Package 'leaflet.extras'

October 13, 2022

Type Package

Title Extra Functionality for 'leaflet' Package

Version 1.0.0

Description The 'leaflet' JavaScript library provides many plugins some of which are available in the core 'leaflet' package, but there are many more. It is not possible to support them all in the core 'leaflet' package. This package serves as an add-on to the 'leaflet' package by providing extra functionality via 'leaflet' plugins.

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

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

Imports htmlwidgets, htmltools, stringr, magrittr

Suggests jsonlite, readr

URL https://github.com/bhaskarvk/leaflet.extras,
https://bhaskarvk.github.io/leaflet.extras/

BugReports https://github.com/bhaskarvk/leaflet.extras/issues

RoxygenNote 6.0.1

NeedsCompilation no

Author Bhaskar Karambelkar [aut, cre],
Barret Schloerke [aut],
Bangyou Zheng [ctb] (Leaflet-search and Leaflet-GPS plugin integration),
Robin Cura [ctb] (Fixes for Draw Options),
Markus Voge [ctb] (Enhancements for Draw Options),
Markus Dumke [ctb] (Bounce Marker addition),
Mapbox [ctb, cph] (leaflet-omnivore, csv2geojson, and togeojson libraries),
Henry Thasler [ctb, cph] (Leaflet.Geodesic library),
Dennis Wilhelm [ctb, cph] (Leaflet.StyleEditor library),
Kirollos Risk [ctb, cph] (fuse.js library),
Tim Wisniewski [ctb, cph] (leaflet-choropleth library),

Leaflet [ctb, cph] (leaflet-draw library),
Alexander Milevski [ctb, cph] (leaflet-draw-drag library),
John Firebaugh [ctb, cph] (leaflet-fullscreen library),
Stefano Cudini [ctb, cph] (leaflet-gps library),
Johannes Rudolph [ctb, cph] (leaflet-hash library),
Per Liedman [ctb, cph] (leaflet-measure-path library),
Pavel Shramov [ctb, cph] (leaflet-plugins library),
Filip Zavadil [ctb, cph] (leaflet-pulse-icon library),
Stefano Cudini [ctb, cph] (leaflet-search library),
CliffCloud [ctb, cph] (leaflet-sleep library),
Ursudio [ctb, cph] (leaflet-webgl-heatmap library),
Maxime Hadjinlian [ctb, cph] (leaflet.BounceMarker library),
Vladimir Agafonkin [ctb, cph] (leaflet.heat library),
Iván Sánchez Ortega [ctb, cph] (leaflet.tilelayer.pouchdbcached library),
Dale Harvey [ctb, cph] (pouchdb-browser library),
Mike Bostock [ctb, cph] (topojson library)

Maintainer Bhaskar Karambelkar <bhaskarvk@gmail.com>

Repository CRAN

Date/Publication 2018-04-21 21:20:53 UTC

R topics documented:

`addAwesomeMarkersDependencies` *Add AwesomeMarkers and related lib dependencies to a map*

Description

Add AwesomeMarkers and related lib dependencies to a map

Usage

```
addAwesomeMarkersDependencies(map, libs)
```

Arguments

`map`	the map widget
`libs`	char vector with lib names.

`addBingTiles` *Adds Bing Tiles Layer*

Description

Adds Bing Tiles Layer

Usage

```
addBingTiles(map, apikey = Sys.getenv("BING_MAPS_API_KEY"),
  imagerySet = c("Aerial", "AerialWithLabels", "CanvasDark", "CanvasLight",
  "CanvasGray", "Road"), layerId = NULL, group = NULL, ...)
```

Arguments

`map`	The Map widget
`apikey`	String. Bing API Key
`imagerySet`	String. Type of Tiles to display
`layerId`	String. An optional unique ID for the layer
`group`	String. An optional group name for the layer
`...`	Optional Parameters required by the Bing API described at `https://msdn.microsoft.com/en-us/library/ff701716.aspx`

See Also

Get a Bing Maps API Key: `https://msdn.microsoft.com/en-us/library/ff428642.aspx`

`addBootstrapDependency` *Add Bootstrap dependency to a map*

Description

Add Bootstrap dependency to a map

Usage

```
addBootstrapDependency(map)
```

Arguments

`map`	the map widget

`addBounceMarkers` *Add Bounce Markers to map*

Description

Add Bounce Markers to map

Usage

```
addBounceMarkers(map, lat, lng, duration = 1000, height = 100)
```

Arguments

`map`	map object created by leaflet::leaflet
`lat`	numeric latitude
`lng`	numeric longitude
`duration`	integer scalar: The duration of the animation in milliseconds.
`height`	integer scalar: Height at which the marker is dropped.

Author(s)

Markus Dumke

See Also

GitHub: leaflet.bouncemarker

Examples

```
leaflet() %>%
  addTiles() %>%
  addBounceMarkers(49, 11)
```

---

addDrawToolbar *Adds a Toolbar to draw shapes/points on the map.*

Description

Adds a Toolbar to draw shapes/points on the map.

Removes the draw toolbar

Usage

```
addDrawToolbar(map, targetLayerId = NULL, targetGroup = NULL,
  position = c("topleft", "topright", "bottomleft", "bottomright"),
  polylineOptions = drawPolylineOptions(),
  polygonOptions = drawPolygonOptions(),
  circleOptions = drawCircleOptions(),
  rectangleOptions = drawRectangleOptions(),
  markerOptions = drawMarkerOptions(),
  circleMarkerOptions = drawCircleMarkerOptions(), editOptions = FALSE,
  singleFeature = FALSE)

removeDrawToolbar(map, clearFeatures = FALSE)
```

Arguments

`map`	The map widget.
`targetLayerId`	An optional layerId of a GeoJSON/TopoJSON layer whose features need to be editable. Used for adding a GeoJSON/TopoJSON layer and then editing the features using the draw plugin.
`targetGroup`	An optional group name of a Feature Group whose features need to be editable. Used for adding shapes(markers, lines, polygons) and then editing them using the draw plugin. You can either set layerId or group or none but not both.
`position`	The position where the toolbar should appear.
`polylineOptions`	See `drawPolylineOptions`(). Set to FALSE to disable polyline drawing.
`polygonOptions`	See `drawPolygonOptions`(). Set to FALSE to disable polygon drawing.
`circleOptions`	See `drawCircleOptions`(). Set to FALSE to disable circle drawing.
`rectangleOptions`	See `drawRectangleOptions`(). Set to FALSE to disable rectangle drawing.
`markerOptions`	See `drawMarkerOptions`(). Set to FALSE to disable marker drawing.

`circleMarkerOptions`	See `drawCircleMarkerOptions`(). Set to FALSE to disable circle marker drawing.
`editOptions`	By default editing is disable. To enable editing pass `editToolbarOptions`().
`singleFeature`	When set to TRUE, only one feature can be drawn at a time, the previous ones being removed.
`clearFeatures`	whether to clear the map of drawn features.

