Package ‘leaflet.extras2’

May 18, 2020

Type Package

Title Extra Functionality for 'leaflet' Package

Version 1.0.0

Description
Several 'leaflet' plugins are integrated, which are available as extension to the 'leaflet' package.

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.1.0), leaflet (>= 2.0.0)

Imports htmlwidgets, htmltools, magrittr, utils

Suggests jsonlite, shiny, sf, geojsonsf, sp, testthat (>= 2.1.0), covr

URL https://trafficonese.github.io/leaflet.extras2,
https://github.com/trafficonese/leaflet.extras2

BugReports https://github.com/trafficonese/leaflet.extras2/issues

RoxygenNote 7.0.2

NeedsCompilation no

Author Gatscha Sebastian [aut, cre]

Maintainer Gatscha Sebastian <sebastian_gatscha@gmx.at>

Repository CRAN

Date/Publication 2020-05-18 15:10:06 UTC

R topics documented:

addAntpath *Add Antpath Lines*

Description

Can be used almost exactly like addPolylines but instead of pathOptions you can use antpathOptions to adapt the Antpath behaviour. See leaflet-ant-path for further details.

Usage

```
addAntpath(
  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 = antpathOptions(),
  highlightOptions = NULL,
  data = getMapData(map)
)
```

Arguments

map	a map widget object created from leaflet()
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.

`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

`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

`popup` a character vector of the HTML content for the popups (you are recommended to escape the text using `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 named list of options. See `antpathOptions`

`highlightOptions`
Options for highlighting the shape on mouse over.

`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

Value

A modified leaflet map, with an 'ant-path' animated polyline

References

`https://github.com/rubenspgcavalcante/leaflet-ant-path`

See Also

Other Antpath Functions: `antpathOptions()`, `clearAntpath()`, `removeAntpath()`

Examples

```
library(leaflet)
leaflet() %>%
  addAntpath(data = atlStorms2005)
```

`addEasyprint` *Add easyPrint Plugin*

Description

Add a control, which allows to print or export a map as .PNG.

Usage

```
addEasyprint(map, options = easyprintOptions())
```

Arguments

`map`	a map widget object created from `leaflet`
`options`	A named list of options. See `easyprintOptions`

Value

A leaflet map object

References

https://github.com/rowanwins/leaflet-easyPrint

See Also

Other EasyPrint Functions: `easyprintMap()`, `easyprintOptions()`, `removeEasyprint()`

Examples

```
library(leaflet)
leaflet()  %>%
  addTiles() %>%
  addEasyprint(options = easyprintOptions(
    title = 'Print map',
    position = 'bottomleft',
    exportOnly = TRUE))
```

`addGIBS` *Add GIBS Layers*

Description

A leaflet plugin for NASA EOSDIS GIBS imagery integration. 154 products are available. The date can be set dynamically for multi-temporal products. No-data pixels of MODIS Multiband Imagery can be made transparent.

Usage

```
addGIBS(
  map,
  layers = NULL,
  group = NULL,
  dates = NULL,
  opacity = 0.5,
  transparent = TRUE
)
```

Arguments

`map`	a map widget object created from `leaflet()`
`layers`	A character vector of GIBS-layers. See `gibs_layers`
`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.
`dates`	Date object. If multiple `layers` are added, you can add a Date vector of the same length
`opacity`	Numeric value determining the opacity. If multiple `layers` are added, you can add a numeric vector of the same length
`transparent`	Should the layer be transparent. If multiple `layers` are added, you can add a boolean vector of the same length

Value

the new `map` object

References

https://github.com/aparshin/leaflet-GIBS

See Also

Other GIBS Functions: `setDate()`, `setTransparent()`

Examples

```
library(leaflet)
library(leaflet.extras2)

layers <- gibs_layers$title[c(35, 128, 185)]

leaflet()  %>%
  addTiles() %>%
  setView(9, 50, 4) %>%
  addGIBS(layers = layers,
          dates = Sys.Date() - 1,
          group = layers) %>%
  addLayersControl(overlayGroups = layers)
```

addHeightgraph *Add a Heightgraph layer*

Description

Visualize height information and road attributes of linestring segments. The linestrings must be a Simple Feature LINESTRING Z and are transformed to GeoJSON. The function therefore inherits arguments from addGeoJSON.

Usage

```
addHeightgraph(
  map,
  data = NULL,
  columns = NULL,
  layerId = NULL,
  group = NULL,
  color = "#03F",
  weight = 5,
  opacity = 0.5,
  dashArray = NULL,
  smoothFactor = 1,
  noClip = FALSE,
  pathOpts = leaflet::pathOptions(),
  options = heightgraphOptions()
)
```

Arguments

`map`	a map widget object created from `leaflet()`
`data`	A Simple Feature LINESTRING with Z dimension.
`columns`	A character vector of the columns you want to include in the heightgraph control
`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.

`color` stroke color

`weight` stroke width in pixels

`opacity` stroke opacity (or layer opacity for tile layers)

`dashArray` a string that defines the stroke dash pattern

`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

`pathOpts` List of further options for the path. See `pathOptions`

`options` List of further plugin options. See `heightgraphOptions`

Value

the new `map` object

Note

When used in Shiny, 3 events update a certain Shiny Input:

1. A click updates `input$MAPID_heightgraph_click`
2. A mouseover updates `input$MAPID_heightgraph_mouseover`
3. A mouseout updates `input$MAPID_heightgraph_mouseout`

If you want to explicitly remove the Heightgraph control, please use `removeControl` with the `layerId = "hg_control"`.

References

`https://github.com/GIScience/Leaflet.Heightgraph`

See Also

Other Heightgraph Functions: `heightgraphOptions()`

Examples

```
## Not run:
library(leaflet)
library(leaflet.extras2)
library(sf)

data <- st_cast(st_as_sf(leaflet::atlStorms2005[4,]), "LINESTRING")
data <- st_transform(data, 4326)
data <- data.frame(st_coordinates(data))
```

```
data$elev <-  runif(nrow(data), 10, 500)
data$L1 <- NULL
L1 <- round(seq.int(1, 4, length.out = nrow(data)))
data <- st_as_sf(st_sfc(lapply(split(data, L1), sfg_linestring)))
data <- st_as_sf(st_sfc(lapply(split(data, L1), function(x) {
    st_linestring(as.matrix(x))
})))
data$steepness <- 1:nrow(data)
data$suitability <- nrow(data):1
data$popup <- apply(data, 1, function(x) {
 sprintf("Steepness: %s<br>Suitability: %s", x$steepness, x$suitability)
})

leaflet() %>%
  addTiles(group = "base") %>%
  addHeightgraph(color = "red", columns = c("steepness", "suitability"),
                 opacity = 1, data = data, group = "heightgraph",
                 options = heightgraphOptions(width = 400))

## End(Not run)
```

addHexbin *Add a Hexbin layer*

Description

Create dynamic hexbin-based heatmaps on Leaflet maps. This plugin leverages the data-binding power of d3 to allow you to dynamically update the data and visualize the transitions.

