs` argument instead.
probs
See `quantile`. If provided, the `n` argument is ignored.
levels
An alternate way of specifying levels; if specified, domain is ignored.
ordered
If TRUE and domain needs to be coerced to a factor, treat it as already in the correct order.

Details

colorNumeric is a simple linear mapping from continuous numeric data to an interpolated palette.
colorBin also maps continuous numeric data, but performs binning based on value (see the `cut` function).
colorQuantile similarly bins numeric data, but via the `quantile` function.
colorFactor maps factors to colors. If the palette is discrete and has a different number of colors than the number of factors, interpolation is used.

The palette argument can be any of the following:

1. A character vector of RGB or named colors. Examples: `palette(c("#000000", "#0000FF", "#FFFFFF"), topo.colors(10))`
2. The name of an RColorBrewer palette, e.g. "BuPu" or "Greens".
3. The full name of a viridis palette: "viridis", "magma", "inferno", or "plasma".
4. A function that receives a single value between 0 and 1 and returns a color. Examples: `colorRamp(c("#000000", "#FFFFFF"), interpolate="spline")`.

Value

A function that takes a single parameter `x`; when called with a vector of numbers (except for `colorFactor`, which expects factors/characters), #RRGGBB color strings are returned (unless `alpha=TRUE` in which case #RRGGBBAA may also be possible).

Examples

```r
pal = colorBin("Greens", domain = 0:100)
pal(runif(10, 60, 100))
```

# Exponential distribution, mapped continuously
previewColors(colorNumeric("Blues", domain = NULL), sort(rexp(16)))
# Exponential distribution, mapped by interval
previewColors(colorBin("Blues", domain = NULL, bins = 4), sort(rexp(16)))
# Exponential distribution, mapped by quantile
previewColors(colorQuantile("Blues", domain = NULL), sort(rexp(16)))

# Categorical data; by default, the values being colored span the gamut...
previewColors(colorFactor("RdYlBu", domain = NULL, LETTERS[1:5]))
# ...unless the data is a factor, without droplevels...
previewColors(colorFactor("RdYlBu", domain = NULL), factor(LETTERS[1:5], levels=LETTERS))
# ...or the domain is stated explicitly.
createLeafletMap

Legacy functions

Description

These functions are provided for backwards compatibility with the first iteration of the leaflet bindings (https://github.com/jcheng5/leaflet-shiny).

Usage

createLeafletMap(session, outputId)

leafletMap(outputId, width, height,
  initialTileLayer = "http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
  initialTileLayerAttribution = NULL, options = NULL)

Arguments

  session, outputId
    Deprecated
  width, height, initialTileLayer, initialTileLayerAttribution, options
    Deprecated

derivePoints

Given a data object and lng/lat arguments (which may be NULL [meaning infer from data], formula [which should be evaluated with respect to the data], or vector data [which should be used as-is]) return a lng/lat data frame.

Description

Given a data object and lng/lat arguments (which may be NULL [meaning infer from data], formula [which should be evaluated with respect to the data], or vector data [which should be used as-is]) return a lng/lat data frame.

Usage

derivePoints(data, lng = NULL, lat = NULL, missingLng = missing(lng),
  missingLat = missing(lat), funcName = "f")
derivePolygons

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>map data</td>
</tr>
<tr>
<td>lng</td>
<td>longitude</td>
</tr>
<tr>
<td>lat</td>
<td>latitude</td>
</tr>
<tr>
<td>missingLng</td>
<td>whether lng is missing</td>
</tr>
<tr>
<td>missingLat</td>
<td>whether lat is missing</td>
</tr>
<tr>
<td>funcName</td>
<td>Name of calling function (for logging)</td>
</tr>
</tbody>
</table>

derivePolygons: Given a data object and lng/lat arguments (which may be NULL [meaning infer from data], formula [which should be evaluated with respect to the data], or vector data [which should be used as-is]) return a spatial object

Description

Given a data object and lng/lat arguments (which may be NULL [meaning infer from data], formula [which should be evaluated with respect to the data], or vector data [which should be used as-is]) return a spatial object

Usage

derivePolygons(data, lng = NULL, lat = NULL, missingLng = missing(lng), missingLat = missing(lat), funcName = "f")

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>map data</td>
</tr>
<tr>
<td>lng</td>
<td>longitude</td>
</tr>
<tr>
<td>lat</td>
<td>latitude</td>
</tr>
<tr>
<td>missingLng</td>
<td>whether lng is missing</td>
</tr>
<tr>
<td>missingLat</td>
<td>whether lat is missing</td>
</tr>
<tr>
<td>funcName</td>
<td>Name of calling function (for logging)</td>
</tr>
</tbody>
</table>
dispatch

Extension points for plugins

Description

Extension points for plugins

Usage

dispatch(map, funcName, leaflet = stop(paste(funcName, "requires a map proxy object")), leaflet_proxy = stop(paste(funcName, "does not support map proxy objects")))

invokeMethod(map, data, method, ...)

Arguments

- `map` a map object, as returned from `leaflet` or `leaflet_proxy`
- `funcName` the name of the function that the user called that caused this dispatch call; for error message purposes
- `leaflet` an action to be performed if the map is from `leaflet`
- `leaflet_proxy` an action to be performed if the map is from `leafletProxy`
- `data` a data object that will be used when evaluating formulas in ...
- `method` the name of the JavaScript method to invoke
- ... unnamed arguments to be passed to the JavaScript method

Value

dispatch returns the value of `leaflet` or `leaflet_proxy`, or an error. invokeMethod returns the map object that was passed in, possibly modified.

easyButtonState

Create an easyButton statestate

Description

Create an easyButton statestate

Creates an easy button.