Examples

```
leaflet() %>%
  setView(0, 0, 2) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addDrawToolbar(
    targetGroup = "draw",
    editOptions = editToolbarOptions(
      selectedPathOptions = selectedPathOptions()
    )
  )  %>%
  addLayersControl(
    overlayGroups = c("draw"),
    options = layersControlOptions(collapsed = FALSE)
  ) %>%
  addStyleEditor()

## for more examples see
# browseURL(system.file("examples/draw.R", package = "leaflet.extras"))
```

`addFullscreenControl` *Add fullscreen control*

Description

Add a fullscreen control button

Usage

```
addFullscreenControl(map, position = "topleft", pseudoFullscreen = FALSE)
```

Arguments

`map`	The leaflet map
`position`	position of control: "topleft", "topright", "bottomleft", or "bottomright"
`pseudoFullscreen`	if true, fullscreen to page width and height

Examples

```
leaflet() %>%
  addTiles() %>%
  addFullscreenControl()
```

---

addGeodesicPolylines *Add Geodesic Lines*

Description

Add Geodesic Lines

Usage

```
addGeodesicPolylines(map, lng = NULL, lat = NULL, layerId = NULL,
  group = NULL, steps = 10, wrap = TRUE, stroke = TRUE,
  color = "#03F", weight = 5, opacity = 0.5, dashArray = NULL,
  smoothFactor = 1, noClip = FALSE, popup = NULL, popupOptions = NULL,
  label = NULL, labelOptions = NULL, options = pathOptions(),
  highlightOptions = NULL, data = getMapData(map))

addGreatCircles(map, lat_center = NULL, lng_center = NULL, radius,
  layerId = NULL, group = NULL, steps = 10, wrap = TRUE,
  stroke = TRUE, color = "#03F", weight = 5, opacity = 0.5,
  dashArray = NULL, smoothFactor = 1, noClip = FALSE, popup = NULL,
  popupOptions = NULL, label = NULL, labelOptions = NULL,
  options = pathOptions(), highlightOptions = NULL,
  data = getMapData(map))
```

Arguments

`map`	map object
`lng`	vector of longitudes
`lat`	vector of latitudes
`layerId`	the layer id
`group`	the name of the group this raster image should belong to (see
`steps`	Defines how many intermediate points are generated along the path. More steps mean a smoother path.
`wrap`	Wrap line at map border (date line). Set to "false" if you want lines to cross the dateline (experimental, see noWrap-example on how to use)
`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)

dashArray	a string that defines the stroke dash pattern
smoothFactor	how much to simplify the polyline on each zoom level
noClip	whether to disable polyline clipping (more means better performance and less accurate representation)
popup	a character vector of the HTML content for the popups (you are recommended to escape the text using htmlEscape())
popupOptions	A Vector of popupOptions to provide popups for security reasons)
label	a character vector of the HTML content for the labels
labelOptions	A Vector of labelOptions to provide label options for each label. Default NULL
options	a list of additional options, intended to be provided by a call to pathOptions()
highlightOptions	Options for highlighting the shape on mouse over.
data	map data
lat_center, lng_center	lat/lng for the center
radius	in meters

Functions

- addGreatCircles: Adds a Great Circle to the map

Examples

```
berlin <- c(52.51, 13.4)
losangeles <- c(34.05, -118.24)
santiago <- c(-33.44, -70.71)
tokio <- c(35.69, 139.69)
sydney <- c(-33.91, 151.08)
capetown <- c(-33.91, 18.41)
calgary <- c(51.05, -114.08)
hammerfest <- c(70.67, 23.68)
barrow <- c(71.29, -156.76)

df <- as.data.frame(rbind(hammerfest, calgary, losangeles, santiago, capetown, tokio, barrow))
names(df) <- c("lat","lng")

leaflet(df) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addGeodesicPolylines(lng = ~lng, lat = ~lat, weight = 2, color = "red",
                       steps = 50, opacity = 1) %>%
  addCircleMarkers(df, lat = ~lat,lng = ~lng, radius = 3, stroke = FALSE,
                   fillColor = "black", fillOpacity = 1)

## for more examples see
# browseURL(system.file("examples/geodesic.R", package = "leaflet.extras"))
```

`addGeoJSONv2` *Adds a GeoJSON/TopoJSON to the leaflet map.*

Description

This is a feature rich alternative to the `addGeoJSON` & `addTopoJSON` with options to map feature properties to labels, popups, colors, markers etc.

Options to customize a Choropleth Legend

Adds a GeoJSON/TopoJSON Choropleth.

Adds a KML to the leaflet map.

Adds a KML Choropleth.

Options for parsing CSV

Adds a CSV to the leaflet map.

Adds a GPX to the leaflet map.

Usage

```
addGeoJSONv2(map, geojson, layerId = NULL, group = NULL,
  markerType = NULL, markerIcons = NULL, markerIconProperty = NULL,
  markerOptions = leaflet::markerOptions(), clusterOptions = NULL,
  clusterId = NULL, labelProperty = NULL,
  labelOptions = leaflet::labelOptions(), popupProperty = NULL,
  popupOptions = leaflet::popupOptions(), stroke = TRUE, color = "#03F",
  weight = 5, opacity = 0.5, fill = TRUE, fillColor = color,
  fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE,
  pathOptions = leaflet::pathOptions(), highlightOptions = NULL)

legendOptions(title = NULL, position = c("bottomleft", "bottomright",
  "topleft", "topright"), locale = "en-US", numberFormatOptions = list(style
  = "decimal", maximumFractionDigits = 2))

addGeoJSONChoropleth(map, geojson, layerId = NULL, group = NULL,
  valueProperty, labelProperty = NULL,
  labelOptions = leaflet::labelOptions(), popupProperty = NULL,
  popupOptions = leaflet::popupOptions(), scale = c("white", "red"),
  steps = 5, mode = "q", channelMode = c("rgb", "lab", "hsl", "lch"),
  padding = NULL, correctLightness = FALSE, bezierInterpolate = FALSE,
  colors = NULL, stroke = TRUE, color = "#03F", weight = 1,
  opacity = 0.5, fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1,
  noClip = FALSE, pathOptions = leaflet::pathOptions(),
  highlightOptions = NULL, legendOptions = NULL)

addKML(map, kml, layerId = NULL, group = NULL, markerType = NULL,
  markerIcons = NULL, markerIconProperty = NULL,
  markerOptions = leaflet::markerOptions(), clusterOptions = NULL,
  clusterId = NULL, labelProperty = NULL,
  labelOptions = leaflet::labelOptions(), popupProperty = NULL,
  popupOptions = leaflet::popupOptions(), stroke = TRUE, color = "#03F",
  weight = 5, opacity = 0.5, fill = TRUE, fillColor = color,
  fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE,
  pathOptions = leaflet::pathOptions(), highlightOptions = NULL)

addKMLChoropleth(map, kml, layerId = NULL, group = NULL, valueProperty,
  labelProperty = NULL, labelOptions = leaflet::labelOptions(),
  popupProperty = NULL, popupOptions = leaflet::popupOptions(),
  scale = c("white", "red"), steps = 5, mode = "q",
  channelMode = c("rgb", "lab", "hsl", "lch"), padding = NULL,
  correctLightness = FALSE, bezierInterpolate = FALSE, colors = NULL,
  stroke = TRUE, color = "#03F", weight = 1, opacity = 0.5,
  fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE,
  pathOptions = leaflet::pathOptions(), highlightOptions = NULL,
  legendOptions = NULL)

csvParserOptions(latfield, lonfield, delimiter = ",")

addCSV(map, csv, csvParserOptions, layerId = NULL, group = NULL,
  markerType = NULL, markerIcons = NULL, markerIconProperty = NULL,
  markerOptions = leaflet::markerOptions(), clusterOptions = NULL,
  clusterId = NULL, labelProperty = NULL,
  labelOptions = leaflet::labelOptions(), popupProperty = NULL,
  popupOptions = leaflet::popupOptions(), stroke = TRUE, color = "#03F",
  weight = 5, opacity = 0.5, fill = TRUE, fillColor = color,
  fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE,
  pathOptions = leaflet::pathOptions(), highlightOptions = NULL)

addGPX(map, gpx, layerId = NULL, group = NULL, markerType = NULL,
  markerIcons = NULL, markerIconProperty = NULL,
  markerOptions = leaflet::markerOptions(), clusterOptions = NULL,
  clusterId = NULL, labelProperty = NULL,
  labelOptions = leaflet::labelOptions(), popupProperty = NULL,
  popupOptions = leaflet::popupOptions(), stroke = TRUE, color = "#03F",
  weight = 5, opacity = 0.5, fill = TRUE, fillColor = color,
  fillOpacity = 0.2, dashArray = NULL, smoothFactor = 1, noClip = FALSE,
  pathOptions = leaflet::pathOptions(), highlightOptions = NULL)
```