Usage

```
addHexbin(
  map,
  lng = NULL,
  lat = NULL,
  radius = 20,
  layerId = NULL,
  group = NULL,
  opacity = 0.5,
  options = hexbinOptions(),
  data = getMapData(map)
)
```

Arguments

map	a map widget object created from leaflet()
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`)

`radius` Radius of the hexbin layer

`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.

`opacity` Opacity of the hexbin layer

`options` List of further options. See `hexbinOptions`

`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

Value

the new `map` object

Note

Currently doesn't respect `layerId` nor `group`.

References

https://github.com/Asymmetrik/leaflet-d3#hexbins-api

See Also

Other Hexbin-D3 Functions: `clearHexbin()`, `hexbinOptions()`, `hideHexbin()`, `showHexbin()`, `updateHexbin()`

Examples

```
library(leaflet)
library(leaflet.extras2)

n <- 1000
df <- data.frame(lat = rnorm(n, 42.0285, .01),
                 lng = rnorm(n, -93.65, .01))

leaflet()  %>%
  addTiles() %>%
  addHexbin(lng = df$lng, lat = df$lat,
            options = hexbinOptions(
               colorRange = c("red", "yellow", "blue"),
               radiusRange = c(10, 20)
           ))
```

`addHistory` *Add History Plugin*

Description

The plugin enables tracking of map movements in a history similar to a web browser. By default, it is a simple pair of buttons – back and forward.

Usage

```
addHistory(map, layerId = NULL, options = historyOptions())
```

Arguments

`map`	a map widget object created from `leaflet`
`layerId`	the control id
`options`	A named list of options. See `historyOptions`

Value

the new `map` object

References

https://github.com/cscott530/leaflet-history

See Also

Other History Functions: `clearFuture()`, `clearHistory()`, `goBackHistory()`, `goForwardHistory()`, `historyOptions()`

Examples

```
library(leaflet)
leaflet()  %>%
  addTiles() %>%
  addHistory()
```

addMapkeyMarkers *Add Mapkey Markers*

Description

Add Mapkey Markers

Usage

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

Arguments

`map` the map to add mapkey 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 `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 markers. See `markerOptions`

`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

Value

the new `map` object

References

https://github.com/mapshakers/leaflet-mapkey-icon

See Also

Other Mapkey Functions: `[.leaflet_mapkey_icon_set()`, `makeMapkeyIcon()`, `mapkeyIconList()`, `mapkeyIcons()`

Examples

```
library(leaflet)

leaflet()  %>%
  addTiles() %>%
  addMapkeyMarkers(data = breweries91,
                icon = makeMapkeyIcon(icon = "mapkey",
                                      iconSize = 30,
                                      boxShadow = FALSE,
                                      background = "transparent"),
                group = "mapkey",
                label = ~state, popup = ~village)
```

`addOpenweatherCurrent` *Add current OpenWeatherMap Marker*

Description

Add current OpenWeatherMap Marker

Usage

```
addOpenweatherCurrent(
  map,
  apikey = NULL,
  group = NULL,
  layerId = NULL,
  options = openweatherCurrentOptions()
)
```

Arguments

map	a map widget object created from leaflet()
apikey	a valid Openweathermap-API key. Get one from here.
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.
layerId	the layer id
options	List of further options. See openweatherCurrentOptions

Value

the new map object

Note

The current weather icons will appear beginning with zoom level 9 and if used in Shiny, a click on an icon will update a Shiny input at input$MAPID_owm_click.

References

https://github.com/trafficonese/leaflet-openweathermap

See Also

Other Openweathermap Functions: addOpenweatherTiles(), openweatherCurrentOptions(), openweatherOptions()

Examples

```
## Not run:
library(leaflet)
library(leaflet.extras2)

Sys.setenv("OPENWEATHERMAP" = 'Your_API_Key')

leaflet()  %>%
  addTiles() %>% setView(9, 50, 9) %>%
  addOpenweatherCurrent(options = openweatherCurrentOptions(
    lang = "en", popup = TRUE))

## End(Not run)
```

addOpenweatherTiles *Add OpenWeatherMap Tiles*

Description

Add OpenWeatherMap Tiles

Usage

```
addOpenweatherTiles(
  map,
  apikey = NULL,
  layers = NULL,
  group = NULL,
  layerId = NULL,
  opacity = 0.5,
  options = openweatherOptions()
)
```

Arguments

`map` a map widget object created from `leaflet()`

`apikey` a valid OpenWeatherMap-API key. Get one from here.

`layers` character vector of layers you wish to add to the map. The following layers are currently possible `c("clouds","cloudsClassic","precipitation","precipitationClassic","ra`

`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.

`layerId` the layer id

`opacity` opacity of the layer

`options` List of further options. See `openweatherOptions`

Value

the new `map` object

Note

Out of the box a legend image is only available for Pressure, Precipitation Classic, Clouds Classic, Rain Classic, Snow, Temperature and Wind Speed. Please add your own images if you need some more.

References

https://github.com/trafficonese/leaflet-openweathermap

See Also

Other Openweathermap Functions: addOpenweatherCurrent(), openweatherCurrentOptions(), openweatherOptions()

Examples

```
## Not run:
library(leaflet)
library(leaflet.extras2)

Sys.setenv("OPENWEATHERMAP" = 'Your_API_Key')

leaflet()  %>%
  addTiles() %>% setView(9, 50, 6) %>%
  addOpenweatherTiles(layers = "wind")

## End(Not run)
```

addPlayback *Add Playback to Leaflet*

Description

The LeafletPlayback plugin provides the ability to replay GPS Points in the form of POINT Simple Features. Rather than simply animating a marker along a polyline, the speed of the animation is synchronized to a clock. The playback functionality is similar to a video player; you can start and stop playback or change the playback speed.