Add a EasyButton on the map see https://github.com/CliffCloud/Leaflet.EasyButton

Add a easyButton bar on the map see https://github.com/CliffCloud/Leaflet.EasyButton
easyButtonState

Usage

easyButtonState(stateName, icon, title, onClick)

```r
easyButton(icon = NULL, title = NULL, onClick = NULL,
    position = "topleft", id = NULL, states = NULL)
```

```r
addEasyButton(map, button)
```

```r
addEasyButtonBar(map, ..., position = "topleft", id = NULL)
```

Arguments

- **statename**: a unique name for the state
- **icon**: the button icon
- **title**: text to show on hover
- **onClick**: the action to take
- **position**: topleft|topright|bottomleft|bottomright
- **id**: id for the button
- **states**: the states
- **map**: a map widget object
- **button**: the button object created with `easyButton`
- **NNN**: a list of buttons created with `easyButton`

Functions

- `easyButtonState`: state of an easyButton.
- `addEasyButton`: add an EasyButton to the map
- `addEasyButtonBar`: add an EasyButton to the map

See Also

- `easyButton`
- [https://github.com/CliffCloud/Leaflet.EasyButton](https://github.com/CliffCloud/Leaflet.EasyButton)
- `addEasyButton`

Examples

```r
library(leaflet)

leaf <- leaflet() %>%
  addTiles() %>%
  addEasyButton(easyButton(
    icon = htmltools::span(class = "star", &star;),
    onClick = JS("function(btn, map){ map.setZoom(1);}")))```
library(leaflet)

leaf <- leaflet()
  addTiles()
  addEasyButtonBar(
    easyButton(
      icon = htmltools::span(class='star', '&starf;'),
      onClick = JS("function(btn, map){ alert('Button 1');}"),
    easyButton(
      icon = htmltools::span(class='star', '&target;'),
      onClick = JS("function(btn, map){ alert('Button 2');}")))

---

evalFormula

Evaluate list members that are formulae, using the map data as the environment (if provided, otherwise the formula environment)

Description

Evaluate list members that are formulae, using the map data as the environment (if provided, otherwise the formula environment)

Usage

evalFormula(list, data)

Arguments

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>with members as formulae</td>
</tr>
<tr>
<td>data</td>
<td>map data</td>
</tr>
</tbody>
</table>

expandLimits

Notifies the map of new latitude/longitude of items of interest on the map

Description

Notifies the map of new latitude/longitude of items of interest on the map

Usage

expandLimits(map, lat, lng)

Arguments

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>map object</td>
</tr>
<tr>
<td>lat</td>
<td>vector of latitudes</td>
</tr>
<tr>
<td>lng</td>
<td>vector of longitudes</td>
</tr>
</tbody>
</table>
### expandLimitsBbox

**Same as expandLimits, but takes a polygon (that presumably has a bbox attr) rather than lat/lng.**

**Description**

Same as expandLimits, but takes a polygon (that presumably has a bbox attr) rather than lat/lng.

**Usage**

```r
expandLimitsBbox(map, poly)
```

**Arguments**

- `map` : map object
- `poly` : A spatial object representing a polygon.

---

### filterNULL

**remove NULL elements from a list**

**Description**

remove NULL elements from a list

**Usage**

```r
filterNULL(x)
```

**Arguments**

- `x` : A list whose NULL elements will be filtered
Administrative borders of Switzerland (level 1)

Description
Administrative borders of Switzerland (level 1)

Format
sp::SpatialPolygonsDataFrame

Details
This dataset comes from http://gadm.org. It was downloaded using getData.

Source
http://gadm.org

gemapData

Description
returns the map’s data

Usage
gemapData(map)

Arguments
map the map
iconList  

Make icon set

Description
Make icon set

Usage
iconList(...)

Arguments
...  icons created from makeIcon()

Examples

iconSet = iconList(
    red = makeIcon("leaf-red.png", iconWidth=32, iconHeight=32),
    green = makeIcon("leaf-green.png", iconWidth=32, iconHeight=32)
)
iconSet[c('red', 'green', 'red')]

icons  

Create a list of icon data

Description
An icon can be represented as a list of the form list(iconUrl, iconSize, ...). This function is vectorized over its arguments to create a list of icon data. Shorter argument values will be re-cycled. NULL values for these arguments will be ignored.

Usage
icons(iconUrl = NULL, iconRetinaUrl = NULL, iconWidth = NULL,
     iconHeight = NULL, iconAnchorX = NULL, iconAnchorY = NULL,
     shadowUrl = NULL, shadowRetinaUrl = NULL, shadowWidth = NULL,
     shadowHeight = NULL, shadowAnchorX = NULL, shadowAnchorY = NULL,
     popupAnchorX = NULL, popupAnchorY = NULL, className = NULL)
**Icons**

**Arguments**

- `iconUrl` the URL or file path to the icon image
- `iconRetinaUrl` the URL or file path to a retina sized version of the icon image
- `iconWidth`, `iconHeight` size of the icon image in pixels
- `iconAnchorX`, `iconAnchorY` the coordinates of the "tip" of the icon (relative to its top left corner, i.e. the top left corner means `iconAnchorX = 0` and `iconAnchorY = 0`), and the icon will be aligned so that this point is at the marker’s geographical location
- `shadowUrl` the URL or file path to the icon shadow image
- `shadowRetinaUrl` the URL or file path to the retina sized version of the icon shadow image
- `shadowWidth`, `shadowHeight` size of the shadow image in pixels
- `shadowAnchorX`, `shadowAnchorY` the coordinates of the "tip" of the shadow
- `popupAnchorX`, `popupAnchorY` the coordinates of the point from which popups will "open", relative to the icon anchor
- `className` a custom class name to assign to both icon and shadow images

**Value**

A list of icon data that can be passed to the `icon` argument of `addMarkers()`.

**Examples**