Arguments

`map`	the leaflet map widget
`geojson`	a GeoJSON/TopoJSON URL or file contents in a character vector.
`layerId`	the layer id
`group`	the name of the group this raster image should belong to (see the same parameter under `addTiles`)

`markerType`	The type of marker. either "marker" or "circleMarker"
`markerIcons`	Icons for Marker. Can be a single marker using `makeIcon` or a list of markers using `iconList`
`markerIconProperty`	The property of the feature to use for marker icon. Can be a JS function which accepts a feature and returns an index of `markerIcons`. In either case the result must be one of the indexes of markerIcons.
`markerOptions`	The options for markers
`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
`labelProperty`	The property to use for the label. You can also pass in a JS function that takes in a feature and returns a text/HTML content.
`labelOptions`	A Vector of `labelOptions` to provide label
`popupProperty`	The property to use for popup content You can also pass in a JS function that takes in a feature and returns a text/HTML content.
`popupOptions`	A Vector of `popupOptions` to provide popups
`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
`pathOptions`	Options for shapes
`highlightOptions`	Options for highlighting the shape on mouse over. options for each label. Default `NULL` you can use `highlightOptions()` to specify highlight options
`title`	An optional title for the legend
`position`	legend position
`locale`	The numbers will be formatted using this locale
`numberFormatOptions`	Options for formatting numbers
`valueProperty`	The property to use for coloring
`scale`	The scale to use from chroma.js
`steps`	number of breakes

`mode`	q for quantile, e for equidistant, k for k-means
`channelMode`	Default "rgb", can be one of "rgb", "lab", "hsl", "lch"
`padding`	either a single number or a 2 number vector for clipping color values at ends.
`correctLightness`	whether to correct lightness
`bezierInterpolate`	whether to use bezier interpolate for determining colors
`colors`	overrides scale with manual colors
`legendOptions`	Options to show a legend.
`kml`	a KML URL or contents in a character vector.
`latfield`	field name for latitude
`lonfield`	field name for longitude
`delimiter`	field separator
`csv`	a CSV URL or contents in a character vector.
`csvParserOptions`	options for parsing the CSV. Use `csvParserOptions()` to supply csv parser options.
`gpx`	a GPX URL or contents in a character vector.

Examples

```
## addGeoJSONv2
geoJson <- readr::read_file(
  "https://rawgit.com/benbalter/dc-maps/master/maps/historic-landmarks-points.geojson"
)

leaflet() %>%
  setView(-77.0369, 38.9072, 12) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addWebGLGeoJSONHeatmap(
    geoJson, size = 30 , units = "px"
  ) %>%
  addGeoJSONv2(
    geoJson,
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "black", fillOpacity = 0.7,
    markerOptions = markerOptions(radius = 2)
  )

## for more examples see
# browseURL(system.file("examples/draw.R", package = "leaflet.extras"))
# browseURL(system.file("examples/geojsonv2.R", package = "leaflet.extras"))
# browseURL(system.file("examples/search.R", package = "leaflet.extras"))
# browseURL(system.file("examples/TopoJSON.R", package = "leaflet.extras"))


## addGeoJSONChoropleth
```

```
geoJson <- readr::read_file(
  "https://rawgit.com/benbalter/dc-maps/master/maps/ward-2012.geojson"
)

leaflet() %>%
  addTiles() %>%
  setView(-77.0369, 38.9072, 11) %>%
  addBootstrapDependency() %>%
  enableMeasurePath() %>%
  addGeoJSONChoropleth(
    geoJson,
    valueProperty = "AREASQMI",
    scale = c("white", "red"),
    mode = "q",
    steps = 4,
    padding = c(0.2, 0),
    labelProperty = "NAME",
    popupProperty = propstoHTMLTable(
    props = c("NAME", "AREASQMI", "REP_NAME", "WEB_URL", "REP_PHONE", "REP_EMAIL", "REP_OFFICE"),
      table.attrs = list(class = "table table-striped table-bordered"),
      drop.na = TRUE
    ),
    color = "#ffffff", weight = 1, fillOpacity = 0.7,
    highlightOptions = highlightOptions(
      weight = 2, color = "#000000",
      fillOpacity = 1, opacity = 1,
      bringToFront = TRUE, sendToBack = TRUE),
    pathOptions = pathOptions(
      showMeasurements = TRUE,
      measurementOptions = measurePathOptions(imperial = TRUE)))

## for more examples see
# browseURL(system.file("examples/geojsonv2.R", package = "leaflet.extras"))
# browseURL(system.file("examples/measurePath.R", package = "leaflet.extras"))
# browseURL(system.file("examples/search.R", package = "leaflet.extras"))
# browseURL(system.file("examples/TopoJSON.R", package = "leaflet.extras"))


## addKML
kml <- readr::read_file(
  system.file("examples/data/kml/crimes.kml.zip", package = "leaflet.extras")
)

leaflet() %>%
  setView(-77.0369, 38.9072, 12) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addWebGLKMLHeatmap(kml, size = 20, units = "px") %>%
  addKML(
    kml,
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "black", fillOpacity = 1,
    markerOptions = markerOptions(radius = 1)
  )
```

```
## addKMLChoropleth
kml <- readr::read_file(
 system.file("examples/data/kml/cb_2015_us_state_20m.kml.zip", package = "leaflet.extras")
)

leaflet() %>%
  addBootstrapDependency() %>%
  setView(-98.583333, 39.833333, 4) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addKMLChoropleth(
    kml,
    valueProperty = JS(
      'function(feature){
         var props = feature.properties;
         var aland = props.ALAND/100000;
         var awater = props.AWATER/100000;
         return 100*awater/(awater+aland);
      }'
    ),
    scale = "OrRd", mode = "q", steps = 5,
    padding = c(0.2, 0),
    popupProperty = "description",
    labelProperty = "NAME",
    color = "#ffffff", weight = 1, fillOpacity = 1,
    highlightOptions = highlightOptions(
      fillOpacity = 1, weight = 2, opacity = 1, color = "#000000",
      bringToFront = TRUE, sendToBack = TRUE
    ),
    legendOptions = legendOptions(
      title = "% of Water Area",
      numberFormatOptions = list(
        style = "decimal",
        maximumFractionDigits = 2
      )
    )
  )


## addCSV
csv <- readr::read_file(
  system.file("examples/data/csv/world_airports.csv.zip", package = "leaflet.extras")
)

leaflet() %>%
  setView(0, 0, 2) %>%
  addProviderTiles(providers$CartoDB.DarkMatterNoLabels) %>%
  addCSV(
    csv,
    csvParserOptions("latitude_deg", "longitude_deg"),
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "red", fillOpacity = 1,
    markerOptions = markerOptions(radius = 0.5))


## addGPX
airports <- readr::read_file(
  system.file("examples/data/gpx/md-airports.gpx.zip", package = "leaflet.extras")
)

leaflet() %>%
  addBootstrapDependency() %>%
  setView(-76.6413, 39.0458, 8) %>%
  addProviderTiles(
    providers$CartoDB.Positron,
    options = providerTileOptions(detectRetina = TRUE)
  ) %>%
  addWebGLGPXHeatmap(airports, size = 20000, group = "airports", opacity = 0.9) %>%
  addGPX(
    airports,
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "black", fillOpacity = 1,
    markerOptions = markerOptions(radius = 1.5),
    group = "airports"
  )

## for a larger example see
# browseURL(system.file("examples/GPX.R", package = "leaflet.extras"))
```