Usage

```
addPlayback(
  map,
  data,
  time = "time",
  icon = NULL,
  pathOpts = pathOptions(),
  options = playbackOptions()
)
```

Arguments

`map`	a map widget
`data`	data must be a POINT Simple Feature or a list of POINT Simple Feature's with a time column. It can also be a JSON string which must be in a specific form. See the Details for further information.
`time`	The column name of the time column. Default is `"time"`.
`icon`	an icon which can be created with `makeIcon`
`pathOpts`	style the CircleMarkers with `pathOptions`
`options`	List of additional options. See `playbackOptions`

Details

If data is a JSON string, it must have the following form:

```
{
  "type": "Feature",
  "geometry": {
    "type": "MultiPoint",
    "coordinates": [
      [-123.2653968, 44.54962188],
      [-123.26542599, 44.54951009]
    ]
},
  "properties": {
    "time": [1366067072000, 1366067074000]
  }
}
```

Value

the new `map` object

Note

If used in Shiny, you can listen to 2 events

- ‘map-ID‘+"_pb_mouseover"
- ‘map-ID‘+"_pb_click"

References

https://github.com/hallahan/LeafletPlayback

See Also

Other Playback Functions: `playbackOptions()`, `removePlayback()`

Examples

```
## Not run:
library(leaflet)
library(leaflet.extras2)
library(sf)

## Single Elements
data <- sf::st_as_sf(leaflet::atlStorms2005[1,])
data <- st_cast(data, "POINT")
data$time = as.POSIXct(
  seq.POSIXt(Sys.time() - 1000, Sys.time(), length.out = nrow(data)))

leaflet() %>%
  addTiles() %>%
  addPlayback(data = data,
              options = playbackOptions(radius = 3),
              pathOpts = pathOptions(weight = 5))


## Multiple Elements
data <- sf::st_as_sf(leaflet::atlStorms2005[1:5,])
data$Name <- as.character(data$Name)
data <- st_cast(data, "POINT")
data <- split(data, f = data$Name)
lapply(1:length(data), function(x) {
  data[[x]]$time <<- as.POSIXct(
    seq.POSIXt(Sys.time() - 1000, Sys.time(), length.out = nrow(data[[x]])))
})

leaflet() %>%
  addTiles() %>%
  addPlayback(data = data,
              options = playbackOptions(radius = 3,
                                        color = c("red","green","blue",
                                                  "orange","yellow")),
              pathOpts = pathOptions(weight = 5))

## End(Not run)
```

addReachability *Add Isochrones to Leaflet*

Description

A leaflet plugin which shows areas of reachability based on time or distance for different modes of travel using the openrouteservice isochrones API. Based on the leaflet.reachability plugin

Usage

```
addReachability(map, apikey = NULL, options = reachabilityOptions())
```

Arguments

map	a map widget
apikey	a valid Openrouteservice API-key. Can be obtained from Openrouteservice
options	A list of further options. See reachabilityOptions

Value

the new map object

Note

When used in Shiny, 3 events update a certain shiny Input:

1. reachability:displayed updates `input$MAPID_reachability_displayed`
2. reachability:delete updates `input$MAPID_reachability_delete`
3. reachability:error updates `input$MAPID_reachability_error`

References

https://github.com/traffordDataLab/leaflet.reachability

See Also

Other Reachability Functions: `reachabilityOptions()`, `removeReachability()`

Examples

```
## Not run:
library(leaflet)
library(leaflet.extras2)

Sys.setenv("OPRS" = 'Your_API_Key')

leaflet() %>%
  addTiles() %>%
  setView(8, 50, 10) %>%
  addReachability()

## End(Not run)
```

addSidebar *Add a Sidebar Leaflet Control*

Description

The sidebar plugin only works in a reactive environment (e.g Shiny), as the HTML must be created by using sidebar_tabs and sidebar_pane and it must be created before leafletOutput.

Usage

```
addSidebar(map, id = "sidebar", options = list(position = "left", fit = TRUE))
```

Arguments

map	A leaflet map widget
id	Id of the sidebar-div. Must match with the id of sidebar_tabs
options	A named list with position and fit elements.

Value

the new map object

References

https://github.com/Turbo87/sidebar-v2

See Also

Other Sidebar Functions: closeSidebar(), openSidebar(), removeSidebar(), sidebar_pane(), sidebar_tabs()

Examples

```
## Not run:
library(shiny)
runApp(paste0(system.file("examples", package = "leaflet.extras2"),
              "/sidebar_app.R"))

## End(Not run)
```

addSidebyside *Add Side by Side View*

Description

A Leaflet control to add a split screen to compare two map overlays. The plugin works with Panes, see the example.

Usage

```
addSidebyside(
  map,
  layerId = NULL,
  leftId = NULL,
  rightId = NULL,
  options = list(thumbSize = 42, padding = 0)
)
```

Arguments

map	a map widget
layerId	the layer id, needed for removeSidebyside
leftId	the layerId of the Tile layer that should be visible on the **left** side
rightId	the layerId of the Tile layer that should be visible on the **right** side
options	A list of options. Currently only thumbSize and padding can be changed.

Value

the new map object

Note

It is currently not working correctly if the baseGroups are defined in addLayersControl.

References

https://github.com/digidem/leaflet-side-by-side

See Also

Other Sidebyside Functions: removeSidebyside()

Examples

```
library(leaflet)
library(leaflet.extras2)

leaflet(quakes) %>%
  addMapPane("left", zIndex = 0) %>%
  addMapPane("right", zIndex = 0) %>%
  addTiles(group = "base", layerId = "baseid",
           options = pathOptions(pane = "right")) %>%
  addProviderTiles(providers$CartoDB.DarkMatter, group="carto", layerId = "cartoid",
                   options = pathOptions(pane = "left")) %>%
  addCircleMarkers(data = breweries91[1:15,], color = "blue", group = "blue",
                   options = pathOptions(pane = "left")) %>%
  addCircleMarkers(data = breweries91[15:20,], color = "yellow", group = "yellow") %>%
  addCircleMarkers(data = breweries91[15:30,], color = "red", group = "red",
                   options = pathOptions(pane = "right")) %>%
  addLayersControl(overlayGroups = c("blue","red", "yellow")) %>%
  addSidebyside(layerId = "sidecontrols",
                rightId = "baseid",
                leftId = "cartoid")
```

addTangram *Adds a Tangram layer to a Leaflet map in a Shiny App.*

Description

Adds a Tangram layer to a Leaflet map in a Shiny App.

Usage

```
addTangram(map, scene = NULL, layerId = NULL, group = NULL, options = NULL)
```

Arguments

map	A leaflet map widget
scene	Path to a required **.yaml** or **.zip** file. If the file is within the /www folder of a Shiny-App, only the filename must be given, otherwise the full path is needed. See the Tangram repository or the Tangram docs for further information on how to edit such a .yaml file.
layerId	A layer ID
group	The name of the group the newly created layer should belong to (for clearGroup and addLayersControl purposes).
options	A list of further options. See the app in the examples/tangram folder or the docs for further information.

Value

the new map object

Note

Only works correctly in a Shiny-App environment.

