Package 'leafem'

August 28, 2025

Title 'leaflet' Extensions for 'mapview'

Version 0.2.5

Maintainer Tim Appelhans <tim.appelhans@gmail.com>

Description Provides extensions for packages 'leaflet' & 'mapdeck',
many of which are used by package 'mapview'.
Focus is on functionality readily available in
Geographic Information Systems such as 'Quantum GIS'. Includes functions
to display coordinates of mouse pointer position, query image values via
mouse pointer and zoom-to-layer buttons. Additionally, provides a feature
type agnostic function to add points, lines, polygons to a map.

License MIT + file LICENSE

URL https://github.com/r-spatial/leafem,
https://r-spatial.github.io/leafem/

BugReports https://github.com/r-spatial/leafem/issues

Depends R (>= 3.1.0)

Imports base64enc, geojsonsf, htmltools (>= 0.3), htmlwidgets,
jsonlite, leaflet (>= 2.0.1), methods, raster, sf, png

Suggests clipr, fontawesome, leafgl, lwgeom, mapdeck, plainview,
stars, terra, tools

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Tim Appelhans [cre, aut],
Christoph Reudenbach [ctb],
Kenton Russell [ctb],
Jochen Darley [ctb],
Daniel Montague [ctb, cph] (Leaflet.EasyButton plugin),
Lorenzo Busetto [ctb],
Luigi Ranghetti [ctb],
Miles McBain [ctb],
Sebastian Gatscha [ctb],

Björn Harrtell [ctb, cph] (FlatGeobuf plugin),
Daniel Dufour [ctb, cph] (georaster-layer-for-leaflet),
Yeedle Neuwirth [ctb],
Derek Friend [ctb],
Kevin Cazelles [ctb] (ORCID: <https://orcid.org/0000-0001-6619-9874>)

Repository CRAN

Date/Publication 2025-08-28 12:30:02 UTC

Contents

addCOG . . . 2
addCopyExtent . . . 4
addExtent . . . 5
addFeatures . . . 6
addFgb . . . 7
addGeoRaster . . . 9
addGeotiff . . . 11
addHomeButton . . . 13
addImageQuery . . . 14
addLocalFile . . . 16
addLogo . . . 18
addMouseCoordinates . . . 19
addPMPolygons . . . 21
addRasterRGB . . . 24
addReactiveFeatures . . . 26
addStarsImage . . . 28
addStaticLabels . . . 30
addTileFolder . . . 31
colorOptions . . . 32
customizeLayersControl . . . 32
garnishMap . . . 36
imagequeryOptions . . . 37
paintRules . . . 38
updateLayersControl . . . 38

Index **40**

`addCOG` *Add Cloud Optimised Geotiff (COG) to a leaflet map.*

Description

Add Cloud Optimised Geotiff (COG) to a leaflet map.

Usage

```
addCOG(
  map,
  url = NULL,
  group = NULL,
  layerId = NULL,
  resolution = 96,
  opacity = 0.8,
  options = leaflet::tileOptions(),
  colorOptions = NULL,
  pixelValuesToColorFn = NULL,
  autozoom = TRUE,
  rgb = FALSE,
  imagequery = TRUE,
  imagequeryOptions = NULL,
  ...
)
```

Arguments

map	the map to add the COG to.
url	url to the COG file to render.
group	he name of the group this raster image should belong to.
layerId	the layerId.
resolution	the target resolution for the simple nearest neighbor interpolation. Larger values will result in more detailed rendering, but may impact performance. Default is 96 (pixels).
opacity	opacity of the rendered layer.
options	options to be passed to the layer. See tileOptions for details.
colorOptions	list defining the palette, breaks and na.color to be used. Currently not used.
pixelValuesToColorFn	optional JS function to be passed to the browser. Can be used to fine tune and manipulate the color mapping. See examples & https://github.com/r-spatial/leafem/issues/25 for some examples. Currently not used.
autozoom	whether to automatically zoom to the full extent of the layer. Default is TRUE
rgb	logical, whether to render Geotiff as RGB. Currently not used.
imagequery	If TRUE a leaflet control with the hovered/clicked value will appear on the map.
imagequeryOptions	additional options for the control panel.
...	currently not used.

Details

This function will overlay Cloud Optimised Geotiff data from a remote url on a leaflet map. Like addGeotiff it uses the leaflet plugin 'georaster-layer-for-leaflet' to render the data. See addGeotiff for a bit more detail what that means.

Value

A leaflet map object.

Examples

```
if (interactive()) {
  library(leaflet)
  library(leafem)

  base_url = "https://sentinel-cogs.s3.us-west-2.amazonaws.com"
  image_url = "sentinel-s2-l2a-cogs/46/X/DG/2022/8/S2B_46XDG_20220829_0_L2A/L2A_PVI.tif"
  url = sprintf("%s/%s", base_url, image_url)

  leaflet() %>%
    addTiles() %>%
    leafem:::addCOG(
      url = url
      , group = "COG"
      , resolution = 512
      , autozoom = TRUE
    )
}
```

---

addCopyExtent *Copy current view extent to the clipboard*

Description

Add JavaScript functioality to enable copying of the current view bouding box to the clipboard. The `copy.btn` argument expects a valid keycode `event.code` such as "KeyE" (the default). Use https://www.toptal.com/developers/keycode to find the approprate codes for your keyboard.

Usage

```
addCopyExtent(map, event.code = "KeyE")
```

Arguments

`map`	a mapview or leaflet object.
`event.code`	the JavaScript event.code for ley strokes.

Examples

```
  library(leaflet)

  leaflet() %>%
  addProviderTiles("CartoDB.Positron") %>%
    addCopyExtent(event.code = "KeyE") %>%
```

```
  addMouseCoordinates()

# now click on the map (!) and zoom to anywhere in the map, then press 'e' on
# your keyboard. This will copy the current extent/bounding box as a JSON object
# to your clipboard which can then be parsed with:

# jsonlite::fromJSON(<Ctrl+v>)
```

---

addExtent *Add extent/bbox of spatial objects to a leaflet map*

Description

This function adds the bounding box of a spatial object to a leaflet or mapview map.

Usage

```
addExtent(map, data, ...)
```

Arguments

map	A leaflet or mapview map.
data	A sf object to be added to the map.
...	additional arguments passed on to addFeatures

Examples

```
library(leaflet)

# Usage in leaflet
leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addExtent(gadmCHE)

leaflet(gadmCHE) %>%
  addProviderTiles("OpenStreetMap") %>%
  addExtent()
```

addFeatures *Type agnositc version of* leaflet::add* *functions.*

Description

Add simple features geometries from sf

Usage

```
addFeatures(map, data, pane = "overlayPane", ...)
```

Arguments

map	A leaflet or mapview map.
data	A sf object to be added to the map.
pane	The name of the map pane for the features to be rendered in.
...	Further arguments passed to the respective leaflet::add* functions. See addCircleMarkers, addPolylines and addPolygons.