addHash *Add dynamic URL Hash*

Description

Leaflet-hash lets you to add dynamic URL hashes to web pages with Leaflet maps. You can easily link users to specific map views.

Usage

```
addHash(map)
```

Arguments

map	The leaflet map

Examples

```
leaflet() %>%
  addTiles() %>%
  addHash()
```

addHeatmap *Add a heatmap*

Description

Add a heatmap

Adds a heatmap with data from a GeoJSON/TopoJSON file/url

Adds a heatmap with data from a KML file/url

Adds a heatmap with data from a CSV file/url

Adds a heatmap with data from a GPX file/url

removes the heatmap

clears the heatmap

Usage

```
addHeatmap(map, lng = NULL, lat = NULL, intensity = NULL,
  layerId = NULL, group = NULL, minOpacity = 0.05, max = 1,
  radius = 25, blur = 15, gradient = NULL, cellSize = NULL,
  data = leaflet::getMapData(map))

addGeoJSONHeatmap(map, geojson, layerId = NULL, group = NULL,
  intensityProperty = NULL, minOpacity = 0.05, max = 1, radius = 25,
  blur = 15, gradient = NULL, cellSize = NULL)

addKMLHeatmap(map, kml, layerId = NULL, group = NULL,
  intensityProperty = NULL, minOpacity = 0.05, max = 1, radius = 25,
  blur = 15, gradient = NULL, cellSize = NULL)

addCSVHeatmap(map, csv, csvParserOptions, layerId = NULL, group = NULL,
  intensityProperty = NULL, minOpacity = 0.05, max = 1, radius = 25,
  blur = 15, gradient = NULL, cellSize = NULL)

addGPXHeatmap(map, gpx, layerId = NULL, group = NULL,
  intensityProperty = NULL, minOpacity = 0.05, max = 1, radius = 25,
  blur = 15, gradient = NULL, cellSize = NULL)

removeHeatmap(map, layerId)

clearHeatmap(map)
```

Arguments

`map`	the map widget.

`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`)
`intensity`	intensity of the heat. A vector of numeric values or a formula.
`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.
`minOpacity`	minimum opacity at which the heat will start
`max`	maximum point intensity. The default is `1.0`
`radius`	radius of each "point" of the heatmap. The default is `25`.
`blur`	amount of blur to apply. The default is `15`. `blur=1` means no blur.
`gradient`	palette name from `RColorBrewer` or an array of of colors to be provided to `colorNumeric`, or a color mapping function returned from `colorNumeric`
`cellSize`	the cell size in the grid. Points which are closer than this may be merged. Defaults to ‘radius / 2‘.s Set to ‘1‘ to do almost no merging.
`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
`geojson`	The geojson or topojson url or contents as string.
`intensityProperty`	The property to use for determining the intensity at a point. Can be a "string" or a JS function, or NULL.
`kml`	The KML url or contents as string.
`csv`	The CSV url or contents as string.
`csvParserOptions`	options for parsing the CSV. Use `csvParserOptions`() to supply csv parser options.
`gpx`	The GPX url or contents as string.

Examples

```
leaflet(quakes) %>%
  addProviderTiles(providers$CartoDB.DarkMatter) %>%
  setView( 178, -20, 5 ) %>%
  addHeatmap(
    lng = ~long, lat = ~lat, intensity = ~mag,
    blur = 20, max = 0.05, radius = 15
  )
```

```
## for more examples see
# browseURL(system.file("examples/heatmaps.R", package = "leaflet.extras"))
kml <- readr::read_file(
  system.file("examples/data/kml/crimes.kml.zip", package = "leaflet.extras")
)

leaflet() %>%
  setView(-77.0369, 38.9072, 12) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addKMLHeatmap(kml, radius = 7) %>%
  addKML(
    kml,
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "black", fillOpacity = 1,
    markerOptions = markerOptions(radius = 1))

## for more examples see
# browseURL(system.file("examples/KML.R", package = "leaflet.extras"))
```

addResetMapButton *Reset map's view to original view*

Description

Reset map's view to original view

Usage

```
addResetMapButton(map)
```

Arguments

map	The map widget

Examples

```
leaflet() %>%
  addTiles() %>%
  addResetMapButton()
```

`addSearchFeatures` *Add a feature search control to the map.*

Description

Add a feature search control to the map.

Removes the feature search control from the map.

Usage

```
addSearchFeatures(map, targetGroups, options = searchFeaturesOptions())

removeSearchFeatures(map, clearFeatures = FALSE)
```

Arguments

`map`	a map widget object
`targetGroups`	A vector of group names of groups whose features need to be searched.
`options`	Search Options
`clearFeatures`	Boolean. If TRUE the features that this control searches will be removed too.

