Package: leaflet (via r-universe)

September 6, 2024

Type Package

Title Create Interactive Web Maps with the JavaScript 'Leaflet' Library

Version 2.2.2.9000

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 applications and R Markdown documents.

License GPL-3

URL https://rstudio.github.io/leaflet/, https://github.com/rstudio/leaflet

BugReports https://github.com/rstudio/leaflet/issues

Depends R (>= 3.1.0)

Imports crosstalk, htmltools, htmlwidgets (>= 1.5.4), jquerylib, leaflet.providers (>= 2.0.0), magrittr, methods, png, raster (>= 3.6.3), RColorBrewer, scales (>= 1.0.0), sp, stats, viridisLite, xfun

Suggests knitr, maps, purrr, R6, RJSONIO, rmarkdown, s2, sf (>= 0.9-6), shiny, terra, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/Needs/website dplyr, ncdf4, rnaturalearth, tidyverse/tidytemplate

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/rstudio/leaflet

RemoteRef HEAD

RemoteSha 9bf137b8f58d62a199e1eceb906f5b705030b75a

Contents

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
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.
icon	the icon(s) for markers;
popup	a character vector of the HTML content for the popups (you are recommended to escape the text using htmltools::htmlEscape() for security reasons)
popupOptions	A Vector of popupOptions() to provide popups

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`
options	a list of extra options for tile layers, popups, paths (circles, rectangles, polygons, ...), or other map elements
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
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

---

`addControl` *Graphics elements and layers*

Description

Add graphics elements and layers to the map widget.

Usage

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

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

addWMSTiles(
  map,
  baseUrl,
  layerId = NULL,
  group = NULL,
  options = WMSTileOptions(),
```

```
    attribution = NULL,
    layers = "",
    data = getMapData(map)
  )

  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,
    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(),
  data = getMapData(map)
)

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 `htmltools::htmlEscape()` for security reasons)

`icon` the icon(s) for markers; an icon is represented by an R list of the form `list(iconUrl = "?", iconSize = c(x, y))`, 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

`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: https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html

See Also

tileOptions(), WMSTileOptions(), popupOptions(), markerOptions(), pathOptions()

addGraticule *Add a Graticule on the map*

Description

Add a Graticule on the map

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 https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#path-option
layerId	the layer id
group	the name of the group this layer belongs to.
options	the path options for the graticule layer

See Also

https://github.com/turban/Leaflet.Graticule

Examples

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

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

```
addLayersControl(
  map,
  baseGroups = character(0),
  overlayGroups = character(0),
  position = c("topright", "bottomright", "bottomleft", "topleft"),
  options = layersControlOptions(),
  data = getMapData(map)
)

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

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()`
`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
`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()`

Examples

```
leaflet() %>%
  addTiles(group = "OpenStreetMap") %>%
  addProviderTiles("CartoDB.Voyager", group = "Carto Voyager") %>%
  addMarkers(runif(20, -75, -74), runif(20, 41, 42), group = "Markers") %>%
  addLayersControl(
    baseGroups = c("OpenStreetMap", "Carto Voyager"),
    overlayGroups = c("Markers")
  )
```

---

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 = NULL,
  labFormat = labelFormat(),
  title = NULL,
  className = "info legend",
  layerId = NULL,
  group = NULL,
  data = getMapData(map)
)

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
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 labelFormat() returns by default; you can either use the helper function labelFormat(), 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().
group	group name of a leaflet layer group. Supplying this value will tie the legend to the leaflet layer group with this name and will auto add/remove the legend as the group is added/removed, for example via layerControl(). You will need to set the group when you add a layer (e.g., addPolygons()) and supply the same name here.
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
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
  data.frame(x, y, z)
})
pal <- colorNumeric("OrRd", df$z)
leaflet(df) %>%
  addTiles() %>%
  addCircleMarkers(~x, ~y, color = ~pal(z), group = "circles") %>%
  addLegend(pal = pal, values = ~z, group = "circles", position = "bottomleft") %>%
  addLayersControl(overlayGroups = c("circles"))

# 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) %>%
  addTiles() %>%
  addCircleMarkers(~x, ~y, color = ~pal(z), group = "circles") %>%
  addLegend(pal = pal, values = ~z, group = "circles", position = "bottomleft") %>%
  addLayersControl(overlayGroups = c("circles"))
```

```
leaflet(df) %>%
  addTiles() %>%
  addCircleMarkers(~x, ~y, color = ~pal(z), group = "circles") %>%
  addLegend(pal = pal, values = ~z, labFormat = labelFormat(
    prefix = "(", suffix = ")%", between = ", ",
    transform = function(x) 100 * x
  ),  group = "circles", position = "bottomleft" ) %>%
  addLayersControl(overlayGroups = c("circles"))
```

