Package 'mapview'

April 28, 2018

Type Package

Title Interactive Viewing of Spatial Data in R

Version 2.4.0

Date 2018-04-28

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

Description Quickly and conveniently create interactive visualisations of spatial data with or without background maps. Attributes of displayed features are fully queryable via pop-up windows. Additional functionality includes methods to visualise true- and false-color raster images, bounding boxes, small multiples and 3D raster data cubes.

License GPL (>= 3) | file LICENSE

URL https://github.com/r-spatial/mapview

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

LazyData TRUE

Depends R (>= 2.10), methods

Imports leaflet, sp, raster, satellite, scales (>= 0.2.5), brew, htmlwidgets, htmltools, png, Rcpp (>= 0.11.3), lattice, gdalUtils, webshot, viridisLite, sf, base64enc, svglite, uuid

LinkingTo Rcpp

RoxygenNote 6.0.1

SystemRequirements GNU make

ByteCompile yes

Suggests knitr, rmarkdown, dplyr, testthat, covr, lwgeom

NeedsCompilation yes

Author Tim Appelhans [cre, aut],
Florian Detsch [aut],
Christoph Reudenbach [aut],
Stefan Woellauer [aut],
Spaska Forteva [ctb],
Thomas Nauss [ctb],
Edzer Pebesma [ctb],

Kenton Russell [ctb],
Michael Sumner [ctb],
Jochen Darley [ctb],
Pierre Roudier [ctb],
Environmental Informatics Marburg [ctb]

Repository CRAN

Date/Publication 2018-04-28 09:23:07 UTC

R topics documented:

`mapview-package` *Interactive viewing of spatial objects in R*

Description

Interactive viewing of spatial objects in R

Details

The package provides functionality to view spatial objects interactively. The intention is to provide interactivity for easy and quick visualization during spatial data analysis. It is not intended for fine-tuned presentation quality map production.

Author(s)

Tim Appelhans, Florian Detsch, Chris Reudenbach, Stephan Woellauer, Spaska Forteva, Thomas Nauss, Environmental Informatics Marburg

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

`+` *mapview + mapview adds data from the second map to the first*

Description

mapview + mapview adds data from the second map to the first

mapview + data adds spatial data (raster*, sf*, sp*) to a mapview map

[...]

Usage

```
## S4 method for signature 'mapview,mapview'
e1 + e2

## S4 method for signature 'mapview,ANY'
e1 + e2

## S4 method for signature 'mapview,character'
e1 + e2
```

Arguments

`e1`	a leaflet or mapview map to which e2 should be added.
`e2`	a (spatial) object to be added or a mapview object from which the objects should be added to e1.

Examples

```
m1 <- mapView(franconia, col.regions = "red")
m2 <- mapView(breweries)

### add two mapview objects
m1 + m2
'+'(m2, m1)

### add layers to a mapview object
m1 + breweries + poppendorf[[4]]
```

---

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 = breweries)
leaflet() %>% addProviderTiles("OpenStreetMap") %>% addFeatures(data = breweries)

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

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

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.

Use removeHomeButton to remove home button

Usage

```
addHomeButton(map, ext, layer.name = "layer", position = "bottomright",
  add = TRUE)

removeHomeButton(map)
```

Arguments

map	a mapview or leaflet object.
ext	the extent / bbox to zoom to.
layer.name	the name of the 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).

Functions

- removeHomeButton: remove a homeButton from a map

Examples

```
library(leaflet)
library(raster)

m <- leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addCircleMarkers(data = breweries) %>%
  addHomeButton(extent(breweries), "breweries")
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, group = NULL, layerId = NULL, project = TRUE,
  type = c("mousemove", "click"), digits, position = "topright",
  prefix = "Layer", ...)
```

Arguments

map	the map with the RasterLayer to be queried.
x	the RasterLayer that is 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 a supplied in addRasterImage.
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.
...	currently not used.