References

https://github.com/tangrams/tangram

Examples

```
## Not run:
library(shiny)
library(leaflet)
library(leaflet.extras2)

## In the /www folder of a ShinyApp. Must contain the Nextzen API-key
scene <- "scene.yaml"

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

server <- function(input, output, session) {
  output$map <- renderLeaflet({
    leaflet() %>%
      addTiles(group = "base") %>%
      addTangram(scene = scene, group = "tangram") %>%
      addCircleMarkers(data = breweries91, group = "brews") %>%
      setView(11, 49.4, 14) %>%
      addLayersControl(baseGroups = c("tangram", "base"),
                       overlayGroups = c("brews"))
  })
}

shinyApp(ui, server)

## End(Not run)
```

addVelocity *Add Velocity Animation*

Description

Add velocity animated data to leaflet. Based on the leaflet-velocity plugin

Usage

```
addVelocity(
  map,
  layerId = NULL,
  group = NULL,
  content = NULL,
  options = velocityOptions()
)
```

Arguments

`map` a map widget object created from `leaflet()`

`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.

`content` the path or URL to a JSON file representing the velocity data or a data.frame which can be transformed to such a JSON file. Please see the demo files for some example data.

`options` List of further options. See `velocityOptions`

Value

the new `map` object

References

`https://github.com/danwild/leaflet-velocity`

See Also

Other Velocity Functions: `removeVelocity()`, `setOptionsVelocity()`, `velocityOptions()`

Examples

```
## Not run:
library(leaflet)
library(leaflet.extras2)
content <- "https://raw.githubusercontent.com/danwild/leaflet-velocity/master/demo/wind-gbr.json"
leaflet() %>%
  addTiles(group = "base") %>%
  setView(145, -20, 4) %>%
  addVelocity(content = content, group = "velo", layerId = "veloid") %>%
  addLayersControl(baseGroups = "base", overlayGroups = "velo")

## End(Not run)
```

addWMS *Add Queryable WMS Layer*

Description

A Leaflet plugin for working with Web Map services, providing: single-tile/untiled/nontiled layers, shared WMS sources, and GetFeatureInfo-powered identify.

Usage

```
addWMS(
  map,
  baseUrl,
  layers = NULL,
  group = NULL,
  options = WMSTileOptions(),
  attribution = NULL,
  popupOptions = NULL,
  data = getMapData(map)
)
```

Arguments

`map`	a map widget object created from `leaflet()`
`baseUrl`	a base URL of the WMS service
`layers`	vector or list of WMS layers to show. The name of the layer is used as the `layerId` (for `removeTiles` purposes)
`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`	List of further options. See `WMSTileOptions`
`attribution`	the attribution text of the tile layer (HTML)
`popupOptions`	List of popup options. See `popupOptions`. Default is NULL.
`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

Value

the new `map` object

References

`https://github.com/heigeo/leaflet.wms`

Examples

```
library(leaflet)
library(leaflet.extras2)

leaflet() %>%
  addTiles(group = "base") %>%
  setView(9, 50, 5) %>%
  addWMS(baseUrl = "https://maps.dwd.de/geoserver/dwd/wms",
         layers = "dwd:BRD_1km_winddaten_10m",
      popupOptions = popupOptions(maxWidth = 600),
      options = WMSTileOptions(
        transparent = TRUE,
        format = "image/png",
        info_format = "text/html"))
```

antpathOptions *Antpath Options*

Description

Additional list of options for 'ant-path' animated polylines.

Usage

```
antpathOptions(
  delay = 400,
  paused = FALSE,
  reverse = FALSE,
  hardwareAccelerated = FALSE,
  dashArray = c(10, 20),
  pulseColor = "#ffffff",
  lineCap = NULL,
  lineJoin = NULL,
  interactive = TRUE,
  pointerEvents = NULL,
  className = ""
)
```

Arguments

`delay`	Add a delay to the animation flux. Default is `400`
`paused`	Should the animation be paused. Default is `FALSE`
`reverse`	Defines if the flow follows the path order or not. Default is `FALSE`
`hardwareAccelerated`	Makes the animation run with hardware acceleration. Default is `FALSE`
`dashArray`	The size of the animated dashes. Default is `c(10,20)`

`pulseColor`	Adds a color to the dashed flux. Default is `#ffffff`
`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
`interactive`	whether the element emits mouse events
`pointerEvents`	sets the `pointer-events` attribute on the path if SVG backend is used
`className`	a CSS class name set on an element

Value

A list of options for `addAntpath` animated polylines

See Also

Other Antpath Functions: `addAntpath()`, `clearAntpath()`, `removeAntpath()`

`clearAntpath` *clearAntpath*

Description

Clear all Antpaths

Usage

```
clearAntpath(map)
```

Arguments

`map`	a map widget object, possibly created from `leaflet()` but more likely from `leafletProxy()`

Value

the new `map` object

See Also

Other Antpath Functions: `addAntpath()`, `antpathOptions()`, `removeAntpath()`

`clearFuture` *clearFuture*

Description

Resets the stack of future items.

Usage

```
clearFuture(map)
```

Arguments

`map`	a map widget object created from `leafletProxy`

Value

the new `map` object

References

https://github.com/cscott530/leaflet-history

See Also

Other History Functions: `addHistory()`, `clearHistory()`, `goBackHistory()`, `goForwardHistory()`, `historyOptions()`

`clearHexbin` *clearHexbin*

Description

Clears the data of the hexbinLayer.

Usage

```
clearHexbin(map)
```

Arguments

`map`	The map widget

Value

the new `map` object

See Also

Other Hexbin-D3 Functions: `addHexbin()`, `hexbinOptions()`, `hideHexbin()`, `showHexbin()`, `updateHexbin()`

`clearHistory` *clearHistory*

Description

Resets the stack of history items.

Usage

```
clearHistory(map)
```

Arguments

`map`	a map widget object created from `leafletProxy`

Value

the new `map` object

References

`https://github.com/cscott530/leaflet-history`

See Also

Other History Functions: `addHistory()`, `clearFuture()`, `goBackHistory()`, `goForwardHistory()`, `historyOptions()`

`closeSidebar` *Close the Sidebar*

Description

Close the Sidebar

Usage

```
closeSidebar(map)
```

Arguments

`map`	A leaflet map widget

Value

the new map object

See Also

Other Sidebar Functions: addSidebar(), openSidebar(), removeSidebar(), sidebar_pane(), sidebar_tabs()

---

easyprintMap *easyprintMap*

Description

Print or export a map programmatically (e.g. in a Shiny environment).

Usage

```
easyprintMap(map, sizeModes = "A4Portrait", filename = "map")
```

Arguments

map	the map widget
sizeModes	Options available include CurrentSize, A4Portrait, A4Landscape or a custom size object. Default is A4Portrait
filename	Name of the file if exportOnly option is TRUE.