```r
library(leaflet)

# adapted from http://leafletjs.com/examples/custom-icons.html

iconData = data.frame(
  lat = c(rnorm(10, 0), rnorm(10, 1), rnorm(10, 2)),
  lng = c(rnorm(10, 0), rnorm(10, 3), rnorm(10, 6)),
  group = rep(sort(c('green', 'red', 'orange')), each = 10),
  stringsAsFactors = FALSE
)

leaflet() %>%
  addMarkers(
    data = iconData,
    icon = icons(
      iconUrl = sprintf('http://leafletjs.com/docs/images/leaf-%s.png', group),
      shadowUrl = 'http://leafletjs.com/docs/images/leaf-shadow.png',
      iconWidth = 38, iconHeight = 95, shadowWidth = 50, shadowHeight = 64,
      iconAnchorX = 22, iconAnchorY = 94, shadowAnchorX = 4, shadowAnchorY = 62,
      popupAnchorX = -3, popupAnchorY = -76
    )
  )
```


# use point symbols from base R graphics as icons
pchIcons = function(pch = 0:14, width = 30, height = 30, ...) {
  n = length(pch)
  files = character(n)
  # create a sequence of png images
  for (i in seq_len(n)) {
    f = tempfile(fileext = '.png')
    png(f, width = width, height = height, bg = 'transparent')
    par(mar = c(0, 0, 0, 0))
    plot.new()
    points(.5, .5, pch = pch[i], cex = min(width, height) / 8, ...)
    dev.off()
    files[i] = f
  }
  files
}

iconData = matrix(rnorm(500), ncol = 2)
res = kmeans(iconData, 10)
iconData = cbind(iconData, res$cluster)
colnames(iconData) = c('lat', 'lng', 'group')
iconData = as.data.frame(iconData)

# 10 random point shapes for the 10 clusters in iconData
shapes = sample(0:14, 10)
iconFiles = pchIcons(shapes, 40, 40, col = 'steelblue', lwd = 2)

# note the data has 250 rows, and there are 10 icons in iconFiles; they are
# connected by the `group` variable: the i-th row of iconData uses the
# group[i]-th icon in the icon list
leaflet() %>% addMarkers(
  data = iconData,
  icon = ~ icons(
    iconUrl = iconFiles[groups],
    popupAnchorX = 20, popupAnchorY = 0
  ),
  popup = ~ sprintf(
    'lat = %s, long = %s, group = %s, pch = %s', lat, lng, group, shapes[groups]
  )
)

unlink(iconFiles)  # clean up the tmp png files that have been embedded

---

**leaflet**

*Create a Leaflet map widget*
**leaflet**

**Description**

This function creates a Leaflet map widget using **htmlwidgets**. The widget can be rendered on HTML pages generated from R Markdown, Shiny, or other applications.

Options for Map creation

creates a custom CRS Refer to [https://kartena.github.io/Proj4Leaflet/api/](https://kartena.github.io/Proj4Leaflet/api/) for details.

**Usage**

```r
leaflet(data = NULL, width = NULL, height = NULL, padding = 0,
    options = leafletOptions())

leafletOptions(minZoom = NULL, maxZoom = NULL, crs = leafletCRS(),
    worldCopyJump = NULL, ...)