---

addMapPane *Add additional panes to leaflet map to control layer order*

Description

map panes can be created by supplying a name and a zIndex to control layer ordering. We recommend a `zIndex` value between 400 (the default overlay pane) and 500 (the default shadow pane). You can then use this pane to render overlays (points, lines, polygons) by setting the `pane` argument in `leafletOptions()`. This will give you control over the order of the layers, e.g., points always on top of polygons. If two layers are provided to the same pane, overlay will be determined by order of adding. See examples below. See https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#map-pane for details.

If the error "Cannot read property 'appendChild' of undefined" occurs, make sure the pane being used for used for display has already been added to the map.

Usage

```
addMapPane(map, name, zIndex)
```

Arguments

`map`	A `leaflet` or `mapview` object.
`name`	The name of the new pane (refer to this in `leafletOptions()`.
`zIndex`	The zIndex of the pane. Panes with higher index are rendered above panes with lower indices.

Examples

```
rand_lng <- function(n = 10) rnorm(n, -93.65, .01)
rand_lat <- function(n = 10) rnorm(n, 42.0285, .01)

random_data <- data.frame(
  lng = rand_lng(50),
  lat = rand_lat(50),
  radius = runif(50, 50, 150),
  circleId = paste0("circle #", 1:50),
  lineId = paste0("circle #", 1:50)
)
```

```
# display circles (zIndex: 420) above the lines (zIndex: 410), even when added first
leaflet() %>%
  addTiles() %>%
  # move the center to Snedecor Hall
  setView(-93.65, 42.0285, zoom = 14) %>%
  addMapPane("ames_lines", zIndex = 410) %>% # shown below ames_circles
  addMapPane("ames_circles", zIndex = 420) %>% # shown above ames_lines
  # points above polygons
  addCircles(
    data = random_data, ~lng, ~lat, radius = ~radius, popup = ~circleId,
    options = pathOptions(pane = "ames_circles")
  ) %>%
  # lines in 'ames_lines' pane
  addPolylines(
    data = random_data, ~lng, ~lat, color = "#F00", weight = 20,
    options = pathOptions(pane = "ames_lines")
  )


# same example but circles (zIndex: 420) are below the lines (zIndex: 430)
leaflet() %>%
  addTiles() %>%
  # move the center to Snedecor Hall
  setView(-93.65, 42.0285, zoom = 14) %>%
  addMapPane("ames_lines", zIndex = 430) %>% # shown below ames_circles
  addMapPane("ames_circles", zIndex = 420) %>% # shown above ames_lines
  # points above polygons
  addCircles(
    data = random_data, ~lng, ~lat, radius = ~radius, popup = ~circleId,
    options = pathOptions(pane = "ames_circles")
  ) %>%
  # lines in 'ames_lines' pane
  addPolylines(
    data = random_data, ~lng, ~lat, color = "#F00", weight = 20,
    options = pathOptions(pane = "ames_lines")
  )
```

---

addMeasure *Add a measure control to the map.*

Description

Add a measure control to the map.

Usage

```
addMeasure(
  map,
```

```
  position = "topright",
  primaryLengthUnit = "feet",
  secondaryLengthUnit = NULL,
  primaryAreaUnit = "acres",
  secondaryAreaUnit = NULL,
  activeColor = "#ABE67E",
  completedColor = "#C8F2BE",
 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

```
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*

Description

Add a minimap to the map

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 LatLng-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 minimize 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 (https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.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 (https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#path-option) 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

https://github.com/Norkart/Leaflet-MiniMap

providers()

Examples

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

---

addProviderTiles *Add a tile layer from a known map provider*

Description

Add a tile layer from a known map provider

Usage

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

providerTileOptions(
  errorTileUrl = "",
  noWrap = FALSE,
  opacity = NULL,
  zIndex = NULL,
  updateWhenIdle = NULL,
  detectRetina = FALSE,
  ...
)
```

Arguments

map	the map to add the tile layer to
provider	the name of the provider (see https://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
check	Check that the specified provider matches the available currently loaded leaflet providers? Defaults to TRUE, but can be toggled to FALSE for advanced users.
errorTileUrl, noWrap, opacity, zIndex, updateWhenIdle, detectRetina	the tile layer options; see https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#tilelayer
...	named parameters to add to the options

Value

modified map object

Examples