Value

A leaflet map object.

Examples

```
library(leaflet)

leaflet() %>% addProviderTiles("OpenStreetMap") %>% addCircleMarkers(data = breweries91)
leaflet() %>% addProviderTiles("OpenStreetMap") %>% addFeatures(data = breweries91)

leaflet() %>% addProviderTiles("OpenStreetMap") %>% addPolylines(data = atlStorms2005)
leaflet() %>% addProviderTiles("OpenStreetMap") %>% addFeatures(atlStorms2005)

leaflet() %>% addProviderTiles("OpenStreetMap") %>% addPolygons(data = gadmCHE)
leaflet() %>% addProviderTiles("OpenStreetMap") %>% addFeatures(gadmCHE)
```

addFgb *Add a flatgeobuf file to leaflet map*

Description

flatgeobuf is a performant binary geo-spatial file format suitable for serving large data. For more details see https://github.com/flatgeobuf/flatgeobuf and the respective documentation for the GDAL/OGR driver at https://gdal.org/en/latest/drivers/vector/flatgeobuf.html.

In contrast to classical ways of serving data from R onto a leaflet map, flatgeobuf can stream the data chunk by chunk so that rendering of the map is more or less instantaneous. The map is responsive while data is still loading so that popup queries, zooming and panning will work even though not all data has been rendered yet. This makes for a rather pleasant user experience as we don't have to wait for all data to be added to the map before interacting with it.

Usage

```
addFgb(
  map,
  file = NULL,
  url = NULL,
  layerId = NULL,
  group = NULL,
  popup = NULL,
  label = NULL,
  radius = 10,
  stroke = TRUE,
  color = "#03F",
  weight = 5,
  opacity = 0.5,
  fill = FALSE,
  fillColor = NULL,
  fillOpacity = 0.2,
  dashArray = NULL,
  options = NULL,
  className = NULL,
  scale = scaleOptions(),
  minZoom = NULL,
  maxZoom = 52,
  highlightOptions = NULL,
  labelOptions = NULL,
  ...
)
```

Arguments

map	a mapview or leaflet object.

file	file path to the .fgb file to be added to map. If set, url is ignored.
url	url of the data to be added to map. Only respected if file = NULL.
layerId	the layer id.
group	the group name for the file to be added to map.
popup	either a logical of whether to show the feature properties (fields) in popups or the name of the field to show in popups.
label	name of the field to be shown as a tooltip.
radius	the size of the circle markers.
stroke	whether to draw stroke along the path (e.g. the borders of polygons or circles).
color	stroke color.
weight	stroke width in pixels.
opacity	stroke opacity.
fill	whether to fill the path with fillColor. If fillColor is set, this will be set to TRUE, default is FALSE.
fillColor	fill color. If set, fill will be set to TRUE.
fillOpacity	fill opacity.
dashArray	a string that defines the stroke dash pattern.
options	a list of extra options for tile layers, popups, paths (circles, rectangles, polygons, ...), or other map elements.
className	optional class name for the popup (table). Can be used to define css for the popup.
scale	named list with instructions on how to scale radius, width, opacity, fillOpacity if those are to be mapped to an attribute column.
minZoom	minimum zoom level at which data should be rendered.
maxZoom	maximum zoom level at which data should be rendered.
highlightOptions	Options for highlighting the shape on mouse over.
labelOptions	A Vector of labelOptions to provide label options for each label. Default NULL
...	currently not used.

Details

Styling options in addFgb offer flexibility by allowing users to either specify styles directly as function arguments or define them as attributes in the data object:

- **Direct Styling:** You can pass style arguments (e.g., color, weight, opacity) directly to the function. These will apply uniformly to all features in the layer.
- **Attribute-based Styling:** Alternatively, you can include styling properties (e.g., color, fillColor, weight) as columns in your data object before writing it to an FGB file. Set the corresponding arguments in addFgb to NULL, and the function will use these attributes for styling during map rendering.

  For example:

```
  ## using custom `color`
  data$color <- colorNumeric(palette = "viridis", domain = data$var)(data$var)
  sf::st_write(obj = data, dsn = "myfile.fgb", driver = "FlatGeobuf")
  leafem::addFgb(file = "myfile.fgb", color = NULL)

  ## using custom `fillColor`
  data$fillColor <- colorNumeric(palette = "viridis", domain = data$var)(data$var)
  sf::st_write(obj = data, dsn = "myfile.fgb", driver = "FlatGeobuf")
  leafem::addFgb(file = "myfile.fgb", fill = TRUE, fillColor = NULL)
```

Examples

```
if (interactive()) {
  library(leaflet)
  library(leafem)

  # via URL
 url = "https://raw.githubusercontent.com/flatgeobuf/flatgeobuf/3.0.1/test/data/UScounties.fgb"

  leaflet() %>%
    addTiles() %>%
    leafem:::addFgb(
      url = url
      , group = "counties"
      , label = "NAME"
      , popup = TRUE
      , fill = TRUE
      , fillColor = "blue"
      , fillOpacity = 0.6
      , color = "black"
      , weight = 1
    ) %>%
      addLayersControl(overlayGroups = c("counties")) %>%
      addMouseCoordinates() %>%
      setView(lng = -105.644, lat = 51.618, zoom = 3)
}
```

---

addGeoRaster *Add stars/raster image to a leaflet map using optimised rendering.*

Description

Add stars/raster image to a leaflet map using optimised rendering.

Usage

```
addGeoRaster(
  map,
```

```
  x,
  group = NULL,
  layerId = NULL,
  resolution = 96,
  opacity = 0.8,
  options = leaflet::tileOptions(),
  colorOptions = NULL,
  project = TRUE,
  pixelValuesToColorFn = NULL,
  autozoom = TRUE,
  ...
)
```

Arguments

map	the map to add the raster data to.
x	the stars/raster object to be rendered.
group	the name of the group this raster image should belong to.
layerId	the layerId.
resolution	the target resolution for the simple nearest neighbor interpolation. Larger values will result in more detailed rendering, but may impact performance. Default is 96 (pixels).
opacity	opacity of the rendered layer.
options	options to be passed to the layer. See tileOptions for details.
colorOptions	list defining the palette, breaks and na.color to be used.
project	whether to project the RasterLayer to conform with leaflets expected crs. Defaults to TRUE and things are likely to go haywire if set to FALSE.
pixelValuesToColorFn	optional JS function to be passed to the browser. Can be used to fine tune and manipulate the color mapping. See https://github.com/r-spatial/leafem/issues/25 for some examples.
autozoom	whether to automatically zoom to the full extent of the layer. Default is TRUE
...	Further arguments passed to addGeotiff.