Value

A leaflet map object

See Also

Other EasyPrint Functions: addEasyprint(), easyprintOptions(), removeEasyprint()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {
library(shiny)
library(leaflet)
library(leaflet.extras2)

ui <- fluidPage(
  leafletOutput("map"),
 selectInput("scene", "Select Scene", choices = c("CurrentSize", "A4Landscape", "A4Portrait")),
  actionButton("print", "Print Map")
)
server <- function(input, output, session) {
  output$map <- renderLeaflet({
    input$print
    leaflet()  %>%
      addTiles() %>%
      setView(10, 50, 9) %>%
      addEasyprint(options = easyprintOptions(
        exportOnly = TRUE
      ))
})
  observeEvent(input$print, {
    leafletProxy("map") %>%
      easyprintMap(sizeModes = input$scene)
})
}

shinyApp(ui, server)
}
```

easyprintOptions *easyprintOptions*

Description

Create a list of further options for the easyprint plugin.

Usage

```
easyprintOptions(
  title = "Print map",
  position = "topleft",
  sizeModes = list("A4Portrait", "A4Landscape", "Current"),
  defaultSizeTitles = NULL,
  exportOnly = FALSE,
  tileLayer = NULL,
  tileWait = 500,
  filename = "map",
  hidden = FALSE,
  hideControlContainer = TRUE,
  hideClasses = list(),
  customWindowTitle = NULL,
  spinnerBgColor = "#0DC5C1",
  customSpinnerClass = "epLoader"
)
```

Arguments

`title`	Sets the text which appears as the tooltip of the print/export button

`position` Positions the print button

`sizeModes` Options available include `CurrentSize`, `A4Portrait`, `A4Landscape` or a custom size object

`defaultSizeTitles` Button tooltips for the default page sizes

`exportOnly` If set to `TRUE` the map is exported to a .png file

`tileLayer` A tile layer that you can wait for to draw (helpful when resizing)

`tileWait` How long to wait for the tiles to draw (helpful when resizing)

`filename` Name of the file if `exportOnly` option is `TRUE`

`hidden` Set to `TRUE` if you don't want to display the toolbar. Instead you can create your own buttons or fire print events programmatically.

`hideControlContainer` Hides the leaflet controls like the zoom buttons and the attribution on the print out

`hideClasses` Hides classes on the print out. Use a list of strings as follow : list('div1', 'div2')

`customWindowTitle` A title for the print window which will get added the printed paper

`spinnerBgColor` A valid css colour for the spinner background color

`customSpinnerClass` A class for a custom css spinner to use while waiting for the print.

Value

A list of options for the 'easyprint' control

References

https://github.com/rowanwins/leaflet-easyPrint

See Also

Other EasyPrint Functions: `addEasyprint()`, `easyprintMap()`, `removeEasyprint()`

`gibs_layers` *The available GIBS layers with attributes*

Description

The available GIBS layers with attributes

Usage

```
gibs_layers
```

Format

An object of class `data.frame` with 276 rows and 4 columns.

goBackHistory *goBackHistory*

Description

If possible, will go to previous map extent. Pushes current extent to the "future" stack.

Usage

```
goBackHistory(map)
```

Arguments

`map`	a map widget object created from `leafletProxy`

Value

the new `map` object

References

https://github.com/cscott530/leaflet-history

See Also

Other History Functions: `addHistory()`, `clearFuture()`, `clearHistory()`, `goForwardHistory()`, `historyOptions()`

goForwardHistory *goForwardHistory*

Description

If possible, will go to next map extent. Pushes current extent to the "back" stack.

Usage

```
goForwardHistory(map)
```

Arguments

`map`	a map widget object created from `leafletProxy`

Value

the new `map` object

References

https://github.com/cscott530/leaflet-history

See Also

Other History Functions: `addHistory()`, `clearFuture()`, `clearHistory()`, `goBackHistory()`, `historyOptions()`

`heightgraphOptions` *heightgraphOptions*

Description

Customize the heightgraph with the following additional options.

Usage

```
heightgraphOptions(
  position = c("bottomright", "topleft", "topright", "bottomleft"),
  width = 800,
  height = 200,
  margins = list(top = 10, right = 30, bottom = 55, left = 50),
  expand = TRUE,
  expandCallback = NULL,
  mappings = NULL,
  highlightStyle = list(color = "red"),
  translation = NULL,
  xTicks = 3,
  yTicks = 3
)
```

Arguments

`position`	position of control: "topleft", "topright", "bottomleft", or "bottomright". Default is `bottomright`.
`width`	The width of the expanded heightgraph display in pixels. Default is `800`.
`height`	The height of the expanded heightgraph display in pixels. Default is `200`.
`margins`	The margins define the distance between the border of the heightgraph and the actual graph inside. You are able to specify margins for top, right, bottom and left in pixels. Default is `list(top = 10,right = 30,bottom = 55,left = 50)`.
`expand`	Boolean value that defines if the heightgraph should be expanded on creation. Default is `200`.
`expandCallback`	Function to be called if the heightgraph is expanded or reduced. The state of the heightgraph is passed as an argument. It is `TRUE` when expanded and `FALSE` when reduced. Default is `NULL`.

`mappings` You may add a mappings object to customize the colors and labels in the height graph. Without adding custom mappings the segments and labels within the graph will be displayed in random colors. Each key of the object must correspond to the `summary` key in `properties` within the `FeatureCollection`. Default is `NULL`.

`highlightStyle` You can customize the highlight style when using the horizontal line to find parts of the route above an elevation value. Use any Leaflet Path options as value of the highlightStyle parameter. Default is `list(color = "red")`.

`translation` You can change the labels of the heightgraph info field by passing translations for `distance`, `elevation`, `segment_length`, `type` and `legend`. Default is `NULL`.

`xTicks` Specify the tick frequency in the x axis of the graph. Corresponds approximately to 2 to the power of value ticks. Default is `3`.

`yTicks` Specify the tick frequency in the y axis of the graph. Corresponds approximately to 2 to the power of value ticks. Default is `3`.

Value

A list of further options for `addHeightgraph`

See Also

Other Heightgraph Functions: `addHeightgraph()`

`hexbinOptions` *hexbinOptions*

Description

A list of options for customizing the appearance/behavior of the hexbin layer.

Usage