```
leaflet() %>%
  addProviderTiles("Esri.WorldTopoMap") %>%
  addProviderTiles("CartoDB.DarkMatter")
```

---

addRasterImage *Add a raster image as a layer*

Description

Create an image overlay from a RasterLayer or a SpatRaster 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

```
addRasterImage(
  map,
  x,
  colors = if (is.factor(x)[1]) "Set1" else "Spectral",
  opacity = 1,
  attribution = NULL,
  layerId = NULL,
```

```
  group = NULL,
  project = TRUE,
  method = c("auto", "bilinear", "ngb"),
  maxBytes = 4 * 1024 * 1024,
  options = gridOptions(),
  data = getMapData(map)
)

projectRasterForLeaflet(x, method)
```

Arguments

map	a map widget object
x	a terra::SpatRaster() or a RasterLayer object–see raster::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). The palette is ignored if x is a SpatRaster with a color table or if it has RGB channels.
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
method	the method used for computing values of the new, projected raster image. "bilinear" (the default) is appropriate for continuous data, "ngb" - nearest neighbor - is appropriate for categorical data. Ignored if project = FALSE. See projectRaster() for details.
maxBytes	the maximum number of bytes to allow for the projected image (before base64 encoding); defaults to 4MB.
options	a list of additional options, intended to be provided by a call to gridOptions()
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

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.

To reduce the size of a SpatRaster, you can use `terra::spatSample()` as in `x = spatSample(x, 100000, method="regular", as.raster=TRUE)`. With a `RasterLayer` you can use `raster::sampleRegular()` as in `sampleRegular(x, 100000, asRaster=TRUE)`.

By default, `addRasterImage()` will project the raster data `x` to the Pseudo-Mercator projection (EPSG:3857). This can be a time-consuming operation for even moderately sized rasters; although it is much faster for SpatRasters than for RasterLayers. 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`.

See Also

`addRasterLegend()` for an easy way to add a legend for a SpatRaster with a color table.

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")

pal <- colorNumeric("Spectral", domain = c(0, 1000))
leaflet() %>% addTiles() %>%
  addRasterImage(r, colors = pal, opacity = 0.8) %>%
  addLegend(pal = pal, values = c(0, 1000))
```

---

addRasterLegend *Add a color legend for a SpatRaster to a map*

Description

A function for adding a legend that is specifically designed for terra::SpatRaster objects, with categorical values, that carry their own color table.

Usage

```
addRasterLegend(map, x, layer = 1, ...)
```

Arguments

`map`	a map widget object
`x`	a SpatRaster object with a color table
`layer`	the layer of the raster to target
`...`	additional arguments to pass through to `addLegend()`

See Also

addRasterImage()

Examples

```
library(terra)

r <- rast("/vsicurl/https://geodata.ucdavis.edu/test/pr_nlcd.tif")
leaflet() %>%
  addTiles() %>%
  addRasterImage(r, opacity = 0.75) %>%
  addRasterLegend(r, opacity = 0.75)

plot.new() # pause in interactive mode

rr <- r
levels(rr)  <- NULL
leaflet() %>%
  addTiles() %>%
  addRasterImage(rr, opacity = 0.75) %>%
  addRasterLegend(rr, opacity = 0.75)
```

---

addScaleBar *Add or remove a scale bar*

Description

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

Usage

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

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

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

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

---

addSimpleGraticule *Add a simple Graticule on the map*

Description

Add a simple Graticule on the map

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.

See Also

https://github.com/ablakey/Leaflet.SimpleGraticule

Examples

```
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

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

---

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

See Also

Other built in datasets: breweries91, gadmCHE

---

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*

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 recycled. `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`

See Also

https://github.com/lennardv2/Leaflet.awesome-markers

---

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"

See Also

Other built in datasets: atlStorms2005, gadmCHE

---

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,
  right = FALSE
)

colorQuantile(
  palette,
  domain,
  n = 4,
  probs = seq(0, 1, length.out = n + 1),
  na.color = "#808080",
  alpha = FALSE,
  reverse = FALSE,
  right = 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".
right	parameter supplied to cut. See Details
n	Number of equal-size quantiles desired. For more precise control, use probs instead.
probs	See stats::quantile(). If provided, n 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 base::cut() function). colorBin() defaults for the base::cut() function are include.lowest = TRUE and right = FALSE.

colorQuantile() similarly bins numeric data, but via stats::quantile().

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: "magma", "inferno", "plasma", "viridis", "cividis", "rocket", "mako", or "turbo"
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

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

if (interactive()) {
  # 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.
  previewColors(colorFactor("RdYlBu", levels = LETTERS), LETTERS[1:5])
}
```