Details

This uses the leaflet plugin 'georaster-layer-for-leaflet' to render raster data. See https://github.com/GeoTIFF/georaster-layer-for-leaflet for details. The clue is that rendering uses simple nearest neighbor interpolation on-the-fly to ensure smooth rendering. This enables handling of larger rasters than with the standard addRasterImage.

Value

A leaflet map object.

Examples

```
if (interactive()) {
  library(leaflet)
  library(leafem)
  library(stars)

  tif = system.file("tif/L7_ETMs.tif", package = "stars")
  x1 = read_stars(tif)
  x1 = x1[, , , 3] # band 3

  leaflet() %>%
    addTiles() %>%
    leafem:::addGeoRaster(
      x1
      , opacity = 1
      , colorOptions = colorOptions(
        palette = grey.colors(256)
      )
    )
}
```

---

addGeotiff *Add a GeoTIFF file to a leaflet map using optimised rendering.*

Description

Add a GeoTIFF file to a leaflet map using optimised rendering.

Usage

```
addGeotiff(
  map,
  file = NULL,
  url = NULL,
  group = NULL,
  layerId = NULL,
  resolution = 96,
  bands = NULL,
  arith = NULL,
  project = TRUE,
  method = NULL,
  opacity = 0.8,
  options = leaflet::tileOptions(),
  colorOptions = NULL,
  rgb = FALSE,
  pixelValuesToColorFn = NULL,
  autozoom = TRUE,
```

```
  imagequery = TRUE,
  imagequeryOptions = NULL,
  ...
)
```

Arguments

map	the map to add the raster data to.
file	path to the GeoTIFF file to render.
url	url to the GeoTIFF file to render. Ignored if file is provided.
group	he name of the group this raster image should belong to.
layerId	the layerId.
resolution	the target resolution for the simple nearest neighbor interpolation. Larger values will result in more detailed rendering, but may impact performance. Default is 96 (pixels).
bands	which bands to use in case of multi-band Geotiff.
arith	an optional function to be applied to a multi-layer object. Will be computed on-the-fly in the browser.
project	if TRUE (default), automatically project x to the map projection expected by georaster-layer-for-leaflet (EPSG:4326); if FALSE, it's the caller's responsibility to ensure that file is already projected.
method	character defining the resampling method to be used when project is TRUE. See https://gdal.org/en/latest/programs/gdalwarp.html#cmdoption-gdalwarp-r for possible values.
opacity	opacity of the rendered layer.
options	options to be passed to the layer. See tileOptions for details.
colorOptions	list defining the palette, breaks and na.color to be used.
rgb	logical, whether to render Geotiff as RGB.
pixelValuesToColorFn	optional JS function to be passed to the browser. Can be used to fine tune and manipulate the color mapping. See examples & https://github.com/r-spatial/leafem/issues/25 for some examples.
autozoom	whether to automatically zoom to the full extent of the layer. Default is TRUE
imagequery	If TRUE a leaflet control with the hovered/clicked value will appear on the map.
imagequeryOptions	additional options for the control panel.
...	currently not used.

Details

This uses the leaflet plugin 'georaster-layer-for-leaflet' to render GeoTIFF data. See https://github.com/GeoTIFF/georaster-layer-for-leaflet for details. The GeoTIFF file is read directly in the browser using geotiffjs (https://geotiffjs.github.io/), so there's no need to read data into the current R session. GeoTIFF files can be read from the file system or via url. The clue is that rendering uses simple nearest neighbor interpolation on-the-fly to ensure smooth rendering. This enables handling of larger rasters than with the standard addRasterImage.

Value

A leaflet map object.

Examples

```
if (interactive()) {
  library(leaflet)
  library(leafem)
  library(stars)

  tif = system.file("tif/L7_ETMs.tif", package = "stars")
  x1 = read_stars(tif)
  x1 = x1[, , , 3] # band 3

  tmpfl = tempfile(fileext = ".tif")

  write_stars(st_warp(x1, crs = 4326), tmpfl)

  leaflet() %>%
    addTiles() %>%
    addGeotiff(
      file = tmpfl
      , opacity = 0.9
      , colorOptions = colorOptions(
        palette = hcl.colors(256, palette = "inferno")
        , na.color = "transparent"
      )
    )

}
```

---

addHomeButton *Add a home button / zoom-to-layer button to a map.*

Description

This function adds a button to the map that enables zooming to a provided extent / bbox.

Usage

```
addHomeButton(
  map,
  ext,
  group = "layer",
  position = "bottomright",
  add = TRUE,
  css = list(),
  hover_css = list()
```

```
)

removeHomeButton(map)
```

Arguments

map	a mapview or leaflet object.
ext	the extent / bbox to zoom to.
group	the name of the group/layer to be zoomed to (or any character string)
position	the position of the button (one of 'topleft', 'topright', 'bottomleft', 'bottomright'). Defaults to 'bottomright'.
add	logical. Whether to add the button to the map (mainly for internal use).
css, hover_css	list of valid CSS key-value pairs. See e.g. https://www.w3schools.com/cssref/index.php for possible values.

Functions

- `removeHomeButton()`: remove a homeButton from a map

Examples

```
library(leaflet)
library(raster)

## pass a group name only
m <- leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addCircleMarkers(data = breweries91, group = "breweries91") %>%
  addHomeButton(group = "breweries91")
m

## pass a raster extent - group can now be an arbitrary label
m <- leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addCircleMarkers(data = breweries91, group = "breweries91") %>%
  addHomeButton(ext = extent(breweries91), group = "Brew")
m

## remove the button
removeHomeButton(m)
```

`addImageQuery` *Add image query functionality to leaflet/mapview map.*

Description

Add image query functionality to leaflet/mapview map.

Usage

```
addImageQuery(
  map,
  x,
  band = 1,
  group = NULL,
  layerId = NULL,
  project = TRUE,
  type = c("mousemove", "click"),
  digits,
  position = "topright",
  prefix = "Layer",
  className = "",
  ...
)
```

Arguments

map	the map with the RasterLayer to be queried.
x	the RasterLayer that is to be queried.
band	for stars layers, the band number to be queried.
group	the group of the RasterLayer to be queried.
layerId	the layerId of the RasterLayer to be queried. Needs to be the same as supplied in addRasterImage or addStarsImage.
project	whether to project the RasterLayer to conform with leaflets expected crs. Defaults to TRUE and things are likely to go haywire if set to FALSE.
type	whether query should occur on 'mousemove' or 'click'. Defaults to 'mousemove'.
digits	the number of digits to be shown in the display field.
position	where to place the display field. Default is 'topright'.
prefix	a character string to be shown as prefix for the layerId.
className	a character string to append to the control legend.
...	currently not used.