leafletCRS(crsClass = "L.CRS.EPSG3857", code = NULL, proj4def = NULL,
    projectedBounds = NULL, origin = NULL, transformation = NULL,
    scales = NULL, resolutions = NULL, bounds = NULL, tileSize = NULL)
```

**Arguments**

- **data** a data object. Currently supported objects are matrices, data frames, spatial objects from the **sp** package (SpatialPoints, SpatialPointsDataFrame, Polygon, Polygons, SpatialPolygons, SpatialPolygonsDataFrame, Line, Lines, SpatialLines, and SpatialLinesDataFrame), and spatial data frames from the **sf** package.
- **width** the width of the map
- **height** the height of the map
- **padding** the padding of the map
- **options** the map options
- **minZoom** Minimum zoom level of the map. Overrides any minZoom set on map layers.
- **maxZoom** Maximum zoom level of the map. This overrides any maxZoom set on map layers.
- **crs** Coordinate Reference System to use. Don’t change this if you’re not sure what it means.
- **worldCopyJump** With this option enabled, the map tracks when you pan to another "copy" of the world and seamlessly jumps to the original one so that all overlays like markers and vector layers are still visible.
- **...** other options.
- **crsClass** One of L.CRS.EPSG3857, L.CRS.EPSG4326, L.CRS.EPSG3395, L.CRS.Simple, L.Proj.CRS, L.Proj.CRS.TMS
- **code** CRS identifier
- **proj4def** Proj4 string
- **projectedBounds** Only when crsClass = ’L.Proj.CRS.TMS’
- **origin** Origin in projected coordinates, if set overrides transformation option.
transformation to use when transforming projected coordinates into pixel coordinates
scales Scale factors (pixels per projection unit, for example pixels/meter) for zoom levels; specify either scales or resolutions, not both
resolutions factors (projection units per pixel, for example meters/pixel) for zoom levels; specify either scales or resolutions, not both
bounds Bounds of the CRS, in projected coordinates; if defined, Proj4Leaflet will use this in the getSize method, otherwise defaulting to Leaflet’s default CRS size
tileSize Tile size, in pixels, to use in this CRS (Default 256) Only needed when crsClass = 'L.Proj.CRS.TMS'

Details

The data argument is only needed if you are going to reference variables in this object later in map layers. For example, data can be a data frame containing columns latitude and longitude, then we may add a circle layer to the map by leaflet(data) %>% addCircles(lat = ~latitude, lng = ~longitude), where the variables in the formulae will be evaluated in the data.

Value

A HTML widget object, on which we can add graphics layers using %>% (see examples).

Functions

• leafletOptions: Options for map creation
• leafletCRS: class to create a custom CRS

See Also

leafletCRS for creating a custom CRS.


Examples

# !formatR
library(leaflet)
m = leaflet() %>% addTiles()
m # a map with the default OSM tile layer

# set bounds
m %>% fitBounds(0, 40, 10, 50)

# move the center to Snedecor Hall
m = m %>% setView(-93.65, 42.0285, zoom = 17)
m

# popup
m %>% addPopups(-93.65, 42.0285, 'Here is the <b>Department of Statistics</b>, ISU')
rnd_lng = function(n = 10) rnorm(n, -93.65, .01)
rnd_lat = function(n = 10) rnorm(n, 42.0285, .01)
# use automatic bounds derived from lng/lat data
m = m %>% clearBounds()

# popup
m %>% addPopups(rand_lng(), rand_lat(), 'Random popups')

# marker
m %>% addMarkers(rand_lng(), rand_lat())
m %>% addMarkers(
  rand_lng(), rand_lat(), popup = paste('A random letter', sample(LETTERS, 10))
)

Rlogo = file.path(R.home('doc'), 'html', 'logo.jpg')
m %>% addMarkers(
  174.7690922, -36.8523071, icon = list(
    iconUrl = Rlogo, iconSize = c(100, 76)
  ), popup = 'R was born here!'
)

m %>% addMarkers(rnorm(30, 175), rnorm(30, -37), icon = list(
  iconUrl = Rlogo, iconSize = c(25, 19)
))

m %>% addMarkers(
  c(-71.0382679, -122.1217866), c(42.3489054, 47.6763144), icon = list(
    iconUrl = 'http://www.rstudio.com/wp-content/uploads/2014/03/blue-125.png'
  ), popup = c('RStudio @ Boston', 'RStudio @ Seattle')
)

# circle (units in metres)
m %>% addCircles(rand_lng(50), rand_lat(50), radius = runif(50, 50, 150))

# circle marker (units in pixels)
m %>% addCircleMarkers(rand_lng(50), rand_lat(50), color = '#ff0000')
m %>% addCircleMarkers(rand_lng(100), rand_lat(100), radius = runif(100, 5, 15))

# rectangle
m %>% addRectangles(
  rand_lng(), rand_lat(), rand_lng(), rand_lat(),
  color = 'red', fill = FALSE, dashArray = '5,5', weight = 3
)

# polyline
m %>% addPolylines(rand_lng(50), rand_lat(50))

# polygon
m %>% addPolygons(rand_lng(), rand_lat(), layerId = 'foo')

# geoJSON
seattle_geojson = list(
  type = "Feature",         
  properties = 
    list(                 
      name = 'RStudio', 
      address = '110+150 Summer St', 
      city = 'Boston', 
      state = 'Massachusetts', 
      zip = '02110'
    ),
  geometry = 
    list(                  
      type = "Point", 
      coordinates = c(-71.0382679, -122.1217866)
    )
)
```
geometry = list(
    type = "MultiPolygon",
    coordinates = list(list(list(
        c(-122.36875812146, 47.6759920119894),
        c(-122.368781646764, 47.6668890126755),
        c(-122.360782108665, 47.6614990696722),
        c(-122.366199035722, 47.6614990696722),
        c(-122.366199035722, 47.6592874248973),
        c(-122.364582509469, 47.6576254522105),
        c(-122.363887331445, 47.6569107302038),
        c(-122.368865528129, 47.6538418253251),
        c(-122.368866157644, 47.6535254473167),
        c(-122.368866581103, 47.6533126275176),
        c(-122.362526540691, 47.6541872926348),
        c(-122.364442114483, 47.6551892850798),
        c(-122.366077719797, 47.6560733960066),
        c(-122.368818463838, 47.6579742346694),
        c(-122.376115159943, 47.6588730880334),
        c(-122.372295967029, 47.6604350102328),
        c(-122.37381369088, 47.660582362063),
        c(-122.375522972109, 47.6668413027949),
        c(-122.376079703095, 47.6608793094619),
        c(-122.376206315662, 47.6609242364243),
        c(-122.377610811371, 47.6606160735197),
        c(-122.379857378879, 47.6610306942278),
        c(-122.382454873022, 47.6627496239169),
        c(-122.385357955057, 47.6638573778241),
        c(-122.386007328104, 47.6640865692306),
        c(-122.387186331506, 47.6654326177161),
        c(-122.387802656231, 47.6661492860294),
        c(-122.388108244121, 47.6664548739202),
        c(-122.389177800763, 47.6663784774359),
        c(-122.390582858689, 47.6665072251861),
        c(-122.390973942299, 47.6659699214511),
        c(-122.391507906234, 47.6659209462299),
        c(-122.392883050767, 47.6664166747017),
        c(-122.392847210144, 47.6678696739431),
        c(-122.392904778401, 47.6709016021624),
        c(-122.39296705153, 47.6732047491624),
        c(-122.393008803496, 47.6759322346303),
        c(-122.396669453065, 47.6759896306663),
        c(-122.376486363943, 47.6759891899754),
        c(-122.366078869215, 47.6759641734893),
        c(-122.36075812146, 47.6759920119894)
    )))
),
properties = list(
    name = "Ballard",
    population = 48000,
    # You can inline styles if you want
    style = list(
        fillColor = "yellow",
        weight = 2,
    )
)```
color = "#000000"
)
",
id = "ballard"
)
m %>% setView(-122.36075812146, 47.6759920119894, zoom = 13) %>% addGeoJSON(seattle_geojson)