`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 `leafletProxy()`
`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

Usage

```
easyButtonState(stateName, icon, title, onClick)

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

addEasyButton(map, button)

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|toprightbottomleft|bottomright
id	id for the button
states	the states
map	a map widget object
button	the button object created with easyButton()
...	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

addEasyButton()

Examples

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

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

---

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

`map`	map object
`lat`	vector of latitudes
`lng`	vector of longitudes

`expandLimitsBbox` *Notifies the map of polygons of interest on the map*

Description

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

Usage

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

Arguments

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

`gadmCHE` *Administrative borders of Switzerland (level 1)*

Description

Administrative borders of Switzerland (level 1)

Format

`sp::SpatialPolygonsDataFrame`

Details

This dataset comes from `https://gadm.org`. It was downloaded using `getData()`.

Source

`https://gadm.org`

See Also

Other built in datasets: `atlStorms2005`, `breweries91`

getMapData *Extract the map's data*

Description

Extract the map's data

Usage

```
getMapData(map)
```

Arguments

map	the map

Value

The map's data

groupOptions *Set options on layer groups*

Description

Change options on layer groups. Currently the only option is to control what zoom levels a layer group will be displayed at. The `zoomLevels` option is not compatible with layers control; do not both assign a group to zoom levels and use it with `addLayersControl()`.

Usage

```
groupOptions(map, group, zoomLevels = NULL)
```

Arguments

map	the map to modify
group	character vector of one or more group names to set options on
zoomLevels	numeric vector of zoom levels at which group(s) should be visible, or `TRUE` to display at all zoom levels

Examples

```
pal <- colorQuantile("YlOrRd", quakes$mag)

leaflet() %>%
  # Basic markers
  addTiles(group = "basic") %>%
  addMarkers(data = quakes, group = "basic") %>%
  # When zoomed in, we'll show circles at the base of each marker whose
  # radius and color reflect the magnitude
  addProviderTiles(providers$Esri.WorldTopoMap, group = "detail") %>%
  addCircleMarkers(data = quakes, group = "detail", fillOpacity = 0.5,
    radius = ~mag * 5, color = ~pal(mag), stroke = FALSE) %>%
  # Set the detail group to only appear when zoomed in
  groupOptions("detail", zoomLevels = 7:18)
```

---

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
)
```

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

```
library(leaflet)

# adapted from https://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("https://leafletjs.com/examples/custom-icons/leaf-%s.png", group),
    shadowUrl = "https://leafletjs.com/examples/custom-icons/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[group],
    popupAnchorX = 20, popupAnchorY = 0
  ),
  popup = ~ sprintf(
    "lat = %.4f, long = %.4f, group = %s, pch = %s", lat, lng, group, shapes[group]
  )
)

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

`leaflet` *Create a Leaflet map widget*

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.

Usage

```
leaflet(
  data = NULL,
  width = NULL,
  height = NULL,
  padding = 0,
  options = leafletOptions(),
  elementId = NULL,
  sizingPolicy = leafletSizingPolicy(padding = padding)
)

leafletOptions(
```

```
  minZoom = NULL,
  maxZoom = NULL,
  crs = leafletCRS(),
  worldCopyJump = NULL,
  preferCanvas = 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 matrix, data frame, spatial data from the **sf** package, SpatVector from the **terra** package, and the Spatial* objects from the **sp** package that represent points, lines, or polygons.
width	the width of the map
height	the height of the map
padding	the padding of the map
options	the map options
elementId	Use an explicit element ID for the widget (rather than an automatically generated one).
sizingPolicy	htmlwidgets sizing policy object. Defaults to leafletSizingPolicy()
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.
preferCanvas	Whether leaflet.js Paths should be rendered on a Canvas renderer.
...	other options used for leaflet.js map creation.
crsClass	One of L.CRS.EPSG3857, L.CRS.EPSG4326, L.CRS.EPSG3395, L.CRS.Simple, L.Proj.CRS
code	CRS identifier

proj4def	Proj4 string
projectedBounds	DEPRECATED! Use the bounds argument.
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	DEPRECATED! Specify the tilesize in the `tileOptions()` argument.