Value

modified map

modified map

`addSearchOSM` *Add a OSM search control to the map.*

Description

Add a OSM search control to the map.

Removes the OSM search control from the map.

Add a Google search control to the map.

Removes the Google search control from the map.

Add a US Census Bureau search control to the map.

Removes the US Census Bureau search control from the map.

Usage

```
addSearchOSM(map, options = searchOptions(autoCollapse = TRUE, minLength = 2))

removeSearchOSM(map)

addReverseSearchOSM(map, showSearchLocation = TRUE, showBounds = FALSE,
  showFeature = TRUE, fitBounds = TRUE, displayText = TRUE,
  group = NULL)

addSearchGoogle(map, apikey = Sys.getenv("GOOGLE_MAP_GEOCODING_KEY"),
  options = searchOptions(autoCollapse = TRUE, minLength = 2))

removeSearchGoogle(map)

addReverseSearchGoogle(map, apikey = Sys.getenv("GOOGLE_MAP_GEOCODING_KEY"),
  showSearchLocation = TRUE, showBounds = FALSE, showFeature = TRUE,
  fitBounds = TRUE, displayText = TRUE, group = NULL)

addSearchUSCensusBureau(map, options = searchOptions(autoCollapse = TRUE,
  minLength = 20))

removeSearchUSCensusBureau(map)
```

Arguments

`map`	a map widget object
`options`	Search Options
`showSearchLocation`	Boolean. If TRUE displays a Marker on the searched location's coordinates.
`showBounds`	Boolean. If TRUE show the bounding box of the found feature.
`showFeature`	Boolean. If TRUE show the found feature. Depending upon the feature found this can be a marker, a line or a polygon.
`fitBounds`	Boolean. If TRUE set maps bounds to queried and found location. For this to be effective one of `showSearchLocation`, `showBounds`, `showFeature` shoule also be TRUE.
`displayText`	Boolean. If TRUE show a text box with found location's name on the map.
`group`	String. An optional group to hold all the searched locations and their results.
`apikey`	String. API Key for Google GeoCoding Service.

Value

modified map

modified map

modified map

modified map

modified map

modified map

modified map

modified map

Examples

```
leaflet() %>%
  addProviderTiles(providers$Esri.WorldStreetMap) %>%
  addResetMapButton() %>%
  addSearchGoogle()

## for more examples see
# browseURL(system.file("examples/search.R", package = "leaflet.extras"))
```

`addStyleEditor` *Add style editor*

Description

Add style editor

Remove style editor

Usage

```
addStyleEditor(map, position = c("topleft", "topright", "bottomleft",
  "bottomright"), openOnLeafletDraw = TRUE, useGrouping = FALSE, ...)

removeStyleEditor(map)
```

Arguments

`map`	the map widget
`position`	position of the control
`openOnLeafletDraw`	whether to open automatically when used with `addDrawToolbar`()
`useGrouping`	Should be false to work with `addDrawToolbar`()
`...`	other options. See plugin code

Examples

```
leaflet() %>%
  setView(0, 0, 2) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addDrawToolbar(
    targetGroup = "draw",
    editOptions = editToolbarOptions(selectedPathOptions = selectedPathOptions())
  ) %>%
  addLayersControl(
    overlayGroups = c("draw"), options = layersControlOptions(collapsed = FALSE)
  ) %>%
  # add the style editor to alter shapes added to map
  addStyleEditor()
```

`addWebGLHeatmap` *Add a webgl heatmap*

Description

Add a webgl heatmap

Adds a heatmap with data from a GeoJSON/TopoJSON file/url

Adds a heatmap with data from a KML file/url

Adds a heatmap with data from a CSV file/url

Adds a heatmap with data from a GPX file/url

removes the webgl heatmap

clears the webgl heatmap

Usage

```
addWebGLHeatmap(map, lng = NULL, lat = NULL, intensity = NULL,
  layerId = NULL, group = NULL, size = "30000", units = "m",
  opacity = 1, gradientTexture = NULL, alphaRange = 1,
  data = leaflet::getMapData(map))

addWebGLGeoJSONHeatmap(map, geojson, layerId = NULL, group = NULL,
  intensityProperty = NULL, size = "30000", units = "m", opacity = 1,
  gradientTexture = NULL, alphaRange = 1)

addWebGLKMLHeatmap(map, kml, layerId = NULL, group = NULL,
  intensityProperty = NULL, size = "30000", units = "m", opacity = 1,
  gradientTexture = NULL, alphaRange = 1)

addWebGLCSVHeatmap(map, csv, csvParserOptions, layerId = NULL, group = NULL,
  intensityProperty = NULL, size = "30000", units = "m", opacity = 1,
  gradientTexture = NULL, alphaRange = 1)
```

```
addWebGLGPXHeatmap(map, gpx, layerId = NULL, group = NULL,
  intensityProperty = NULL, size = "30000", units = "m", opacity = 1,
  gradientTexture = NULL, alphaRange = 1)

removeWebGLHeatmap(map, layerId)

clearWebGLHeatmap(map)
```

Arguments

`map`	the map to add pulse 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`)
`intensity`	intensity of the heat. A vector of numeric values or a formula.
`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.
`size`	in meters or pixels
`units`	either "m" or "px"
`opacity`	for the canvas element
`gradientTexture`	Alternative colors for heatmap. allowed values are "skyline", "deep-sea"
`alphaRange`	adjust transparency by changing to value between 0 and 1
`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
`geojson`	The geojson or topojson url or contents as string.
`intensityProperty`	The property to use for determining the intensity at a point. Can be a "string" or a JS function, or NULL.
`kml`	The KML url or contents as string.
`csv`	The CSV url or contents as string.
`csvParserOptions`	options for parsing the CSV. Use `csvParserOptions`() to supply csv parser options.
`gpx`	The GPX url or contents as string.