# use the Dark Matter layer from CartoDB
leaflet() %>% addTiles('http://s2.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}.png',
  attribution = paste(
    '©, <a href="http://openstreetmap.org">OpenStreetMap</a> contributors',
    '©, <a href="http://cartodb.com/attributions">CartoDB</a>
  )
) %>% setView(-122.36, 47.67, zoom = 10)

# provide a data frame to leaflet()
categories = LETTERS[1:10]
df = data.frame(
  lat = rand_lat(100), lng = rand_lng(100), size = runif(100, 5, 20),
  category = factor(sample(categories, 100, replace = TRUE), levels = categories),
  value = rnorm(100)
)m = leaflet(df) %>% addTiles()
m %>% addCircleMarkers(~lng, ~lat, radius = ~size)
m %>% addCircleMarkers(~lng, ~lat, radius = runif(100, 4, 10), color = c('red'))

# Discrete colors using the "RdY1Bu" colorbrewer palette, mapped to categories
RdY1Bu = colorFactor("RdY1Bu", domain = categories)
m %>% addCircleMarkers(~lng, ~lat, radius = ~size,
  color = ~RdY1Bu(category), fillOpacity = 0.5)

# Continuous colors using the "Greens" colorbrewer palette, mapped to value
greens = colorNumeric("Greens", domain = NULL)
m %>% addCircleMarkers(~lng, ~lat, radius = ~size,
  color = ~greens(value), fillOpacity = 0.5)

---

## Objects imported from other packages

**leaflet-imports**

### Description

These objects are imported from other packages. Follow the links to their documentation.

- **htmlwidgets** JS
- **magrittr** %>%
leafletDependencies

**Various leaflet dependency functions for use in downstream packages**

**Description**
Various leaflet dependency functions for use in downstream packages

**Usage**

```r
leafletDependencies
```

**Format**
An object of class `list` of length 13.

**Examples**
```r
## Not run:
addBootstrap <- function(map) {
    map$dependencies <- c(map$dependencies, leafletDependencies$bootstrap())
    map
}
## End(Not run)
```

leafletOutput

**Wrapper functions for using leaflet in shiny**

**Description**
Use `leafletOutput()` to create a UI element, and `renderLeaflet()` to render the map widget.

**Usage**

```r
leafletOutput(outputId, width = "100%", height = 400)
renderLeaflet(expr, env = parent.frame(), quoted = FALSE)
```

**Arguments**

- `outputId`: output variable to read from
- `width, height`: the width and height of the map (see `shinyWidgetOutput`)
- `expr`: An expression that generates an HTML widget
- `env`: The environment in which to evaluate `expr`.
- `quoted`: Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.
Examples

```r
# !formatR
library(leaflet)
library(shiny)
app = shinyApp(
    ui = fluidPage(leafletOutput('myMap')),
    server = function(input, output) {
        map = leaflet() %>% addTiles() %>% setView(-93.65, 42.0285, zoom = 17)
        output$myMap = renderLeaflet(map)
    }
)

if (interactive()) print(app)
```

---

**leafletProxy**  
*Send commands to a Leaflet instance in a Shiny app*

**Description**

Creates a map-like object that can be used to customize and control a map that has already been rendered. For use in Shiny apps and Shiny docs only.

**Usage**

`leafletProxy(mapId, session = shiny::get_default_ReactiveDomain(),
  data = NULL, deferUntilFlush = TRUE)`

**Arguments**

- `mapId` single-element character vector indicating the output ID of the map to modify (if invoked from a Shiny module, the namespace will be added automatically)
- `session` the Shiny session object to which the map belongs; usually the default value will suffice
- `data` a data object; see Details under the `leaflet` help topic
- `deferUntilFlush` indicates whether actions performed against this instance should be carried out right away, or whether they should be held until after the next time all of the outputs are updated; defaults to `TRUE`

**Details**

Normally, you create a Leaflet map using the `leaflet` function. This creates an in-memory representation of a map that you can customize using functions like `addPolygons` and `setView`. Such a map can be printed at the R console, included in an R Markdown document, or rendered as a Shiny output.
In the case of Shiny, you may want to further customize a map, even after it is rendered to an output. At this point, the in-memory representation of the map is long gone, and the user’s web browser has already realized the Leaflet map instance.

This is where `leafletProxy` comes in. It returns an object that can stand in for the usual Leaflet map object. The usual map functions like `addPolygons` and `setView` can be called, and instead of customizing an in-memory representation, these commands will execute on the live Leaflet map instance.

**Examples**

```r
library(shiny)

ui <- fluidPage(
  leafletOutput("map1")
)

server <- function(input, output, session) {
  output$map1 <- renderLeaflet(
    leaflet() %>% addCircleMarkers(
      lng = runif(10),
      lat = runif(10),
      layerId = paste0("marker", 1:10)
    )
  )

  observeEvent(input$map1_marker_click, {
    leafletProxy("map1", session) %>%
      removeMarker(input$map1_marker_click$id)
  })
}

shinyApp(ui, server)
```