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 `longtitude`, then we may add a circle layer to the map by `leaflet(data) %>% addCircles(lat = ~latitude, lng = ~longtitude)`, 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.

See https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#map-option for details and more options.

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")
rand_lng <- function(n = 10) rnorm(n, -93.65, .01)
rand_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)
))

# 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",
  geometry = list(
```

```
    type = "MultiPolygon",
    coordinates = list(list(list(
      c(-122.36075812146,  47.6759920119894),
      c(-122.360781646764, 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(-122.360865528129, 47.6538418253251),
      c(-122.360866157644,  47.6535254473167),
      c(-122.360866581103, 47.6533126275176),
      c(-122.362526540691,  47.6541872926348),
      c(-122.364442114483, 47.6551892850798),
      c(-122.366077719797,  47.6560733960606),
      c(-122.368818463838, 47.6579742346694),
      c(-122.370115159943,  47.6588730808334),
      c(-122.372295967029, 47.6604350102328),
      c(-122.37381369088,  47.660582362063),
      c(-122.375522972109, 47.6606413027949),
      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(-122.390793942299,  47.6659699214511),
      c(-122.391507906234, 47.6659200946229),
      c(-122.392883050767,  47.6664166747017),
      c(-122.392847210144, 47.6678696739431),
      c(-122.392904778401,  47.6709016021624),
      c(-122.39296705153, 47.6732047491624),
      c(-122.393000803496,  47.6759322346303),
      c(-122.37666945305, 47.6759896300663),
      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("https://{s}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}.png",
  attribution = paste(
    "&copy; <a href=\"https://openstreetmap.org\">OpenStreetMap</a> contributors",
    "&copy; <a href=\"https://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 "RdYlBu" colorbrewer palette, mapped to categories
RdYlBu <- colorFactor("RdYlBu", domain = categories)
m %>% addCircleMarkers(~lng, ~lat, radius = ~size,
  color = ~RdYlBu(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)
```

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

Description

Various leaflet dependency functions for use in downstream packages

Usage

```
leafletDependencies
```

Format

An object of class `list` of length 13.

Examples

```
## 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

```
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 `htmlwidgets::shinyWidgetOutput()`)
`expr`	An expression that generates an HTML widget (or a promise of 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

```
# !formatR
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()) 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::getDefaultReactiveDomain(),
  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 `leaflet()`. 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

```
library(shiny)

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

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

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

app <- shinyApp(ui, server)
if (interactive()) app
```

---

leafletSizingPolicy *Leaflet sizing policy*

Description

Sizing policy used withing leaflet htmlwidgets.

Usage

```
leafletSizingPolicy(
  defaultWidth = "100%",
  defaultHeight = 400,
  padding = 0,
  browser.fill = TRUE,
  ...
)
```

Arguments

defaultWidth	defaults to "100%" of the available width
defaultHeight	defaults to 400px tall
padding	defaults to 0px
browser.fill	defaults to TRUE

... Arguments passed on to `htmlwidgets::sizingPolicy`

`viewer.defaultWidth` The default width used to display the widget within the RStudio Viewer.

`viewer.defaultHeight` The default height used to display the widget within the RStudio Viewer.

`viewer.padding` Padding around the widget when displayed in the RStudio Viewer (defaults to 15 pixels).

`viewer.fill` When displayed in the RStudio Viewer, automatically size the widget to the viewer dimensions (note that `viewer.padding` is still applied). Default to `TRUE`.

`viewer.suppress` Never display the widget within the RStudio Viewer (useful for widgets that require a large amount of space for rendering). Defaults to `FALSE`.

`viewer.paneHeight` Request that the RStudio Viewer be forced to a specific height when displaying this widget.

`browser.defaultWidth` The default width used to display the widget within a standalone web browser.

`browser.defaultHeight` The default height used to display the widget within a standalone web browser.

`browser.padding` Padding around the widget when displayed in a standalone browser (defaults to 40 pixels).

`browser.external` When displaying in a browser, always use an external browser (via `browseURL()`). Defaults to 'FALSE", which will result in the use of an internal browser within RStudio v1.1 and higher.

`knitr.defaultWidth` The default width used to display the widget within documents generated by knitr (e.g. R Markdown).

`knitr.defaultHeight` The default height used to display the widget within documents generated by knitr (e.g. R Markdown).

`knitr.figure` Apply the default knitr fig.width and fig.height to the widget when it's rendered within R Markdown documents. Defaults to `TRUE`.