Details

This function enables Raster*/stars objects added to leaflet/mapview maps to be queried. Standard query is on 'mousmove', but can be changed to 'click'. Note that for this to work, the layerId needs to be the same as the one that was set in addRasterImage or addStarsImage. Currently only works for numeric values (i.e. numeric/integer and factor values are supported).

Value

A leaflet map object.

Examples

```
if (interactive()) {
  if (requireNamespace("plainview")) {
    library(leaflet)
    library(plainview)

    leaflet() %>%
      addProviderTiles("OpenStreetMap") %>%
      addRasterImage(poppendorf[[1]], project = TRUE, group = "poppendorf",
                     layerId = "poppendorf") %>%
      addImageQuery(poppendorf[[1]], project = TRUE,
                    layerId = "poppendorf") %>%
      addLayersControl(overlayGroups = "poppendorf")
  }
}
```

---

addLocalFile *Add vector data to leaflet map directly from the file system*

Description

Add vector data to leaflet map directly from the file system

Usage

```
addLocalFile(
  map,
  file,
  layerId = NULL,
  group = NULL,
  popup = NULL,
  label = NULL,
  radius = 10,
  stroke = TRUE,
  color = "#03F",
  weight = 5,
  opacity = 0.5,
  fill = TRUE,
  fillColor = color,
  fillOpacity = 0.2,
  dashArray = NULL,
  options = NULL
)
```

Arguments

map	a mapview or leaflet object.
file	file path to the file to be added to map. NOTE: will be reprojected on-the-fly if not in "longlat".
layerId	the layer id.
group	the group name for the file to be added to map.
popup	either a logical of whether to show the feature properties (fields) in popups or the name of the field to show in popups.
label	name of the field to be shown as a tooltip.
radius	the size of the circle markers.
stroke	whether to draw stroke along the path (e.g. the borders of polygons or circles).
color	stroke color.
weight	stroke width in pixels.
opacity	stroke opacity.
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.
options	a list of extra options for tile layers, popups, paths (circles, rectangles, polygons, ...), or other map elements.

Examples

```
if (interactive()) {
  library(leafem)
  library(leaflet)
  library(sf)

  destfile = tempfile(fileext = ".gpkg")

  st_write(st_as_sf(gadmCHE), dsn = destfile)

  leaflet() %>%
    addTiles() %>%
    addLocalFile(destfile, popup = TRUE)
}
```

addLogo *add a local or remote image (png, jpg, gif, bmp, ...) to a leaflet map*

Description

This function adds an image to a map. Both local and remote (web) image sources are supported. Position on the map is completely controllable.

Usage

```
addLogo(
  map,
  img,
  alpha = 1,
  src = NULL,
  url = NULL,
  position = c("topleft", "topright", "bottomleft", "bottomright"),
  offset.x = 50,
  offset.y = 13,
  width = 60,
  height = 60,
  class = NULL,
  layerId = NULL
)

updateLogo(map, img, layerId)

removeLogo(map, layerId)

hideLogo(map, layerId)

showLogo(map, layerId)
```

Arguments

map	a mapview or leaflet object.
img	the image to be added to the map.
alpha	opacity of the added image.
src	DEPRECATED. The function now automatically determines if img is a local or remote image using file.exists(img).
url	an optional URL to be opened when clicking on the image (e.g. company's homepage).
position	one of "topleft", "topright", "bottomleft", "bottomright".
offset.x	the offset in x direction from the chosen position (in pixels).
offset.y	the offset in y direction from the chosen position (in pixels).

width	width of the rendered image in pixels.
height	height of the rendered image in pixels.
class	optional class
layerId	an id for the logo div.

Examples

```
library(leaflet)
## default position is topleft next to zoom control

img <- "https://www.r-project.org/logo/Rlogo.svg"
leaflet() %>% addTiles() %>% addLogo(img, url = "https://www.r-project.org/logo/")

## with local image
if (requireNamespace("png")) {
  library(png)

  img <- system.file("img", "Rlogo.png", package="png")
  leaflet() %>% addTiles() %>% addLogo(img, src = "local", alpha = 0.3)

  ## dancing banana gif :-)
  m <- leaflet() %>%
    addTiles() %>%
    addCircleMarkers(data = breweries91)

  addLogo(m, "https://jeroenooms.github.io/images/banana.gif",
          position = "bottomleft",
          offset.x = 5,
          offset.y = 40,
          width = 100,
          height = 100)
}
```

---

addMouseCoordinates *Add mouse coordinate information at top of map.*

Description

This function adds a box displaying the current cursor location (latitude, longitude and zoom level) at the top of a rendered mapview or leaflet map. In case of mapview, this is automatically added. NOTE: The information will only render once a mouse movement has happened on the map.

Usage

```
addMouseCoordinates(
  map,
```

```
  epsg = NULL,
  proj4string = NULL,
  native.crs = FALSE,
  css = list()
)

removeMouseCoordinates(map)

clip2sfc(x, clipboard = TRUE)
```

Arguments

map	a mapview or leaflet object.
epsg	the epsg string to be shown.
proj4string	the proj4string to be shown.
native.crs	logical. whether to use the native crs in the coordinates box.
css	list of valid CSS key-value pairs. See e.g. https://www.w3schools.com/cssref/index.php for possible values.
x	a charcter string with valid longitude and latitude values. Order matters! If missing and clipboard = TRUE (the default) contents will be read from the clipboard.
clipboard	whether to read contents from the clipboard. Default is TRUE.

Details

If style is set to "detailed", the following information will be displayed:

- x: x-position of the mouse cursor in projected coordinates
- y: y-position of the mouse cursor in projected coordinates
- epsg: the epsg code of the coordinate reference system of the map
- proj4: the proj4 definition of the coordinate reference system of the map
- lat: latitude position of the mouse cursor
- lon: longitude position of the mouse cursor
- zoom: the current zoom level

By default, only 'lat', 'lon' and 'zoom' are shown. To show the details about epsg, proj4 press and hold 'Ctrl' and move the mouse. 'Ctrl' + click will copy the current contents of the box/strip at the top of the map to the clipboard, though currently only copying of 'lon', 'lat' and 'zoom' are supported, not 'epsg' and 'proj4' as these do not change with pan and zoom.