---

```r
makeAwesomeIcon

**Description**

Make Awesome Icon

**Usage**

`makeAwesomeIcon(icon = "home", library = "glyphicon", markerColor = "blue", iconColor = "white", spin = FALSE, extraClasses = NULL, squareMarker = FALSE, iconRotate = 0, fontFamily = "monospace", text = NULL)`
**Arguments**

- **icon**: Name of the icon
- **library**: Which icon library. Default 'glyphicon', other possible values are 'fa' (fontawesome) or 'ion' (ionicons).
- **markercolor**: Possible values are 'red', 'darkred', 'lightred', 'orange', 'beige', 'green', 'darkgreen', 'lightgreen', 'blue', 'darkblue', 'lightblue', 'purple', 'darkpurple', 'pink', 'cadetblue', 'white', 'gray', 'lightgray', 'black'
- **iconColor**: The color to use for the icon itself. Use any CSS-valid color (hex, rgba, etc.) or a named web color.
- **spin**: If TRUE, make the icon spin (only works when library = 'fa')
- **extraclasses**: Additional css classes to include on the icon.
- **squareMarker**: Whether to use a square marker.
- **iconRotate**: Rotate the icon by a given angle.
- **fontFamily**: Used when text option is specified.
- **text**: Use this text string instead of an icon. argument of `addAwesomeMarkers()`.

**Description**

Define icon sets

**Usage**

```javascript
makeIcon(iconUrl = NULL, iconRetinaUrl = NULL, iconWidth = NULL, iconHeight = NULL, iconAnchorX = NULL, iconAnchorY = NULL, shadowUrl = NULL, shadowRetinaUrl = NULL, shadowWidth = NULL, shadowHeight = NULL, shadowAnchorX = NULL, shadowAnchorY = NULL, popupAnchorX = NULL, popupAnchorY = NULL, className = NULL)
```

**Arguments**

- **iconUrl**: the URL or file path to the icon image
- **iconRetinaUrl**: the URL or file path to a retina sized version of the icon image
- **iconWidth**: size of the icon image in pixels
- **iconHeight**: size of the icon image in pixels
- **iconAnchorX**: the coordinates of the "tip" of the icon (relative to its top left corner, i.e. the top left corner means iconAnchorX = 0 and iconAnchorY = 0), and the icon will be aligned so that this point is at the marker's geographical location
- **iconAnchorY**: the coordinates of the "tip" of the icon (relative to its top left corner, i.e. the top left corner means iconAnchorX = 0 and iconAnchorY = 0), and the icon will be aligned so that this point is at the marker's geographical location
mapOptions

- shadowUrl: the URL or file path to the icon shadow image
- shadowRetinaUrl: the URL or file path to the retina sized version of the icon shadow image
- shadowWidth: size of the shadow image in pixels
- shadowHeight: size of the shadow image in pixels
- shadowAnchorX: the coordinates of the "tip" of the shadow
- shadowAnchorY: the coordinates of the "tip" of the shadow
- popupAnchorX: the coordinates of the point from which popups will "open", relative to the icon anchor
- popupAnchorY: the coordinates of the point from which popups will "open", relative to the icon anchor
- className: a custom class name to assign to both icon and shadow images

**Description**

Set options on a leaflet map object

**Usage**

```r
mapOptions(map, zoomToLimits = c("always", "first", "never"))
```

**Arguments**

- `map`: A map widget object created from `leaflet()`
- `zoomToLimits`: Controls whether the map is zooms to the limits of the elements on the map. This is useful for interactive applications where the map data is updated. If "always" (the default), the map always re-zooms when new data is received; if "first", it zooms to the elements on the first rendering, but does not re-zoom for subsequent data; if "never", it never re-zooms, not even for the first rendering.

**Examples**

```r
# Don't auto-zoom to the objects (can be useful in interactive applications)
leaflet() %>%
  addTiles() %>%
  addPopups(174.7690922, -36.8523071, 'R was born here!') %>%
  mapOptions(zoomToLimits = "first")
```
previewColors  

**Description**

Color previewing utility

**Usage**

`previewColors(pal, values)`

**Arguments**

- `pal`  
  A color mapping function, like those returned from `colorNumeric`, et al

- `values`  
  A set of values to preview colors for

**Value**

An HTML-based list of the colors and values

---

providers  

**Description**

List of all providers with their variations

**Usage**

`providers`

**Format**

A list of characters

**Source**

**providers.details**  
Providers Details

**Description**  
List of all providers with their variations and additional info

**Usage**  
providers.details

**Format**  
A list of lists (JSON)

**Source**  
https://github.com/leaflet-extras/leaflet-providers/blob/master/leaflet-providers.js

---

**removeControl**  
Remove elements from a map

**Description**  
Remove one or more features from a map, identified by layerId; or, clear all features of the given type or group.

**Usage**  
removeControl(map, layerId)
clearControls(map)
clearGroup(map, group)
removeImage(map, layerId)
clearImages(map)
removeTiles(map, layerId)
clearTiles(map)
removePopup(map, layerId)
removeControl

clearPopups(map)
removeMarker(map, layerId)
clearMarkers(map)
removeMarkerCluster(map, layerId)
clearMarkerClusters(map)
removeMarkerFromCluster(map, layerId, clusterId)
removeShape(map, layerId)
clearShapes(map)
removeGeoJSON(map, layerId)
clearGeoJSON(map)
removeMeasure(map)
removeTopoJSON(map, layerId)
clearTopoJSON(map)

Arguments

map a map widget object, possibly created from leaflet() but more likely from leafletProxy()
layerId character vector; the layer id(s) of the item to remove
group the name of the group whose members should be removed
clusterId the id of the marker cluster layer

Value

the new map object

Note

When used with a leaflet() map object, these functions don’t actually remove the features from the map object, but simply add an operation that will cause those features to be removed after they are added. In other words, if you add a polygon "foo" and the call removeShape("foo"), it’s not smart enough to prevent the polygon from being added in the first place; instead, when the map is rendered, the polygon will be added and then removed.

For that reason, these functions aren’t that useful with leaflet map objects and are really intended to be used with leafletProxy instead.
WMS tile layers are extensions of tile layers, so they can also be removed or cleared via `removeTiles()` or `clearTiles()`.

---

**setView**  
*Methods to manipulate the map widget*

**Description**  
A series of methods to manipulate the map.

**Usage**
```r
setView(map, lng, lat, zoom, options = list())
fitBounds(map, lng1, lat1, lng2, lat2)
setMaxBounds(map, lng1, lat1, lng2, lat2)
clearBounds(map)
```

**Arguments**
- `map` a map widget object created from `leaflet()`
- `lng` The longitude of the map center
- `lat` The latitude of the map center
- `zoom` the zoom level
- `options` a list of zoom/pan options (see [http://leafletjs.com/reference.html#map-zoompanoptions](http://leafletjs.com/reference.html#map-zoompanoptions))
- `lng1`, `lat1`, `lng2`, `lat2` the coordinates of the map bounds

**Value**  
The modified map widget.

**Functions**
- `setView`: Set the view of the map (center and zoom level)
- `fitBounds`: Set the bounds of a map
- `setMaxBounds`: Restricts the map view to the given bounds
- `clearBounds`: Clear the bounds of a map, and the bounds will be automatically determined from latitudes and longitudes of the map elements if available (otherwise the full world view is used)

**References**
Examples

library(leaflet)
m = leaflet() %>% addTiles() %>% setView(-71.0382579, 42.3489054, zoom = 18)
m # the RStudio 'headquarter'
m %>% fitBounds(-72, 40, -70, 43)
m %>% clearBounds() # world view

showGroup

Show or hide layer groups

Description

Hide groups of layers without removing them from the map entirely. Groups are created using the group parameter that is included on most layer adding functions.

Usage

showGroup(map, group)

hideGroup(map, group)

Arguments

map the map to modify
group character vector of one or more group names to show or hide

See Also

addLayersControl to allow users to show/hide layer groups interactively

tileOptions Extra options for map elements and layers

Description

The rest of all possible options for map elements and layers that are not listed in the layer functions.
Usage