Details

This function enables Raster* 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. Currently only works for numeric values (i.e. numeric/integer and factor values are supported).

Value

A leaflet map object.

Examples

```
library(leaflet)
library(mapview)

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

---

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 = c("remote", "local"), url,
  position = c("topleft", "topright", "bottomleft", "bottomright"),
  offset.x = 50, offset.y = 13, width = 60, height = 60)
```

Arguments

map	a mapview or leaflet object.
img	the image to be added to the map.
alpha	opacity of the added image.
src	character specifying the source location ("local" for images from the disk, "remote" for web image sources).
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.

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
library(png)

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

## dancing banana gif :-)
m <- mapview(breweries91)

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

---

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

Description

map panes can be created by supplying a name and a zIndex to control layer ordering. We recommend a `zIndex` value between 400 (the default overlay pane) and 500 (the default shadow pane). You can then use this pane to render overlays (points, lines, polygons) by setting the `pane` argument in `leafletOptions`. This will give you control over the order of the layers, e.g. points always on top of polygons. If two layers are provided to the same pane, overlay will be determined by order of adding. See examples below. See http://www.leafletjs.com/reference-1.3.0.html#map-pane for details.

Usage

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

Arguments

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

Examples

```
library(leaflet)
library(mapview)

## points above polygons
leaflet() %>%
  addTiles() %>%
  addMapPane("polygons", zIndex = 410) %>%
  addMapPane("points", zIndex = 420) %>%
  addPolygons(data = franconia,
              group = "pol1",
              fillOpacity = 0.7,
              fillColor = "green",
              color = "black",
              options = leafletOptions(pane = "polygons")) %>%
  addPolygons(data = franconia,
              group = "pol2",
              color = "black",
              fillColor = "purple",
              fillOpacity = 0.7,
              options = leafletOptions(pane = "polygons")) %>%
  addCircleMarkers(data = breweries,
                   group = "pts",
                   color = "darkblue",
                   options = leafletOptions(pane = "points")) %>%
  addLayersControl(overlayGroups = c("pol1", "pol2", "pts"))
```

---

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, style = c("detailed", "basic"), epsg = NULL,
  proj4string = NULL, native.crs = FALSE)
```

Arguments

map	a mapview or leaflet object.
style	whether to show 'detailed' or 'basic' mouse position info. See Details for an explanation.

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.

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

If style is set to "basic", only 'lat', 'lon' and 'zoom' are shown.

Examples

```
library(leaflet)

leaflet() %>% addProviderTiles("OpenStreetMap") # without mouse position info
leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addMouseCoordinates(style = "basic") # with basic mouse position info
leaflet() %>%
  addProviderTiles("OpenStreetMap") %>%
  addMouseCoordinates() # with detailed mouse position info
```

---

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 addLabelOnlyMarkers.

Value

A labelled **mapview** object.

Author(s)

Florian Detsch

See Also

addLabelOnlyMarkers.

Examples

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

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

## point labels
m1 = mapview(breweries)
l1 = addStaticLabels(m1,
                     label = breweries$number.of.types,
                     labelOptions = lopt)
l1

## polygon centroid labels
m2 = mapview(franconia)
l2 = addStaticLabels(m2,
                     label = franconia$NAME_ASCI,
                     labelOptions = lopt)
l2

## custom labels
m3 = m2 + m1
l3 = addStaticLabels(m3,
```

```
                    data = franconia,
                    label = franconia$NAME_ASCI,
                    labelOptions = lopt)
l3

## End(Not run)
```

`breweries` *Selected breweries in Franconia*

Description

Selected breweries in Franconia

Format

`sf feature collection POINT`

Details

This dataset contains selected breweries in Franconia. It is partly a subset of a larger database that was compiled by students at the University of Marburg for a seminar called "The Geography of Beer: sustainability in the food industry" and partly consists of breweries downloaded from http://www.bierwandern.de/inhalt/brauereiliste.html with the kind permission of Rainer Kastl. Note that use of these data is restricted to non-commercial use and that they are explixitly excluded from the GPL lincense that mapview is licensed under.