```
hexbinOptions(
  duration = 200,
  colorScaleExtent = NULL,
  radiusScaleExtent = NULL,
  colorRange = c("#f7fbff", "#08306b"),
  radiusRange = c(5, 15),
  pointerEvents = "all",
  resizetoCount = FALSE,
  tooltip = "Count "
)
```

Arguments

`duration` Transition duration for the hexbin layer

`colorScaleExtent` extent of the color scale for the hexbin layer. This is used to override the derived extent of the color values and is specified as a vector of the form c(min= numeric, max= numeric). Can be a numeric vector or a custom `JS` array, like (`JS("[40,undefined]")`)

`radiusScaleExtent` This is the same exact configuration option as colorScaleExtent, only applied to the radius extent.

`colorRange` Sets the range of the color scale used to fill the hexbins on the layer.

`radiusRange` Sets the range of the radius scale used to size the hexbins on the layer.

`pointerEvents` This value is passed directly to an element-level css style for pointer-events. You should only modify this config option if you want to change the mouse event behavior on hexbins. This will modify when the events are propagated based on the visibility state and/or part of the hexbin being hovered.

`resizetoCount` Resizes the hexbin to the count. Default is `FALSE`. If set to `TRUE` it will resize based on the amount of underlying elements. You can also pass a custom `JS` function.

`tooltip` Should tooltips be displayed? If set to `TRUE`, it will show the amount of underlying elements. If a string is given, it will append the string before the count. To disable tooltips, please pass `NULL` or `FALSE`. You can also pass a custom `JS` function.

Value

A list of hexbin-specific options

See Also

Other Hexbin-D3 Functions: `addHexbin()`, `clearHexbin()`, `hideHexbin()`, `showHexbin()`, `updateHexbin()`

`hideHexbin` *hideHexbin*

Description

Hide the hexbinLayer.

Usage

```
hideHexbin(map)
```

Arguments

`map` The map widget

Value

the new map object

See Also

Other Hexbin-D3 Functions: addHexbin(), clearHexbin(), hexbinOptions(), showHexbin(), updateHexbin()

---

historyOptions *History Options*

Description

History Options

Usage

```
historyOptions(
  position = c("topright", "topleft", "bottomleft", "bottomright"),
  maxMovesToSave = 10,
  backImage = "fa fa-caret-left",
  forwardImage = "fa fa-caret-right",
  backText = "",
  forwardText = "",
  backTooltip = "Go to Previous Extent",
  forwardTooltip = "Go to Next Extent",
  backImageBeforeText = TRUE,
  forwardImageBeforeText = FALSE,
  orientation = c("horizontal", "vertical"),
  shouldSaveMoveInHistory = NULL
)
```

Arguments

`position`	Set the position of the History control. Default is `topright`.
`maxMovesToSave`	Number of moves in the history to save before clearing out the oldest. Default value is 10, use 0 or a negative number to make unlimited.
`backImage`	The class for the ‘back‘ button icon. Default is `"fa fa-caret-left"`.
`forwardImage`	The class for the ‘forward‘ button icon. Default is `"fa fa-caret-right"`.
`backText`	The text in the buttons. Default is ”.
`forwardText`	The text in the buttons. Default is ”.
`backTooltip`	Tooltip content. Default is `"Go to Previous Extent"`.
`forwardTooltip`	Tooltip content. Default is `"Go to Next Extent"`.

`backImageBeforeText`
: When both text and image are present, whether to show the image first or the text first (left to right). Default is `TRUE`

`forwardImageBeforeText`
: When both text and image are present, whether to show the image first or the text first (left to right). Default is `FALSE`

`orientation`
: Whether to position the buttons on top of one another or side-by-side. Default is `horizontal`

`shouldSaveMoveInHistory`
: A JS callback you can provide that gets called with every move. return false to not save a move.

Value

A list of further options for `addHistory`

References

https://github.com/cscott530/leaflet-history

See Also

Other History Functions: `addHistory()`, `clearFuture()`, `clearHistory()`, `goBackHistory()`, `goForwardHistory()`

Examples

```
library(leaflet)
leaflet()  %>%
  addTiles() %>%
    addHistory(options = historyOptions(position = "bottomright",
    maxMovesToSave = 20,
    backText =  "Go back",
    forwardText = "Go forward",
    orientation = "vertical"
    ))
```

`leaflet.extras2` *leaflet.extras2: Extra Functionality for 'leaflet' Package.*

Description

This package serves as an add-on to the 'leaflet' package by providing extra functionality via 'leaflet' plugins.

makeMapkeyIcon *Make Mapkey Icon*

Description

Make Mapkey Icon

Usage

```
makeMapkeyIcon(
  icon = "mapkey",
  color = "#ff0000",
  iconSize = 12,
  background = "#1F7499",
  borderRadius = "100%",
  hoverScale = 1.4,
  hoverEffect = TRUE,
  additionalCSS = NULL,
  hoverCSS = NULL,
  htmlCode = NULL,
  boxShadow = TRUE
)
```

Arguments

`icon`	ID of the mapkey Icon you want to use. See mapkeyicons.com for a full list.
`color`	Any CSS color (e.g. 'red','rgba(20,160,90,0.5)', '#686868', ...)
`iconSize`	Size of Icon in Pixels. Default is 12
`background`	Any CSS color or false for no background
`borderRadius`	Any number (for circle size/2, for square 0.001)
`hoverScale`	Any real number (best result in range 1 - 2, use 1 for no effect)
`hoverEffect`	Switch on/off effect on hover
`additionalCSS`	CSS code (e.g. "border:4px solid #aa3838;")
`hoverCSS`	CSS code (e.g. "background-color:#992b00 !important; color:#99defc !important;")
`htmlCode`	e.g. '&#57347;&#xe003;'. See mapkeyicons.com for further information
`boxShadow`	Should a shadow be visible

Value

A list of mapkey-icon data that can be passed to the argument `icon`

References

`https://github.com/mapshakers/leaflet-mapkey-icon`

See Also

Other Mapkey Functions: [.leaflet_mapkey_icon_set(), addMapkeyMarkers(), mapkeyIconList(), mapkeyIcons()

Examples

```
makeMapkeyIcon(icon = "traffic_signal",
               color = "#0000ff",
               iconSize = 12,
               boxShadow = FALSE,
               background="transparent")
```

---

mapkeyIconList *Make Mapkey-icon set*

Description

Make Mapkey-icon set

Usage

```
mapkeyIconList(...)
```

Arguments

...	icons created from makeMapkeyIcon()

Value

A list of class "leaflet_mapkey_icon_set"

References

https://github.com/mapshakers/leaflet-mapkey-icon

See Also

Other Mapkey Functions: [.leaflet_mapkey_icon_set(), addMapkeyMarkers(), makeMapkeyIcon(), mapkeyIcons()

Examples

```
iconSet = mapkeyIconList(
  red = makeMapkeyIcon(color = "#ff0000"),
  blue = makeMapkeyIcon(color = "#0000ff")
)
iconSet[c("red", "blue")]
```

mapkeyIcons *Create a list of Mapkey icon data*

Description

An icon can be represented as a list of the form `list(color,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