Functions

- `removeMouseCoordinates()`: remove mouse coordinates information from a map
- `clip2sfc()`: convert mouse coordinates from clipboard to sfc

Examples

```
library(leaflet)

leaflet() %>%
  addProviderTiles("OpenStreetMap") # without mouse position info
m = leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addMouseCoordinates()

m

removeMouseCoordinates(m)

# adjust css
leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addMouseCoordinates(
    css = list(
     "font-size" = "30px",
     "text-align" = "center",
     "background-color" = "#ff000080",
     "color" = "rgb(255, 255, 255)"
   )
 )
```

`addPMPolygons` *Add vector tiles stored as PMTiles in an AWS S3 bucket to a leaflet map.*

Description

Add vector tiles stored as PMTiles in an AWS S3 bucket to a leaflet map.

Add point data stored as PMTiles

Add polylines stored as PMTiles

Usage

```
addPMPolygons(
  map,
  url,
  style,
  layerId = NULL,
  group = NULL,
  pane = "overlayPane",
  attribution = NULL
)
```

```
addPMPoints(
  map,
  url,
  style,
  layerId = NULL,
  group = NULL,
  pane = "overlayPane",
  attribution = NULL
)

addPMPolylines(
  map,
  url,
  style,
  layerId = NULL,
  group = NULL,
  pane = "overlayPane",
  attribution = NULL
)
```

Arguments

map	the map to add to.
url	the url to the tiles to be served.
style	styling for the layer. See paintRules for details.
layerId	the layer id.
group	group name.
pane	the map pane to which the layer should be added. See addMapPane for details.
attribution	optional attribution character string.

Details

These functions can be used to add cloud optimized vector tiles data in the `.pmtiles` format stored in an Amazon Web Services (AWS) S3 bucket to a leaflet map. For instructions on how to create these files, see https://github.com/protomaps/PMTiles.

NOTE: You may not see the tiles rendered in the RStudio viewer pane. Make sure to open the map in a browser.

Functions

- `addPMPoints()`: add points stored as PMTiles
- `addPMPolylines()`: add ploylines stored as PMTiles

Examples

```
## PMPolygons
library(leaflet)
```

```
library(leafem)

url_nzb = "https://vector-tiles-data.s3.eu-central-1.amazonaws.com/nz-building-outlines.pmtiles"

leaflet() %>%
  addTiles() %>%
  addPMPolygons(
    url = url_nzb
    , layerId = "nzbuildings"
    , group = "nzbuildings"
    , style = paintRules(
      layer = "nz-building-outlines"
      , fillColor = "pink"
      , stroke = "green"
    )
  ) %>%
  setView(173.50, -40.80, 6)

## PMPoints
library(leaflet)
library(leafem)

url_depoints = "https://vector-tiles-data.s3.eu-central-1.amazonaws.com/depoints.pmtiles"

leaflet() %>%
  addTiles() %>%
  addPMPoints(
    url = url_depoints
    , layerId = "depoints"
    , group = "depoints"
    , style = paintRules(
      layer = "depoints"
      , fillColor = "black"
      , stroke = "white"
      , radius = 4
    )
  ) %>%
  setView(10, 51, 6)

## PMPolylines
library(leaflet)
library(leafem)

url_rivers = "https://vector-tiles-data.s3.eu-central-1.amazonaws.com/rivers_africa.pmtiles"

## NOTE: these will only render until a zoom level of 7!!
leaflet() %>%
  addTiles() %>%
  addPMPolylines(
    url = url_rivers
    , layerId = "rivers"
    , group = "rivers"
    , style = paintRules(
```

```
    layer = "rivers_africa"
    , color = "blue"
  )
) %>%
setView(24, 2.5, 4)
```

---

addRasterRGB *Add an RGB image as a layer*

Description

Create a Red-Green-Blue image overlay from a `RasterStack` / `RasterBrick` or `stars` object based on three layers. Three layers (sometimes referred to as "bands" because they may represent different bandwidths in the electromagnetic spectrum) are combined such that they represent the red, green and blue channel. This function can be used to make 'true (or false) color images' from Landsat and other multi-band satellite images. Note, this text is plagiarized, i.e. copied from `plotRGB`. `addRasterRGB` and `addStarsRGB` are aliases.

Usage

```
addRasterRGB(
  map,
  x,
  r = 3,
  g = 2,
  b = 1,
  quantiles = c(0, 1),
  domain = NULL,
  na.color = "#BEBEBE80",
 method = c("auto", "bilinear", "near", "average", "mode", "cubic", "cubicspline",
    "lanczos", "sum", "min", "q1", "median", "q3", "max", "rms"),
  ...
)

addStarsRGB(
  map,
  x,
  r = 3,
  g = 2,
  b = 1,
  quantiles = c(0, 1),
  domain = NULL,
  na.color = "#BEBEBE80",
 method = c("auto", "bilinear", "near", "average", "mode", "cubic", "cubicspline",
    "lanczos", "sum", "min", "q1", "median", "q3", "max", "rms"),
  ...
)
```

Arguments

map	a map widget object created from 'leaflet()"
x	a RasterBrick, RasterStack or 'stars" raster object
r	integer. Index of the Red channel/band, between 1 and nlayers(x)
g	integer. Index of the Green channel/band, between 1 and nlayers(x)
b	integer. Index of the Blue channel/band, between 1 and nlayers(x)
quantiles	the upper and lower quantiles used for color stretching. If set to NULL, stretching is performed basing on domain argument.
domain	the upper and lower values used for color stretching. This is used only if quantiles is NULL. If both domain and quantiles are set to NULL, stretching is applied based on min-max values.
na.color	the color to be used for NA pixels
method	character. Method used for estimating the new cell values of a SpatRaster. One of: bilinear: bilinear interpolation (3x3 cell window). This is used by default if the first layer of x is not categorical average: This can be a good choice with continuous variables if the output cells overlap with multiple input cells. near: nearest neighbor. This is used by default if the first layer of x is categorical. This method is not a good choice for continuous values. mode: The modal value. This can be a good choice for categrical rasters, if the output cells overlap with multiple input cells. cubic: cubic interpolation (5x5 cell window). cubicspline: cubic B-spline interpolation. (5x5 cell window). lanczos: Lanczos windowed sinc resampling. (7x7 cell window). sum: the weighted sum of all non-NA contributing grid cells. min, q1, median, q3, max: the minimum, first quartile, median, third quartile, or maximum value. rms: the root-mean-square value of all non-NA contributing grid cells.
...	additional arguments passed on to addRasterImage

Details

Note, method auto, the default, will choose between near for factorial and bilinear for numeric data. All other methods need to be set manually.