`coords2JSON` *Convert a vector/matrix of coordinates to JSON format*

Description

Similar to `toJSON` from **jsonlite**, this function takes a set of coordinates as input and converts them to proper JSON format. Note that the function is powered by **Rcpp** which makes it a convenient alternative to existing methods when it comes to processing big datasets.

Usage

```
## S4 method for signature 'numeric'
coords2JSON(x)

## S4 method for signature 'character'
coords2JSON(x, xy = c(1, 2))

## S4 method for signature 'matrix'
coords2JSON(x, xy = c(1, 2))
```

Arguments

x	A 'numeric' vector with a single pair of coordinates or a matrix with multiple pairs of input coordinates, typically projected in EPSG:4326 (http://spatialreference.org/ref/epsg/wgs-84/).
xy	An 'integer' vector specifying the coordinate columns.

Value

A single 'character' object in JSON format.

Author(s)

Florian Detsch

Examples

```
crd <- matrix(ncol = 3, nrow = 12)

# x-coordinates
set.seed(10)
crd[, 1] <- rnorm(nrow(crd), 10, 3)

# y-coordinates
set.seed(10)
crd[, 2] <- rnorm(nrow(crd), 50, 3)

# additional data
crd[, 3] <- month.abb

# reformat a single pair of coordinates
coords2JSON(crd[1, ])

# reformat multiple pairs of coordinates at once
coords2JSON(crd)
```

---

coords2Lines *Convert points to SpatialLines**

Description

Create a SpatialLines* object from a Line object or set of point coordinates in one go, i.e. without being required to run through the single steps outlined in SpatialLines.

Usage

```
## S4 method for signature 'matrix'
coords2Lines(coords, ID, data, match.ID = TRUE, ...)

## S4 method for signature 'Line'
coords2Lines(coords, ID, data, match.ID = TRUE, ...)
```

Arguments

coords	Line object or 2-column numeric matrix with x and y coordinates.
ID	character, see Lines.
data	data.frame with data to add to the output SpatialLines* object (optional).
match.ID	logical, see SpatialLinesDataFrame.
...	Further arguments passed on to SpatialLines (i.e., proj4string).

Value

If data is missing, a SpatialLines object; else a SpatialLinesDataFrame object.

See Also

SpatialLines-class, SpatialLinesDataFrame.

Examples

```
library(sp)

coords1 <- cbind(c(2, 4, 4, 1, 2), c(2, 3, 5, 4, 2))
sln1 <- coords2Lines(coords1, ID = "A")

coords2 <- cbind(c(5, 4, 2, 5), c(2, 3, 2, 2))
sln2 <- coords2Lines(coords2, ID = "B")

mapview(sln1)

plot(sln1, col = "grey75")
plot(sln2, col = "grey25", add = TRUE)
```

---

coords2Polygons *Convert points to SpatialPolygons**

Description

Create a SpatialPolygons* object from a Polygon object or set of point coordinates in one go, i.e. without being required to run through the single steps outlined in SpatialPolygons.

Usage

```
## S4 method for signature 'matrix'
coords2Polygons(coords, hole = NA, ID, data,
  match.ID = TRUE, ...)

## S4 method for signature 'Polygon'
coords2Polygons(coords, ID, data, match.ID = TRUE, ...)
```

Arguments

coords	Polygon object or 2-column numeric matrix with x and y coordinates.
hole	logical, see Polygon.
ID	character, see Polygons.
data	data.frame with data to add to the output SpatialPolygons* object (optional).
match.ID	logical, see SpatialPolygonsDataFrame.
...	Further arguments passed on to SpatialPolygons (i.e., p0 and proj4string).

Value

If data is missing, a SpatialPolygons object; else a SpatialPolygonsDataFrame object.

See Also

SpatialPolygons-class, SpatialPolygonsDataFrame.