```javascript
tileOptions(minZoom = 0, maxZoom = 18, maxNativeZoom = NULL, 
tileSize = 256, subdomains = "abc", errorTileUrl = ", tms = FALSE, 
continuousWorld = FALSE, noWrap = FALSE, zoomOffset = 0, 
zoomReverse = FALSE, opacity = 1, zIndex = NULL, 
unloadInvisibleTiles = NULL, updateWhenIdle = NULL, 
detectRetina = FALSE, reuseTiles = FALSE, ...)

WMSTileOptions(styles = ", format = "image/jpeg", transparent = FALSE, 
version = "1.1.1", crs = NULL, ...)

popupOptions(maxWidth = 300, minWidth = 50, maxHeight = NULL, 
autoPan = TRUE, keepInView = FALSE, closeButton = TRUE, 
zoomAnimation = TRUE, closeOnClick = NULL, className = ", ...)

labelOptions(clickable = FALSE, noHide = FALSE, className = ", 
direction = "right", offset = c(12, -15), opacity = 1, 
textSize = "10px", textOnly = FALSE, style = NULL, 
zoomAnimation = TRUE, ...)

markerOptions(clickable = TRUE, draggable = FALSE, keyboard = TRUE, 
title = ", alt = ", zIndexOffset = 0, opacity = 1, 
riseOnHover = FALSE, riseOffset = 250, ...)

markerClusterOptions(showCoverageOnHover = TRUE, zoomToBoundsOnClick = TRUE, 
spiderfyOnMaxZoom = TRUE, removeOutsideVisibleBounds = TRUE, 
spiderLegPolylineOptions = list(weight = 1.5, color = "#222", opacity = 
0.5), freezeAtZoom = FALSE, ...)