Author(s)

Tim Appelhans, Luigi Ranghetti

Examples

```
require(raster)
require(stars)
require(plainview)
require(leaflet)

leaflet() %>%
  addTiles(group = "OpenStreetMap") %>%
  addRasterRGB(plainview::poppendorf, 4,3,2, group = "True colours") %>%
  addStarsRGB(st_as_stars(plainview::poppendorf), 5,4,3, group = "False colours") %>%
  addLayersControl(
    baseGroups = c("Satellite"),
    overlayGroups = c("True colours", "False colours"),
  )
```

---

addReactiveFeatures *Add a reactive layer to map.*

Description

This function adds a layer to a map that is dependent on another layer. The reactive layer will be shown/hidden when holding the Ctrl-button on your keyboard and performing the action defined by on. on can be either "click" (default) or "mouseover".

Note: srcLayer needs to be added to the map using addGeoJSON because we need to be able to link the two layers by a common attribute defined by argument by. Linking will be done via group name of srcLayer.

Usage

```
addReactiveFeatures(
  map,
  x,
  srcLayer,
  by,
  on,
  group,
  layerId = NULL,
  options = NULL,
  style = NULL,
  updateStyle = NULL,
  popup = NULL,
  ...
)
```

Arguments

map	a mapview or leaflet object.
x	the (sf) features to be added to the map.
srcLayer	the group name of the source layer that x should be bound to.
by	shared attribute between x and srcLayer by which the two layers should be bound together.
on	the action to invoke the action. Can be one of "click" (default) and "mouseover". The action will be triggered by holding Ctrl-key and performing on.
group	the group name for the object to be added to map.
layerId	the layerId.
options	options to be passed to the layer. See e.g. pathOptions for details.
style	named list of styling instructions for the geometries in x.
updateStyle	named list of how to update the styling of the srcLayer.
popup	a character vector of the HTML content for the popups of layer x. See addControl for details.
...	currently not used.

Examples

```
library(leaflet)
library(leafem)
library(sf)
library(geojsonsf)

# create some random data
che = st_as_sf(gadmCHE)
if (require(lwgeom)) {
  pts = st_as_sf(st_sample(che, 200))
  pts = st_join(pts, che[, "ID_1"])

  che = sf_geojson(che)

  leaflet() %>%
    addTiles() %>%
    addGeoJSON(che, group = "che") %>%
    addReactiveFeatures(
      pts
      , srcLayer = "che"
      , by = "ID_1"
      , on = "click"
      , group = "pts"
      , style = list(color = "black", fillOpacity = 0.3)
      , updateStyle = list(
        opacity = 0.3
        , fillOpacity = 0.3
        , color = "forestgreen"
        , fillColor = "forestgreen"
```

```
      )
    ) %>%
    addMouseCoordinates() %>%
    setView(lng = 8.31, lat = 46.75, zoom = 8)
}
```

---

addStarsImage *Add stars layer to a leaflet map*

Description

Add stars layer to a leaflet map

Usage

```
addStarsImage(
  map,
  x,
  band = 1,
  colors = "Spectral",
  opacity = 1,
  attribution = NULL,
  layerId = NULL,
  group = NULL,
  project = FALSE,
 method = c("auto", "bilinear", "near", "average", "mode", "cubic", "cubicspline",
    "lanczos", "sum", "min", "q1", "median", "q3", "max", "rms"),
  maxBytes = 4 * 1024 * 1024,
  options = gridOptions(),
  data = getMapData(map),
  ...
)
```

Arguments

map	a mapview or leaflet object.
x	a stars layer.
band	the band number to be plotted.
colors	the color palette (see colorNumeric) or function to use to color the raster values (hint: if providing a function, set na.color to "#00000000" to make NA areas transparent). The palette is ignored if x is a SpatRaster with a color table or if it has RGB channels.
opacity	the base opacity of the raster, expressed from 0 to 1
attribution	the HTML string to show as the attribution for this layer
layerId	the layer id

group	the name of the group this raster image should belong to (see the same parameter under addTiles)
project	if TRUE, automatically project x to the map projection expected by Leaflet (EPSG:3857); if FALSE, it's the caller's responsibility to ensure that x is already projected, and that extent(x) is expressed in WGS84 latitude/longitude coordinates
method	character. Method used for estimating the new cell values of a SpatRaster. One of: bilinear: bilinear interpolation (3x3 cell window). This is used by default if the first layer of x is not categorical average: This can be a good choice with continuous variables if the output cells overlap with multiple input cells. near: nearest neighbor. This is used by default if the first layer of x is categorical. This method is not a good choice for continuous values. mode: The modal value. This can be a good choice for categrical rasters, if the output cells overlap with multiple input cells. cubic: cubic interpolation (5x5 cell window). cubicspline: cubic B-spline interpolation. (5x5 cell window). lanczos: Lanczos windowed sinc resampling. (7x7 cell window). sum: the weighted sum of all non-NA contributing grid cells. min, q1, median, q3, max: the minimum, first quartile, median, third quartile, or maximum value. rms: the root-mean-square value of all non-NA contributing grid cells.
maxBytes	the maximum number of bytes to allow for the projected image (before base64 encoding); defaults to 4MB.
options	a list of additional options, intended to be provided by a call to gridOptions
data	the data object from which the argument values are derived; by default, it is the data object provided to leaflet() initially, but can be overridden.
...	currently not used.

Details

This is an adaption of addRasterImage. See that documentation for details.

Note, method auto, the default, will choose between near for factorial and bilinear for numeric data. All other methods need to be set manually.

Examples

```
library(stars)
library(leaflet)

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addStarsImage(x, project = TRUE)
```

addStaticLabels *Add static labels to* leaflet *or* mapview *objects*

Description

Being a wrapper around addLabelOnlyMarkers, this function provides a smart-and-easy solution to add custom text labels to an existing leaflet or mapview map object.

Usage

```
addStaticLabels(map, data, label, group = NULL, layerId = NULL, ...)
```

Arguments

map	A leaflet or mapview object.
data	A sf or Spatial* object used for label placement, defaults to the locations of the first dataset in 'map'.
label	The labels to be placed at the positions indicated by 'data' as character, or any vector that can be coerced to this type.
group	the group of the static labels layer.
layerId	the layerId of the static labels layer.
...	Additional arguments passed to labelOptions.

Value

A labelled **leaflet** map

Author(s)

Florian Detsch, Lorenzo Busetto

See Also

addLabelOnlyMarkers.