Examples

```
## addWebGLHeatmap
leaflet(quakes) %>%
  addProviderTiles(providers$CartoDB.DarkMatter) %>%
  addWebGLHeatmap(lng = ~long, lat = ~lat, size = 60000)

## for more examples see
# browseURL(system.file("examples/webglHeatmaps.R", package = "leaflet.extras"))
## addWebGLGeoJSONHeatmap
geoJson <- readr::read_file(
  "https://rawgit.com/benbalter/dc-maps/master/maps/historic-landmarks-points.geojson"
)

leaflet() %>%
  setView(-77.0369, 38.9072, 12) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addWebGLGeoJSONHeatmap(
    geoJson, size = 30 , units = "px"
  ) %>%
  addGeoJSONv2(
    geoJson,
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "black", fillOpacity = 0.7,
    markerOptions = markerOptions(radius = 2)
  )

## for more examples see
# browseURL(system.file("examples/geojsonV2.R", package = "leaflet.extras"))
# browseURL(system.file("examples/TopoJSON.R", package = "leaflet.extras"))
## addWebGLKMLHeatmap
kml <- readr::read_file(
  system.file("examples/data/kml/crimes.kml.zip", package = "leaflet.extras")
)

leaflet() %>% setView(-77.0369, 38.9072, 12) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addWebGLKMLHeatmap(kml, size = 20, units = "px") %>%
  addKML(
    kml,
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "black", fillOpacity = 1,
    markerOptions = markerOptions(radius = 1))


## addWebGLCSVHeatmap
csv <- readr::read_file(
  system.file("examples/data/csv/world_airports.csv.zip", package = "leaflet.extras")
)

leaflet() %>%
  setView(0, 0, 2) %>%
  addProviderTiles(providers$CartoDB.DarkMatterNoLabels) %>%
  addWebGLCSVHeatmap(
    csv,
    csvParserOptions("latitude_deg", "longitude_deg"),
    size = 10, units = "px")


airports <- readr::read_file(
  system.file("examples/data/gpx/md-airports.gpx.zip", package = "leaflet.extras")
)

leaflet() %>%
  addBootstrapDependency() %>%
  setView(-76.6413, 39.0458, 8) %>%
  addProviderTiles(
    providers$CartoDB.Positron,
    options = providerTileOptions(detectRetina = TRUE)
  ) %>%
  addWebGLGPXHeatmap(
    airports,
    size = 20000,
    group = "airports",
    opacity = 0.9
  ) %>%
  addGPX(
    airports,
    markerType = "circleMarker",
    stroke = FALSE, fillColor = "black", fillOpacity = 1,
    markerOptions = markerOptions(radius = 1.5),
    group = "airports"
  )


## for a larger example see
# browseURL(system.file("examples/GPX.R", package = "leaflet.extras"))
```

`addWMSLegend` *Add WMS Legend*

Description

Add a WMS Legend

Usage

```
addWMSLegend(map, uri, position = "topright", layerId = NULL)
```

Arguments

`map`	The leaflet map
`uri`	The legend URI

position	position of control: "topleft", "topright", "bottomleft", or "bottomright"
layerId	A unique ID for the Legend

Examples

```
leaflet(
  options = leafletOptions(
    center = c(-33.95293, 20.82824),
    zoom = 14,
    minZoom = 5,
    maxZoom = 18,
    maxBounds = list(
      c(-33.91444, 20.75351),
      c(-33.98731, 20.90626)
    )
  )
) %>%
  addWMSTiles(
    baseUrl = paste0(
      "http://maps.kartoza.com/web/?",
      "map=/web/Boosmansbos/Boosmansbos.qgs"
    ),
    layers = "Boosmansbos",
    options = WMSTileOptions(format = "image/png", transparent = TRUE),
    attribution = paste0(
      "(c)<a href= \"http://kartoza.com\">Kartoza.com</a> and ",
      "<a href= \"http://www.ngi.gov.za/\">SA-NGI</a>"
    )
  ) %>%
  addWMSLegend(
    uri = paste0(
      "http://maps.kartoza.com/web/?",
      "map=/web/Boosmansbos/Boosmansbos.qgs&&SERVICE=WMS&VERSION=1.3.0",
    "&SLD_VERSION=1.1.0&REQUEST=GetLegendGraphic&FORMAT=image/jpeg&LAYER=Boosmansbos&STYLE="
    )
  )
```

debugMap *For debugging a leaflet map*

Description

For debugging a leaflet map

Usage

```
debugMap(map)
```

Arguments

map	The map widget

`drawShapeOptions` *Options for drawn shapes*

Description

Options for drawn shapes

Options for drawing polylines

Options for drawing polygons

Options for drawing rectangles

Options for drawing Circles

Options for drawing markers

Options for drawing markers

Options for path when in editMode

Options for editing shapes

Usage

```
drawShapeOptions(stroke = TRUE, color = "#03f", weight = 1, opacity = 1,
  fill = TRUE, fillColor = "#03f", fillOpacity = 0.4, dashArray = NULL,
  lineCap = NULL, lineJoin = NULL, clickable = TRUE,
  pointerEvents = NULL, smoothFactor = 1, noClip = TRUE)

drawPolylineOptions(allowIntersection = TRUE, drawError = list(color =
  "#b00b00", timeout = 2500), guidelineDistance = 20,
  maxGuideLineLength = 4000, showLength = TRUE, metric = TRUE,
  feet = TRUE, nautic = FALSE, zIndexOffset = 2000,
  shapeOptions = drawShapeOptions(fill = FALSE), repeatMode = FALSE)

drawPolygonOptions(showArea = FALSE, metric = TRUE,
  shapeOptions = drawShapeOptions(), repeatMode = FALSE)

drawRectangleOptions(showArea = TRUE, metric = TRUE,
  shapeOptions = drawShapeOptions(), repeatMode = FALSE)

drawCircleOptions(showRadius = TRUE, metric = TRUE, feet = TRUE,
  nautic = FALSE, shapeOptions = drawShapeOptions(), repeatMode = FALSE)

drawMarkerOptions(markerIcon = NULL, zIndexOffset = 2000,
  repeatMode = FALSE)

drawCircleMarkerOptions(stroke = TRUE, color = "#3388ff", weight = 4,
  opacity = 0.5, fill = TRUE, fillColor = NULL, fillOpacity = 0.2,
  clickable = TRUE, zIndexOffset = 2000, repeatMode = FALSE)
```

```
selectedPathOptions(dashArray = c("10, 10"), weight = 2, color = "black",
  fill = TRUE, fillColor = "black", fillOpacity = 0.6,
  maintainColor = FALSE)

editToolbarOptions(edit = TRUE, remove = TRUE, selectedPathOptions = NULL,
  allowIntersection = TRUE)
```

Arguments

`stroke`	Whether to draw stroke along the path. Set it to false to disable borders on polygons or circles.
`color`	Stroke color.
`weight`	Stroke width in pixels.
`opacity`	Stroke opacity.
`fill`	Whether to fill the path with color. Set it to false to disable filling on polygons or circles.
`fillColor`	same as color Fill color.
`fillOpacity`	Fill opacity.
`dashArray`	A string that defines the stroke dash pattern. Doesn't work on canvas-powered layers (e.g. Android 2).
`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.
`clickable`	If false, the vector will not emit mouse events and will act as a part of the underlying map.
`pointerEvents`	Sets the pointer-events attribute on the path if SVG backend is used.
`smoothFactor`	How much to simplify the polyline on each zoom level. More means better performance and smoother look, and less means more accurate representation.
`noClip`	Disabled polyline clipping.
`allowIntersection`	Determines if line segments can cross.
`drawError`	Configuration options for the error that displays if an intersection is detected.
`guidelineDistance`	Distance in pixels between each guide dash.
`maxGuideLineLength`	Maximum length of the guide lines.
`showLength`	Whether to display the distance in the tooltip.
`metric`	Determines which measurement system (metric or imperial) is used.
`feet`	When not metric, use feet instead of yards for display.
`nautic`	When not metric, not feet, use nautic mile for display.
`zIndexOffset`	This should be a high number to ensure that you can draw over all other layers on the map.
`shapeOptions`	Leaflet Polyline options See `drawShapeOptions()`.