pathOptions(lineCap = NULL, lineJoin = NULL, clickable = TRUE, 
pointerEvents = NULL, className = ", ...)
```

Arguments

- `minZoom`, `maxZoom`, `maxNativeZoom`, `tileSize`, `subdomains`, `errorTileUrl`, `tms`, `continuousWorld`, `noWrap`, `zoomOffset`, `zoomReverse`, `opacity`, `zIndex`, `unloadInvisibleTiles`, `updateWhenIdle`, `detectRetina`, `reuseTiles` are the tile layer options; see [http://leafletjs.com/reference.html#tilelayer](http://leafletjs.com/reference.html#tilelayer)

- `styles` is a comma-separated list of WMS styles

- `format` is the WMS image format (use 'image/png' for layers with transparency)

- `transparent` is `true` if the WMS service will return images with transparency

- `version` is the version of the WMS service to use

- `crs` is the Coordinate Reference System to use for the WMS requests, defaults.

- `maxWidth`, `minWidth`, `maxHeight`, `autoPan`, `keepInView`, `closeButton`, `zoomAnimation`, `closeOnClick`, `popupOptions` are extra options passed to underlying Javascript object constructor.

- `className` is a CSS class name set on an element

- `clickable` is whether the element emits mouse events
tileOptions

noHide, direction, offset, textsize, textOnly, style

label options; see https://github.com/Leaflet/Leaflet.label#options
draggable, keyboard, title, alt, zIndexOffset, opacity, riseOnHover, riseOffset

marker options; see http://leafletjs.com/reference.html#marker

showCoverageOnHover
when you mouse over a cluster it shows the bounds of its markers

zoomToBoundsOnClick
when you click a cluster we zoom to its bounds

spiderfyOnMaxZoom
when you click a cluster at the bottom zoom level we spiderfy it so you can see all of its markers

removeOutsideVisibleBounds
clusters and markers too far from the viewport are removed from the map for performance

spiderLegPolylineOptions
Allows you to specify PolylineOptions (http://leafletjs.com/reference.html#polyline-options) to style spider legs. By default, they are weight: 1.5, color: '#222', opacity: 0.5

freezeAtZoom
Allows you to freeze cluster expansion to a zoom level. Can be a zoom level e.g. 10, 12 or 'max' or 'maxKeepSpiderify'. See https://github.com/ghybs/Leaflet.MarkerCluster.Freezable#api-reference

lineCap
a string that defines shape to be used at the end of the stroke

lineJoin
a string that defines shape to be used at the corners of the stroke

pointerEvents
sets the pointer-events attribute on the path if SVG backend is used

Functions

- tileOptions: Options for tile layers
- WMSTileOptions: Options for WMS tile layers
- popupOptions: Options for popups
- labelOptions: Options for labels
- markerOptions: Options for markers
- markerClusterOptions: Options for marker clusters
- pathOptions: Options for vector layers (polylines, polygons, rectangles, and circles, etc)

See Also

leafletCRS to map CRS (don’t change this if you’re not sure what it means)
validateCoords

Utility function to check if a coordinates is valid

**Description**
Utility function to check if a coordinates is valid

**Usage**
validateCoords(lng, lat, funcName, warn = T)

**Arguments**
- **lng**: vector with longitude values
- **lat**: vector with latitude values
- **funcName**: Name of calling function
- **warn**: A boolean. Whether to generate a warning message if there are rows with missing/invalid data
Index

Topic datasets
  leafletDependencies, 40
  providers, 45
  providers.details, 46
%>(leaflet-imports), 39
%>%, 39

addAwesomeMarkers, 3, 22, 43
addCircleMarkers (addControl), 4
addCircles (addControl), 4
addControl, 4
addEasyButton, 28
addEasyButton (easyButtonState), 27
addEasyButtonBar (easyButtonState), 27
addGeoJSON (addControl), 4
addGraticule, 8
addLabelOnlyMarkers (addControl), 4
addLayersControl, 4, 6, 9, 16, 49
addLegend, 10
addMarkers, 33
addMarkers (addControl), 4
addMeasure, 12
addMiniMap, 14
addPolygons, 41, 42
addPolygons (addControl), 4
addPolylines (addControl), 4
addPopups (addControl), 4
addProviderTiles, 15
addRasterImage, 16
addRectangles (addControl), 4
addScaleBar, 18
addSimpleGraticule, 19
add Terminator, 20
addTiles, 17
addTiles (addControl), 4
addTopoJSON (addControl), 4
addWMSTiles (addControl), 4
atlStorms2005, 20
awesomeIconList, 21
awesomeIcons, 21
breweries91, 22
clearBounds (setView), 48
clearControls (removeControl), 46
clearGeoJSON (removeControl), 46
clearGroup, 4, 6, 16
clearGroup (removeControl), 46
clearImages (removeControl), 46
clearMarkerClusters (removeControl), 46
clearMarkers (removeControl), 46
clearPopups (removeControl), 46
clearShapes (removeControl), 46
clearTiles (removeControl), 46
clearTopoJSON (removeControl), 46
colorBin (colorNumeric), 23
colorFactor (colorNumeric), 23
colorNumeric, 10, 17, 23, 45
colorQuantile (colorNumeric), 23
createLeafletMap, 25
cut, 24
derivePoints, 25
derivePolygons, 26
dispatch, 27
easyButton, 28
easyButton (easyButtonState), 27
easyButtonState, 27
evalFormula, 29
expandLimits, 29
expandLimitsBbox, 30
filterNULL, 30
fitBounds (setView), 48
gadmCHE, 31
getData, 31
gGetMapData, 31
hideGroup (showGroup), 49
highlightOptions (addControl), 4
htmlEscape, 4, 6
iconList, 32
icons, 6, 32
invokeMethod (dispatch), 27
JS, 39
JS (leaflet-imports), 39
labelFormat (addLegend), 10
labelOptions, 4, 7
labelOptions (tileOptions), 49
layersControlOptions
(addLayersControl), 9
leaflet, 6, 10, 27, 34, 41, 44, 47, 48
leaflet-imports, 39
leafletCRS, 36, 51
leafletCRS (leaflet), 34
leafletDependencies, 40
leafletMap (createLeafletMap), 25
leafletOptions (leaflet), 34
leafletOutput, 40
leafletProxy, 27, 41, 47
makeAwesomeIcon, 21, 42
makeIcon, 32, 43
mapOptions, 44
markerClusterOptions, 4, 7
markerClusterOptions (tileOptions), 49
markerOptions, 8
markerOptions (tileOptions), 49
pathOptions, 8
pathOptions (tileOptions), 49
popupOptions, 4, 7, 8
popupOptions (tileOptions), 49
pretty, 23
previewColors, 45
projectRaster, 17
projectRasterForLeaflet (addRasterImage), 16
providers, 15, 45
providers.details, 46
providerTileOptions (addProviderTiles), 15
quantile, 24
raster, 17
remove (removeControl), 46
removeControl, 46
removeGeoJSON (removeControl), 46
removeImage (removeControl), 46
removeLayersControl (addLayersControl), 9
removeMarker (removeControl), 46
removeMarkerCluster (removeControl), 46
removeMarkerFromCluster (removeControl), 46
removeMeasure (removeControl), 46
removePopup (removeControl), 46
removeScaleBar (addScaleBar), 18
removeShape (removeControl), 46
removeTiles (removeControl), 46
removeTopoJSON (removeControl), 46
renderLeaflet (leafletOutput), 40
scaleBarOptions (addScaleBar), 18
seq, 23
setMaxBounds (setView), 48
setView, 41, 42, 48
shinyWidgetOutput, 40
showGroup, 49
tileOptions, 8, 49
validateCoords, 52
WMSTileOptions, 8
WMSTileOptions (tileOptions), 49