Examples

```
## Not run:
## leaflet label display options
library(leaflet)

lopt = labelOptions(noHide = TRUE,
                    direction = 'top',
                    textOnly = TRUE)

## Add labels on a Leaflet map
```

```
indata <- sf::st_read(system.file("shape/nc.shp", package="sf"))

leaflet(indata) %>%
  addProviderTiles("OpenStreetMap") %>%
  addFeatures(.) %>%
  addStaticLabels(., label = indata$NAME)

Modify styling -

leaflet(indata) %>%
  addProviderTiles("OpenStreetMap") %>%
  addFeatures(.) %>%
  addStaticLabels(., label = indata$NAME,
                    style = list("color" = "red", "font-weight" = "bold"))


## End(Not run)
```

`addTileFolder` *Add raster tiles from a local folder*

Description

Add tiled raster data pyramids from a local folder that was created with gdal2tiles.py (see https://gdal.org/en/latest/programs/gdal2tiles.html for details).

Usage

```
addTileFolder(
  map,
  folder,
  tms = TRUE,
  layerId = NULL,
  group = NULL,
  attribution = NULL,
  options = leaflet::tileOptions(),
  data = leaflet::getMapData(map)
)
```

Arguments

`map`	a mapview or leaflet object.
`folder`	the (top level) folder where the tiles (folders) reside.
`tms`	whether the tiles are served as TMS tiles.
`layerId`	the layer id.
`group`	the group name for the tile layer to be added to `map`.

attribution	the attribution text of the tile layer (HTML).
options	a list of extra options for tile layers. See tileOptions for details. When the tiles were created with gdal2tiles.py argument tms needs to be set to TRUE.
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.

---

colorOptions *Color options for addGeoRaster and addGeotiff*

Description

Color options for addGeoRaster and addGeotiff

Usage

```
colorOptions(
  palette = NULL,
  breaks = NULL,
  domain = NULL,
  na.color = "#bebebe22"
)
```

Arguments

palette	the color palette to use. Can be a set of colors or a color generating function such as the result of colorRampPalette.
breaks	the breaks at which color should change.
domain	the value domain (min/max) within which color mapping should occur.
na.color	color for NA values (will map to NaN in Javascript).

---

customizeLayersControl

Customize Layers Control of a Leaflet Map

Description

This function enables customization of an existing layers control in a leaflet map by adding custom views, home buttons, opacity controls, and legends. It enhances the functionality of a layers control created with leaflet or leaflet.extras. It also allows to customize the layersControl appearance via CSS.

Usage

```
customizeLayersControl(
  map,
  view_settings,
  home_btns = FALSE,
  home_btn_options = list(),
  setviewonselect = TRUE,
  opacityControl = list(),
  includelegends = TRUE,
  addCollapseButton = FALSE,
  layersControlCSS = list(),
  increaseOpacityOnHover = FALSE
)
```

Arguments

map	A leaflet or mapview object to which the extended layers control will be added.
view_settings	A list specifying the view settings for each layer. Each list element should contain either: • coords: A vector of length 2 (latitude, longitude) for setting the view, or length 4 (bounding box: lat1, lng1, lat2, lng2) for fitting the bounds. • zoom: The zoom level (used for setView). • fly (optional): A logical indicating whether to use flyTo or flyToBounds instead of setView or fitBounds. • options (optional): Additional options to pass to setView, fitBounds, or flyTo.
home_btns	Logical. If TRUE, adds a "home" button next to each layer name in the layer control. Clicking the home button zooms the map to the view specified for that layer in view_settings.
home_btn_options	A list of options to customize the home button appearance and behavior. Possible options include: • text: The text or emoji to display on the button (default is fontawesome::fa("home")). • cursor: CSS cursor style for the button (default is 'pointer'). • class: CSS class name for the button (default is 'leaflet-home-btn'). • styles: Semicolon separated CSS-string (default is 'float: inline-end;').
setviewonselect	Logical. If TRUE (default) sets the view when the layer is selected.
opacityControl	A list specifying the opacity control settings for each layer. Each list element should contain: • min: Minimum opacity value (default is 0). • max: Maximum opacity value (default is 1). • step: Step size for the opacity slider (default is 0.1). • default: Default opacity value (default is 1).

- width: Width of the opacity slider (default is '100%').
- class: CSS class name for the slider (default is 'leaflet-opacity-slider').

includelegends Logical. If TRUE (default), appends legends to the layer control. Legends are matched to layers by their group name. The legends need to be added with corresponding layer IDs.

addCollapseButton
Logical. If True a button will be added on top of the LayersControl which, when clicked, will expand/collapse the view. This is mainly relevant when the original Control was set to collapsed = FALSE.

layersControlCSS
a list of valid CSS key-value pairs to modify the appearance of the layersControl.

increaseOpacityOnHover
Logical. If TRUE the layersControl will be fully opaque when hovered. Mainly relevant if opacity was adjusted in layersControlCSS.

Details

This function generates JavaScript that listens for overlayadd or baselayerchange events and automatically sets the view or zoom level according to the specified view_settings. If home_btns is enabled, a home button is added next to each layer in the layer control. When clicked, it zooms the map to the predefined view of that layer. The opacity control slider allows users to adjust the opacity of layers. The legend will be appended to the corresponding layer control, matched by the layer's group name.

Value

A modified leaflet map object with extended layers control including view controls, home buttons, opacity controls, and legends.

Examples

```
library(sf)
library(leaflet)
library(leafem)

# Example data ##########
breweries91 <- st_as_sf(breweries91)
lines <- st_as_sf(atlStorms2005)
polys <- st_as_sf(gadmCHE)

# View settings ##########
view_settings <- list(
  "Base_tiles1" = list(
    coords = c(20, 50),
    zoom = 3
  ),
  "Base_tiles2" = list(
    coords = c(-110, 50),
    zoom = 5
  ),
```

```
  "breweries91" = list(
    coords = as.numeric(st_coordinates(st_centroid(st_union(breweries91)))),
    zoom = 8
  ),
  "atlStorms2005" = list(
    coords = as.numeric(st_bbox(lines)),
    options = list(padding = c(110, 110))
  ),
  "gadmCHE" = list(
    coords = as.numeric(st_bbox(polys)),
    options = list(padding = c(2, 2)),
    fly = TRUE
  )
)

# Opacity control settings ##########
opacityControl <- list(
  "breweries91" = list(
    min = 0,
    max = 1,
    step = 0.1,
    default = 1,
    width = '100%',
    class = 'opacity-slider'
  )
)

# Legends ##########
legends <- list(
  "breweries91" = "<div>Legend for breweries</div>"
)

leaflet() %>%
  ## Baselayer
  addTiles(group = "Base_tiles1") %>%
  addProviderTiles("CartoDB", group = "Base_tiles2") %>%

  ## Overlays
  addCircleMarkers(data = breweries91, group = "breweries91") %>%
  addPolylines(data = lines, group = "atlStorms2005") %>%
  addPolygons(data = polys, group = "gadmCHE") %>%

  ## LayersControl
  addLayersControl(
    baseGroups = c("Base_tiles1", "Base_tiles2"),
    overlayGroups = c("breweries91", "atlStorms2005", "gadmCHE"),
    options = layersControlOptions(collapsed = FALSE, autoZIndex = TRUE)
  ) %>%

  ## Customize Layers Control
  customizeLayersControl(
    view_settings = view_settings,
    home_btns = TRUE,
```

```
  home_btn_options = list(
    "Base_tiles1" = list(
      text = fontawesome::fa("home"),
      cursor = 'ns-resize',
      class = 'homebtn'
    ),
    "Base_tiles2" = list(
      text = fontawesome::fa("home"),
      cursor = 'pointer'
    ),
    "atlStorms2005" = list(
      text = fontawesome::fa("wind"),
      cursor = 'all-scroll'
    ),
    "breweries91" = list(
      text = fontawesome::fa("beer-mug-empty"),
      styles = 'background-color: lightgreen; float: inline-end'
    ),
    "gadmCHE" = list(
      text = fontawesome::fa("mountain"),
      styles = 'float: none;'
    )
  ),
  opacityControl = opacityControl,
  includelegends = TRUE,
  addCollapseButton = TRUE,
  layersControlCSS = list("opacity" = 0.6),
  increaseOpacityOnHover = TRUE
)
```