`repeatMode`	Determines if the draw tool remains enabled after drawing a shape.
`showArea`	Show the area of the drawn polygon in m², ha or km². The area is only approximate and become less accurate the larger the polygon is.
`showRadius`	Show the radius of the drawn circle in m, km, ft (feet), or nm (nautical mile).
`markerIcon`	Can be either `makeIcon`() OR `makeAwesomeIcon`
`maintainColor`	Whether to maintain shape's original color
`edit`	Editing enabled by default. Set to false do disable editing.
`remove`	Set to false to disable removing.
`selectedPathOptions`	To customize shapes in editing mode pass `selectedPathOptions`().

`enableMeasurePath` *Enables measuring of length of polylines and areas of polygons*

Description

Enables measuring of length of polylines and areas of polygons

Options for measure-path

Adds a toolbar to enable/disable measuing path distances/areas

Usage

```
enableMeasurePath(map)

measurePathOptions(showOnHover = FALSE, minPixelDistance = 30,
  showDistances = TRUE, showArea = TRUE, imperial = FALSE)

addMeasurePathToolbar(map, options = measurePathOptions())
```

Arguments

`map`	The map widget.
`showOnHover`	If TRUE, the measurements will only show when the user hovers the cursor over the path.
`minPixelDistance`	The minimum length a line segment in the feature must have for a measurement to be added.
`showDistances`	If FALSE, doesn't show distances along line segments of of a polyline/polygon.
`showArea`	If FALSE, doesn't show areas of a polyline/polygon.
`imperial`	If TRUE the distances/areas will be shown in imperial units.
`options`	The measurePathOptions.

Examples

```
geoJson <- readr::read_file(
  "https://rawgit.com/benbalter/dc-maps/master/maps/ward-2012.geojson"
)

leaflet() %>%
  addTiles() %>%
  setView(-77.0369, 38.9072, 11) %>%
  addBootstrapDependency() %>%
  enableMeasurePath() %>%
  addGeoJSONChoropleth(
    geoJson,
    valueProperty = "AREASQMI",
    scale = c("white","red"),
    mode = "q",
    steps = 4,
    padding = c(0.2,0),
    labelProperty = "NAME",
    popupProperty = propstoHTMLTable(
    props = c("NAME", "AREASQMI", "REP_NAME", "WEB_URL", "REP_PHONE", "REP_EMAIL", "REP_OFFICE"),
      table.attrs = list(class = "table table-striped table-bordered"),
      drop.na = TRUE
    ),
    color = "#ffffff", weight = 1, fillOpacity = 0.7,
    highlightOptions = highlightOptions(
      weight = 2, color = "#000000",
      fillOpacity = 1, opacity = 1,
      bringToFront = TRUE, sendToBack = TRUE),
    pathOptions = pathOptions(
      showMeasurements = TRUE,
      measurementOptions = measurePathOptions(imperial = TRUE)))
```

`enableTileCaching` *Enables caching of Tiles*

Description

Enables caching of tiles locally in browser. See `https://github.com/MazeMap/Leaflet.TileLayer.PouchDBCached` for details. In addition to invoking this function, you should also pass useCache=TRUE & `crossOrigin=TRUE` in the `tileOptions` call and pass that to your `addTiles`'s options parameter.

Usage

```
enableTileCaching(map)
```

Arguments

map	The leaflet map

Examples

```
leaflet() %>%
  enableTileCaching() %>%
  addTiles(options = tileOptions(useCache = TRUE, crossOrigin = TRUE))

## for more examples see
# browseURL(system.file("examples/TileLayer-Caching.R", package = "leaflet.extras"))
```

---

gpsOptions *Options for the GPS Control*

Description

Options for the GPS Control

Add a gps to the Map.

Removes the GPS Control

Activate the GPS Control. You should have already added the GPS control before calling this method.

Deactivate the GPS Control. You should have already added the GPS control before calling this method.

Usage

```
gpsOptions(position = "topleft", activate = FALSE, autoCenter = FALSE,
  maxZoom = NULL, setView = FALSE)

addControlGPS(map, options = gpsOptions())

removeControlGPS(map)

activateGPS(map)

deactivateGPS(map)
```

Arguments

position	Position of the Control
activate	If TRUE activates the GPS on addition.
autoCenter	If TRUE auto centers the map when GPS location changes
maxZoom	If set zooms to this level when auto centering
setView	If TRUE sets the view to the GPS location when found
map	a map widget object
options	Options for the GPS control.

Examples

```
leaflet() %>%
  addTiles() %>%
  addControlGPS()
```

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

Description

Description: The 'leaflet' JavaScript library provides many plugins some of which are available in the core 'leaflet' package, but there are many more. It is not possible to support them all in the core 'leaflet' package. This package serves as an add-on to the 'leaflet' package by providing extra functionality via 'leaflet' plugins.

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

Description

Various leaflet dependency functions for use in downstream packages

Usage

```
leafletExtrasDependencies
```

Format

An object of class `list` of length 5.

propsToHTML — *Converts GeoJSON Feature properties to HTML*

Description

Converts GeoJSON Feature properties to HTML

Converts GeoJSON Feature properties to HTML Table.

Customize the leaflet widget style

Usage

```
propsToHTML(props, elem = NULL, elem.attrs = NULL)

propstoHTMLTable(props = NULL, table.attrs = NULL, drop.na = TRUE)

setMapWidgetStyle(map, style = list(background = "transparent"))
```

Arguments

`props`	A list of GeoJSON Property Keys.
`elem`	An optional wrapping element e.g. "div".
`elem.attrs`	An optional named list for the wrapper element properties.
`table.attrs`	An optional named list for the HTML Table.
`drop.na`	whether to skip properties with empty values.
`map`	the map widget
`style`	a A list of CSS key/value properties.

Examples

```
geoJson <- jsonlite::fromJSON(readr::read_file(
  paste0(
    "https://raw.githubusercontent.com/MinnPost/simple-map-d3",
    "/master/example-data/world-population.geo.json"
  )
))

world <- leaflet(
  options = leafletOptions(
    maxZoom = 5,
    crs = leafletCRS(
      crsClass = "L.Proj.CRS", code = "ESRI:53009",
    proj4def = "+proj=moll +lon_0=0 +x_0=0 +y_0=0 +a=6371000 +b=6371000 +units=m +no_defs",
      resolutions = c(65536, 32768, 16384, 8192, 4096, 2048)))) %>%
  addGraticule(style = list(color = "#999", weight = 0.5, opacity = 1, fill = NA)) %>%
 addGraticule(sphere = TRUE, style = list(color = "#777", weight = 1, opacity = 0.25, fill = NA))
```

```
world

# change background to white
world %>%
  setMapWidgetStyle(list(background = "white"))
```

`pulseIconList` *Make pulse-icon set*

Description

Make pulse-icon set

Make Pulse Icon

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.