```
mapkeyIcons(
  icon = "mapkey",
  color = "#ff0000",
  iconSize = 12,
  background = "#1F7499",
  borderRadius = "100%",
  hoverScale = 1.4,
  hoverEffect = TRUE,
  hoverCSS = NULL,
  additionalCSS = NULL,
  htmlCode = NULL,
  boxShadow = TRUE
)
```

Arguments

`icon`	ID of the mapkey Icon you want to use. See mapkeyicons.com for a full list.
`color`	Any CSS color (e.g. 'red','rgba(20,160,90,0.5)', '#686868', ...)
`iconSize`	Size of Icon in Pixels. Default is 12
`background`	Any CSS color or false for no background
`borderRadius`	Any number (for circle size/2, for square 0.001)
`hoverScale`	Any real number (best result in range 1 - 2, use 1 for no effect)
`hoverEffect`	Switch on/off effect on hover
`hoverCSS`	CSS code (e.g. "background-color:#992b00 !important; color:#99defc !important;")
`additionalCSS`	CSS code (e.g. "border:4px solid #aa3838;")
`htmlCode`	e.g. '&#57347;&#xe003;'. See mapkeyicons.com for further information
`boxShadow`	Should a shadow be visible

Value

A list of mapkey-icon data that can be passed to the argument `icon`

References

https://github.com/mapshakers/leaflet-mapkey-icon

See Also

Other Mapkey Functions: [.leaflet_mapkey_icon_set(), addMapkeyMarkers(), makeMapkeyIcon(), mapkeyIconList()

Examples

```
library(leaflet)
leaflet()  %>%
  addMapkeyMarkers(data = breweries91,
                   icon = mapkeyIcons(
                     color = "red",
                     borderRadius = 0,
                     iconSize = 25))
```

openSidebar *Open the Sidebar by ID*

Description

Open the Sidebar by ID

Usage

```
openSidebar(map, id)
```

Arguments

map	A leaflet map widget
id	The id of the sidebar_pane to open

Value

the new map object

See Also

Other Sidebar Functions: addSidebar(), closeSidebar(), removeSidebar(), sidebar_pane(), sidebar_tabs()

`openweatherCurrentOptions` *openweatherCurrentOptions*

Description

openweatherCurrentOptions

Usage

```
openweatherCurrentOptions(lang = "en", minZoom = 7, interval = 10, ...)
```

Arguments

`lang` ’en’, ’de’, ’ru’, ’fr’, ’es’, ’ca’. Language of popup texts. Note: not every translation is finished yet.

`minZoom` Number (7). Minimal zoom level for fetching city data. Use smaller values only at your own risk.

`interval` Number (0). Time in minutes to reload city data. Please do not use less than 10 minutes.

`...` Further options passed to `L.OWM.current`. See the full list of options

Value

A list of options for `addOpenweatherCurrent`

See Also

Other Openweathermap Functions: `addOpenweatherCurrent()`, `addOpenweatherTiles()`, `openweatherOptions()`

`openweatherOptions` *OpenWeatherMap Options*

Description

OpenWeatherMap Options

Usage

```
openweatherOptions(
  showLegend = TRUE,
  legendImagePath = NULL,
  legendPosition = c("bottomleft", "bottomright", "topleft", "topright")
)
```

Arguments

showLegend If TRUE and option legendImagePath is set there will be a legend image on the map

legendImagePath
A URL (is set to a default image for some layers, null for others, see below). URL or relative path to an image which is a legend to this layer

legendPosition Position of the legend images on the map. Must be one of 'bottomleft','bottomright','topleft','

Value

A list of options for addOpenweatherTiles

See Also

Other Openweathermap Functions: addOpenweatherCurrent(), addOpenweatherTiles(), openweatherCurrentOptions

---

playbackOptions *playbackOptions*

Description

A list options for addPlayback. For a full list please visit the plugin repository.

Usage

```
playbackOptions(
  color = "blue",
  radius = 5,
  tickLen = 250,
  speed = 1,
  maxInterpolationTime = 5 * 60 * 1000,
  tracksLayer = TRUE,
  playControl = TRUE,
  dateControl = TRUE,
  sliderControl = TRUE,
  staleTime = 60 * 60 * 1000,
  ...
)
```

Arguments

color colors of the CircleMarkers.

radius a numeric value for the radius of the CircleMarkers.

tickLen Set tick length in milliseconds. Increasing this value, may improve performance, at the cost of animation smoothness. Default is 250

speed Set float multiplier for default animation speed. Default is 1

`maxInterpolationTime`
: Set max interpolation time in seconds. Default is 5*60*1000 (5 minutes).

`tracksLayer` Set `TRUE` if you want to show layer control on the map. Default is `TRUE`

`playControl` Set `TRUE` if play button is needed. Default is `TRUE`

`dateControl` Set `TRUE` if date label is needed. Default is `TRUE`

`sliderControl` Set `TRUE` if slider control is needed. Default is `TRUE`

`staleTime` Set time before a track is considered stale and faded out. Default is 60*60*1000 (1 hour)

`...` Further arguments passed to ‘L.Playback‘

Value

A list of options for `addPlayback`

References

https://github.com/hallahan/LeafletPlayback

See Also

Other Playback Functions: `addPlayback()`, `removePlayback()`

---

reachabilityOptions *reachabilityOptions*

Description

Add extra options. For a full list please visit the plugin repository.

Usage

```
reachabilityOptions(
  collapsed = TRUE,
  pane = "overlayPane",
  position = "topleft",
  ...
)
```

Arguments

`collapsed` Should the control widget start in a collapsed mode. Default is `TRUE`

`pane` Leaflet pane to add the isolines GeoJSON to. Default is `overlayPane`

`position` Leaflet control pane position. Default is `topleft`

`...` Further arguments passed to ‘L.Control.Reachability‘

Value

A list of options for `addReachability`

References

https://github.com/traffordDataLab/leaflet.reachability

See Also

Other Reachability Functions: `addReachability()`, `removeReachability()`

`removeAntpath` *removeAntpath*

Description

Remove one or more Antpaths from a map, identified by `layerId`.

Usage

```
removeAntpath(map, layerId = NULL)
```

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

Value

the new `map` object

See Also

Other Antpath Functions: `addAntpath()`, `antpathOptions()`, `clearAntpath()`

`removeEasyprint` *removeEasyprint*

Description

Removes the easyprint control from the map.

Usage

```
removeEasyprint(map)
```

Arguments

`map`	the map widget

Value

A leaflet map object

See Also

Other EasyPrint Functions: `addEasyprint()`, `easyprintMap()`, `easyprintOptions()`

`removePlayback` *removePlayback*

Description

Remove the Playback controls and markers.

Usage

```
removePlayback(map)
```

Arguments

`map`	the map widget

Value

the new `map` object

See Also

Other Playback Functions: `addPlayback()`, `playbackOptions()`

`removeReachability` *removeReachability*

Description

Remove the reachability controls.

Usage