`fill` Whether or not the widget's container should be treated as a fill item, meaning that its `height` is allowed to grow/shrink to fit a fill container with an opinionated height (see `htmltools::bindFillRole()` for more). Examples of fill containers include `bslib::card()` and `bslib::card_body_fill()`.

Value

An `htmlwidgets::sizingPolicy` object

`makeAwesomeIcon` *Make Awesome Icon*

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().

---

makeIcon *Define icon sets*

Description

Define icon sets

Usage

```
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`, `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

mapOptions *Set options on a leaflet map object*

Description

Set options on a leaflet map object

Usage

```
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

```
# 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 *Color previewing utility*

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 *Providers*

Description

List of all providers with their variations

Format

A list of characters

Source

https://github.com/leaflet-extras/leaflet-providers/blob/0a9e27f8c6c26956b4e78c26e1945d748e3c2869/leaflet-providers.js

Examples

```
providers
```

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)
```

```
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

```
setView(map, lng, lat, zoom, options = list())

flyTo(map, lng, lat, zoom, options = list())

fitBounds(map, lng1, lat1, lng2, lat2, options = list())

flyToBounds(map, lng1, lat1, lng2, lat2, options = list())

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 https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#zoom/pan-options)
`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)
- `flyTo()`: Flys to a given location/zoom-level using smooth pan-zoom.
- `fitBounds()`: Set the bounds of a map
- `flyToBounds()`: Flys to given bound using smooth pan/zoom.

- `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

https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#map-methods-for-modifying-map-state

Examples

```
m <- leaflet() %>% addTiles() %>% setView(-71.0382679, 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

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

gridOptions(
  tileSize = 256,
  updateWhenIdle = NULL,
  zIndex = 1,
  minZoom = 0,
  maxZoom = NULL,
  ...
)

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 = NULL,
  closeOnClick = NULL,
  className = "",
  ...
)

labelOptions(
  interactive = FALSE,
  clickable = NULL,
  noHide = NULL,
  permanent = FALSE,
  className = "",
  direction = "auto",
  offset = c(0, 0),
  opacity = 1,
  textsize = "10px",
  textOnly = FALSE,
  style = NULL,
  zoomAnimation = NULL,
  sticky = TRUE,
  ...
)

markerOptions(
  interactive = TRUE,
  clickable = NULL,
  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 = NULL,
  interactive = TRUE,
  pointerEvents = NULL,
  className = "",
  ...
)
```

Arguments

`minZoom`, `maxZoom`, `maxNativeZoom`, `tileSize`, `subdomains`, `errorTileUrl`, `tms`, `noWrap`, `zoomOffset`, `zoomReverse`, `zIndex`, `unloadInvisibleTiles`, `updateWhenIdle`, `detectRetina`
: the tile layer options; see https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#tilelayer

`opacity`
: Tooltip container opacity. Ranges from 0 to 1. Default value is `1` (different from leaflet.js `0.9`); see https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#tooltip-opacity

`...`
: extra options passed to underlying JavaScript object constructor.

`styles`
: comma-separated list of WMS styles

`format`
: WMS image format (use `"image/png"` for layers with transparency)

`transparent`
: if `TRUE`, the WMS service will return images with transparency

`version`
: version of the WMS service to use

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

`maxWidth`, `minWidth`, `maxHeight`, `autoPan`, `keepInView`, `closeButton`, `closeOnClick`
: popup options; see https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#popup-option

`zoomAnimation`
: deprecated. See https://github.com/Leaflet/Leaflet/blob/master/CHANGELOG.md#api-changes-5

`className`
: a CSS class name set on an element

`interactive`
: whether the element emits mouse events

`clickable`
: DEPRECATED! Use the `interactive` argument.

`noHide`, `direction`, `offset`, `permanent`
: label options; see https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#tooltip-option

`textsize`
: Change the text size of a single tooltip

`textOnly`
: Display only the text, no regular surrounding box.

`style` list of css style to be added to the tooltip

`sticky` If true, the tooltip will follow the mouse instead of being fixed at the feature center. Default value is `TRUE` (different from leaflet.js `FALSE`); see https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#tooltip-sticky

`draggable`, `keyboard`, `title`, `alt`, `zIndexOffset`, `riseOnHover`, `riseOffset`
marker options; see https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#marker-option

`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 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
- `gridOptions()`: Options for grid 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 = TRUE, mode = c("point", "polygon"))
```

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
`mode`	if `"point"` then warn about any `NA` lng/lat values; if `"polygon"` then `NA` values are expected to be used as polygon delimiters

Index