Add Pulse Markers

Usage

```
pulseIconList(...)

## S3 method for class 'leaflet_pulse_icon_set'
x[i]

makePulseIcon(color = "#ff0000", iconSize = 12, animate = TRUE,
  heartbeat = 1)

pulseIcons(color = "#ff0000", iconSize = 12, animate = TRUE,
  heartbeat = 1)

addPulseMarkers(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

`...`	icons created from `makePulseIcon()`
`x`	icons
`i`	offset
`color`	Color of the icon
`iconSize`	Size of Icon in Pixels.

`animate`	To animate the icon or not, defaults to TRUE.
`heartbeat`	Interval between each pulse in seconds.
`map`	the map to add pulse 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`	options for popup
`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

Examples

```
iconSet = pulseIconList(
  red = makePulseIcon(color = "#ff0000"),
  blue = makePulseIcon(color = "#0000ff")
)

iconSet[c("red", "blue")]

leaflet() %>%
  addTiles() %>%
  addPulseMarkers(
    lng = -118.456554, lat = 34.078039,
    label = "This is a label",
    icon = makePulseIcon(heartbeat = 0.5)
)


## for more examples see
# browseURL(system.file("examples/pluseIcon.R", package = "leaflet.extras"))
```

---

searchOptions *Options for search control.*

Description

Options for search control.

Customized searchOptions for Feature Search

Usage

```
searchOptions(url = "", sourceData = NULL, jsonpParam = NULL,
  propertyLoc = "loc", propertyName = "title", formatData = NULL,
  filterData = NULL, moveToLocation = TRUE, zoom = 17, buildTip = NULL,
  container = "", minLength = 1, initial = TRUE, casesensitive = FALSE,
  autoType = TRUE, delayType = 400, tooltipLimit = -1,
  tipAutoSubmit = TRUE, firstTipSubmit = FALSE, autoResize = TRUE,
  collapsed = TRUE, autoCollapse = FALSE, autoCollapseTime = 1200,
  textErr = "Location Not Found", textCancel = "Cancel",
  textPlaceholder = "Search...", position = "topleft",
  hideMarkerOnCollapse = FALSE)

searchFeaturesOptions(propertyName = "label", initial = FALSE,
  openPopup = FALSE, ...)
```

Arguments

`url`	url for search by ajax request, ex: "search.php?q=s". Can be function that returns string for dynamic parameter setting.
`sourceData`	function that fill _recordsCache, passed searching text by first param and callback in second.
`jsonpParam`	jsonp param name for search by jsonp service, ex: "callback".
`propertyLoc`	field for remapping location, using array: ["latname","lonname"] for select double fields(ex. ["lat","lon"]) support dotted format: "prop.subprop.title".
`propertyName`	property in marker.options(or feature.properties for vector layer) trough filter elements in layer,.
`formatData`	callback for reformat all data from source to indexed data object.
`filterData`	callback for filtering data from text searched, params: textSearch, allRecords.
`moveToLocation`	whether to move to the found location.
`zoom`	zoom to this level when moving to location

`buildTip`	function that return row tip html node(or html string), receive text tooltip in first param.
`container`	container id to insert Search Control.
`minLength`	minimal text length for autocomplete.
`initial`	search elements only by initial text.
`casesensitive`	search elements in case sensitive text.
`autoType`	complete input with first suggested result and select this filled-in text..
`delayType`	delay while typing for show tooltip.
`tooltipLimit`	limit max results to show in tooltip. -1 for no limit..
`tipAutoSubmit`	auto map panTo when click on tooltip.
`firstTipSubmit`	auto select first result con enter click.
`autoResize`	autoresize on input change.
`collapsed`	collapse search control at startup.
`autoCollapse`	collapse search control after submit(on button or on tips if enabled tipAutoSubmit).
`autoCollapseTime`	delay for autoclosing alert and collapse after blur.
`textErr`	'Location not error message.
`textCancel`	title in cancel button.
`textPlaceholder`	placeholder value.
`position`	"topleft".
`hideMarkerOnCollapse`	remove circle and marker on search control collapsed.
`openPopup`	whether to open the popup associated with the feature when the feature is searched for
`...`	Other options to pass to `searchOptions()` function.

`suspendScroll` *Prevents accidental map scrolling when scrolling in a document.*

Description

Prevents accidental map scrolling when scrolling in a document.

Usage

```
suspendScroll(map, sleep = TRUE, sleepTime = 750, wakeTime = 750,
  sleepNote = TRUE, hoverToWake = TRUE,
  wakeMessage = "Click or Hover to Wake", sleepOpacity = 0.7)
```

Arguments

`map`	The leaflet map
`sleep`	false if you want an unruly map
`sleepTime`	time(ms) until map sleeps on mouseout
`wakeTime`	time(ms) until map wakes on mouseover
`sleepNote`	should the user receive wake instructions?
`hoverToWake`	should hovering wake the map? (non-touch devices only)
`wakeMessage`	a message to inform users about waking the map
`sleepOpacity`	opacity for the sleeping map

Examples

```
leaflet(width = "100%") %>%
  setView(0, 0, 1) %>%
  addTiles() %>%
  suspendScroll()
```

`weatherIconList` *Make weather-icon set*

Description

Make weather-icon set

Make Weather Icon

An icon can be represented as a list of the form `list(icon, markerColor,...)`. 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.

Add Weather Markers

Usage

```
weatherIconList(...)

## S3 method for class 'leaflet_weather_icon_set'
x[i]

makeWeatherIcon(icon, markerColor = "red", iconColor = "white",
  extraClasses = NULL)

weatherIcons(icon, markerColor = "red", iconColor = "white",
  extraClasses = NULL)

addWeatherMarkers(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

`...`	icons created from `makeWeatherIcon()`
`x`	icons
`i`	offset
`icon`	the weather icon name w/o the "wi-" prefix. For a full list see `https://erikflowers.github.io/weather-icons/`
`markerColor`	color of the marker
`iconColor`	color of the weather icon
`extraClasses`	Character vector of extra classes.
`map`	the map to add weather 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.
`popup`	a character vector of the HTML content for the popups (you are recommended to escape the text using `htmlEscape()` for security reasons)
`popupOptions`	options for popup
`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

Examples

```
iconSet = weatherIconList(
  hurricane = makeWeatherIcon(icon = "hurricane"),
  tornado = makeWeatherIcon(icon = "tornado")
)

iconSet[c("hurricane", "tornado")]
leaflet() %>%
  addTiles() %>%
  addWeatherMarkers(
    lng = -118.456554, lat = 34.078039,
    label = "This is a label",
    icon = makeWeatherIcon(
      icon = "hot",
      iconColor = "#ffffff77",
      markerColor = "blue"
    )
  )

## for more examples see
# browseURL(system.file("examples/weatherIcons.R", package = "leaflet.extras"))
```

Index