`garnishMap` *Garnish/decorate leaflet or mapview maps.*

Description

This function provides a versatile interface to add components to a leaflet or mapview map. It takes functions such as "addMouseCoordinates" or `addLayersControl` and their respective arguments and adds them to the map. Arguments must be named. Functions can be plain or character strings.

Usage

```
garnishMap(map, ...)
```

Arguments

`map`	a mapview or leaflet object.
`...`	functions and their arguments to add things to a map.

Examples

```
library(leaflet)

m <- leaflet() %>% addProviderTiles("OpenStreetMap")
garnishMap(m, addMouseCoordinates)

## add more than one with named argument
library(leaflet)

m1 <- garnishMap(m, addScaleBar, addMouseCoordinates,
                 position = "bottomleft")
m1
```

`imagequeryOptions` *Imagequery options for addGeoRaster, addGeotiff and addCOG*

Description

Imagequery options for addGeoRaster, addGeotiff and addCOG

Usage

```
imagequeryOptions(
  className = "info legend",
  position = c("topright", "topleft", "bottomleft", "bottomright"),
  type = c("mousemove", "click"),
  digits = NULL,
  prefix = "Layer",
  noData = "NoData Value"
)
```

Arguments

`className`	a character string to append to the control legend.
`position`	where to place the display field. Default is 'topright'.
`type`	whether query should occur on 'mousemove' or 'click'. Defaults to 'mousemove'.
`digits`	the number of digits to be shown in the display field.
`prefix`	a character string to be shown as prefix for the layerId.
`noData`	the text shown when the mouse is over a `NoData Value` as identified by GDAL. The default "NoData Value" will show whatever is defined by the Geotiff metadata.

paintRules *Styling options for PMTiles*

Description

Styling options for PMTiles

Usage

```
paintRules(
  layer,
  fillColor = "#0033ff66",
  color = "#0033ffcc",
  do_stroke = TRUE,
  width = 0.5,
  radius = 3,
  stroke = "#000000",
  opacity = 1,
  dash = NULL
)
```

Arguments

layer	the name of the layer in the PMTiles file to visualise.
fillColor	fill color for polygons
color	line color
do_stroke	logical, whether polygon borders should be drawn
width	line width
radius	point radius
stroke	color point border
opacity	point opacity
dash	either NULL (default) for a solid line or a numeric vector of length 2 denoting segment length and spce between segments (in pixels), e.g. c(5, 3)

updateLayersControl *Update the layer controls when adding layers to an existing map.*

Description

When adding additional base layers or overlay layers to an existing map, updateLayersControl will either update the existing layers control or add a new one if map has none.

Usage

```
updateLayersControl(
  map,
  addBaseGroups = character(0),
  addOverlayGroups = character(0),
  position = "topleft",
  ...
)
```

Arguments

map	A leaflet or mapview map.
addBaseGroups	group names of base layers to be added to layers control.
addOverlayGroups	group names of overlay layers to be added to layers control.
position	position of control: "topleft", "topright", "bottomleft", or "bottomright".
...	Further arguments passed to addLayersControl.

Value

A leaflet map object.

Examples

```
library(leaflet)

map = leaflet() %>%
        addProviderTiles("OpenStreetMap", group = "OSM") %>%
        addProviderTiles("CartoDB.DarkMatter", group = "dark") %>%
        addCircleMarkers(data = breweries91, group = "brew")

map # no layers control

map %>%
  updateLayersControl(addBaseGroups = c("OSM", "dark"),
                      addOverlayGroups = "brew")
```

Index

addCircleMarkers, *6*
addCOG, 2
addControl, *27*
addCopyExtent, 4
addExtent, 5
addFeatures, *5*, 6
addFgb, 7
addGeoJSON, *26*
addGeoRaster, 9
addGeotiff, *10*, 11
addHomeButton, 13
addImageQuery, 14
addLabelOnlyMarkers, *30*
addLayersControl, *36*, *39*
addLocalFile, 16
addLogo, 18
addMapPane, *22*
addMouseCoordinates, 19
addPMPoints *(*addPMPolygons*)*, 21
addPMPolygons, 21
addPMPolylines *(*addPMPolygons*)*, 21
addPolygons, *6*
addPolylines, *6*
addRasterImage, *10*, *12*, *15*, *25*, *29*
addRasterRGB, 24
addReactiveFeatures, 26
addStarsImage, *15*, 28
addStarsRGB *(*addRasterRGB*)*, 24
addStaticLabels, 30
addTileFolder, 31
addTiles, *29*

clip2sfc *(*addMouseCoordinates*)*, 19
colorNumeric, *28*
colorOptions, 32
colorRampPalette, *32*
customizeLayersControl, 32

garnishMap, 36
gridOptions, *29*

hideLogo *(*addLogo*)*, 18

imagequeryOptions, 37

labelOptions, *8*, *30*

paintRules, *22*, 38
pathOptions, *27*
plotRGB, *24*

removeHomeButton *(*addHomeButton*)*, 13
removeLogo *(*addLogo*)*, 18
removeMouseCoordinates
*(*addMouseCoordinates*)*, 19

sf, *6*
showLogo *(*addLogo*)*, 18

tileOptions, *3*, *10*, *12*, *32*

updateLayersControl, 38
updateLogo *(*addLogo*)*, 18