```
removeReachability(map)
```

Arguments

`map`	the map widget.

Value

the new `map` object

See Also

Other Reachability Functions: `addReachability()`, `reachabilityOptions()`

`removeSidebar` *Remove the Sidebar*

Description

Remove the Sidebar

Usage

```
removeSidebar(map)
```

Arguments

`map`	A leaflet map widget

Value

the new `map` object

See Also

Other Sidebar Functions: `addSidebar()`, `closeSidebar()`, `openSidebar()`, `sidebar_pane()`, `sidebar_tabs()`

`removeSidebyside` *removeSidebyside*

Description

removeSidebyside

Usage

```
removeSidebyside(map, layerId = NULL)
```

Arguments

`map`	a map widget
`layerId`	the layer id of the `addSidebyside` layer

Value

the new `map` object

See Also

Other Sidebyside Functions: `addSidebyside()`

`removeVelocity` *removeVelocity*

Description

removeVelocity

Usage

```
removeVelocity(map, group)
```

Arguments

`map`	the map widget
`group`	the group to remove

Value

the new `map` object

See Also

Other Velocity Functions: `addVelocity()`, `setOptionsVelocity()`, `velocityOptions()`

`setDate` *Set Date for GIBS Layers*

Description

Set a new date for multi-temporal layers.

Usage

```
setDate(map, layers = NULL, dates = NULL)
```

Arguments

`map`	a map widget object created from `leaflet()`
`layers`	A character vector of GIBS-layers. See `gibs_layers`
`dates`	Date object. If multiple `layers` are added, you can add a Date vector of the same length

Value

the new `map` object

See Also

Other GIBS Functions: `addGIBS()`, `setTransparent()`

`setOptionsVelocity` *setOptionsVelocity*

Description

setOptionsVelocity

Usage

```
setOptionsVelocity(map, layerId, options)
```

Arguments

`map`	the map widget
`layerId`	the layer id
`options`	see `velocityOptions`

Value

the new `map` object

See Also

Other Velocity Functions: `addVelocity()`, `removeVelocity()`, `velocityOptions()`

`setTransparent` *Set Transparency for GIBS Layers*

Description

Change the transparency for no-data pixels.

Usage

```
setTransparent(map, layers = NULL, transparent = TRUE)
```

Arguments

`map`	a map widget object created from `leaflet()`
`layers`	A character vector of GIBS-layers. See `gibs_layers`
`transparent`	Should the layer be transparent. If multiple `layers` are added, you can add a boolean vector of the same length

Value

the new `map` object

See Also

Other GIBS Functions: `addGIBS()`, `setDate()`

`showHexbin` *showHexbin*

Description

Show the hexbinLayer.

Usage

```
showHexbin(map)
```

Arguments

`map`	The map widget

Value

the new map object

See Also

Other Hexbin-D3 Functions: addHexbin(), clearHexbin(), hexbinOptions(), hideHexbin(), updateHexbin()

---

sidebar_pane *Create a Sidebar Pane*

Description

Create a Sidebar Pane

Usage

```
sidebar_pane(
  title = "Sidebar Title",
  id = NULL,
  icon = icon("caret-right"),
  ...
)
```

Arguments

title	A title for the sidebar panel
id	An id for the sidebar panel
icon	An icon for the sidebar panel
...	List of elements to include in the panel

Value

A shiny.tag with sidebar-specific HTML classes

References

https://github.com/Turbo87/sidebar-v2, https://github.com/Turbo87/sidebar-v2/blob/master/doc/usage.md

See Also

Other Sidebar Functions: addSidebar(), closeSidebar(), openSidebar(), removeSidebar(), sidebar_tabs()

Examples

```
## Not run:
library(shiny)
sidebar_pane(id = "id", icon = icon("cars"), tags$div())

## End(Not run)
```

---

sidebar_tabs *Create a Sidebar*

Description

Create a Sidebar

Usage

```
sidebar_tabs(id = "sidebar", iconList = NULL, ...)
```

Arguments

id	The id of the sidebar, which must match the id of addSidebar. Default is "sidebar"
iconList	A list of icons to be shown, when the sidebar is collapsed. The list is required and must match the amount of sidebar_pane.
...	The individual sidebar_pane.

Value

A shiny.tag with individual sidebar panes

References

https://github.com/Turbo87/sidebar-v2, https://github.com/Turbo87/sidebar-v2/blob/master/doc/usage.md

See Also

Other Sidebar Functions: addSidebar(), closeSidebar(), openSidebar(), removeSidebar(), sidebar_pane()

Examples

```
## Not run:
library(shiny)
runApp(paste0(system.file("examples", package = "leaflet.extras2"),
              "/sidebar_app.R"))

## End(Not run)
```

to_ms *to_ms Change POSIX or Date to milliseconds*

Description

to_ms Change POSIX or Date to milliseconds

Usage

```
to_ms(data, time)
```

Arguments

`data`	The data
`time`	Columnname of the time column.

Value

A data.frame with the time column in milliseconds

updateHexbin *updateHexbin*

Description

Dynamically change the `data` and/or the `colorRange`.

Usage

```
updateHexbin(map, data = NULL, lng = NULL, lat = NULL, colorRange = NULL)
```

Arguments

`map`	a map widget object created from `leaflet()`
`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
`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`)
`colorRange`	The range of the color scale used to fill the hexbins

Value

the new map object

See Also

Other Hexbin-D3 Functions: addHexbin(), clearHexbin(), hexbinOptions(), hideHexbin(), showHexbin()

velocityOptions *velocityOptions*

Description

Define further options for the velocity layer.

Usage

```
velocityOptions(
  speedUnit = c("m/s", "k/h", "kt"),
  minVelocity = 0,
  maxVelocity = 10,
  velocityScale = 0.005,
  colorScale = NULL,
  ...
)
```

Arguments

speedUnit	Could be 'm/s' for meter per second, 'k/h' for kilometer per hour or 'kt' for knots
minVelocity	velocity at which particle intensity is minimum
maxVelocity	velocity at which particle intensity is maximum
velocityScale	scale for wind velocity
colorScale	A vector of hex colors or an RGB matrix
...	Further arguments passed to the Velocity layer and Windy.js. For more information, please visit leaflet-velocity plugin

Value

A list of further options for addVelocity

See Also

Other Velocity Functions: addVelocity(), removeVelocity(), setOptionsVelocity()

[.leaflet_mapkey_icon_set *leaflet_mapkey_icon_set*

Description

leaflet_mapkey_icon_set

Usage

```
## S3 method for class 'leaflet_mapkey_icon_set'
x[i]
```

Arguments

x	icons
i	offset

See Also

Other Mapkey Functions: addMapkeyMarkers(), makeMapkeyIcon(), mapkeyIconList(), mapkeyIcons()

Index