Examples

```
library(sp)

coords1 <- cbind(c(2, 4, 4, 1, 2), c(2, 3, 5, 4, 2))
spy1 <- coords2Polygons(coords1, ID = "A")

coords2 <- cbind(c(5, 4, 2, 5), c(2, 3, 2, 2))
spy2 <- coords2Polygons(coords2, ID = "B")

plot(spy1, col = "grey75")
plot(spy2, col = "grey25", add = TRUE)
```

---

cubeView *View a RasterStack or RasterBrick as 3-dimensional data cube.*

Description

Create a 3D data cube from a RasterStack or RasterBrick. The cube can be freely rotated so that Hovmoller views of x - z and y - z are possible.

Usage

```
cubeView(x, at, col.regions = mapviewGetOption("raster.palette"),
  na.color = mapviewGetOption("na.color"), legend = TRUE)

cubeview(...)
```

Arguments

x	a RasterStack or RasterBrick
at	the breakpoints used for the visualisation. See levelplot for details.
col.regions	color (palette).See levelplot for details.
na.color	color for missing values.
legend	logical. Whether to plot a legend.
...	currently not used.

Details

The visible layers are alterable by keys:
x-axis: LEFT / RIGHT arrow key
y-axis: DOWN / UP arrow key
z-axis: PAGE_DOWN / PAGE_UP key

Note: In RStudio cubeView may show a blank viewer window. In this case open the view in a web-browser (RStudio button at viewer: "show in new window").

Note: Because of key focus issues key-press-events are not always recognised within RStudio at Windows. In this case open the view in a web-browser (RStudio button at viewer: "show in new window").

Press and hold left mouse-button to rotate the cube. Press and hold right mouse-button to move the cube. Spin mouse-wheel or press and hold middle mouse-button and move mouse down/up to zoom the cube.

Functions

- cubeview: alias for ease of typing

Author(s)

Stephan Woellauer and Tim Appelhans

Examples

```
## Not run:
library(raster)

kili_data <- system.file("extdata", "kiliNDVI.tif", package = "mapview")
kiliNDVI <- stack(kili_data)
```

```
cubeView(kiliNDVI)

clr <- viridisLite::viridis
cubeView(kiliNDVI, at = seq(-0.15, 0.95, 0.1), col.regions = clr)

## End(Not run)
```

`cubeViewOutput` *Widget output function for use in Shiny*

Description

Widget output function for use in Shiny

Usage

```
cubeViewOutput(outputId, width = "100%", height = "400px")
```

Arguments

`outputId`	Output variable to read from
`width, height`	the width and height of the map (see `shinyWidgetOutput`)

`franconia` *Administrative district borders of Franconia*

Description

Administrative district borders of Franconia

Format

`sf feature collection MULTIPOLYGON`

Details

The NUTS_2013_01M_SH.zip archive was downloaded on 23/03/2017 from http://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units/nuts. https://gist.github.com/tim-salabim/2845fa90813fa25c18cf83f9b88cbde0

Source

http://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-unit
nuts

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, style = "basic")

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

m1 <- garnishMap(m, addMouseCoordinates, addHomeButton,
                 ext = extent(breweries))
m1

## even more flexible
m2 <- garnishMap(m1, addPolygons, data = franconia, popup = popupTable(franconia),
                 fillOpacity = 0.8, color = "black", fillColor = "#BEBEBE")
garnishMap(m2, addCircleMarkers, data = breweries)
```

knit_print.mapview *Print functions for mapview objects used in knitr*

Description

Print functions for mapview objects used in knitr

Usage

```
knit_print.mapview(x, ...)
```

Arguments

x	A mapview object
...	further arguments passed on to knit_print

---

latticeView *View two or more (possibly synchronised) mapview or leaflet maps*

Description

This function produces a lattice like view of two or more maps. It is possible to sync any combination of panels or all or none. For synchronising all panels it is best to use the provided convenience function sync.

Usage

```
latticeView(..., ncol = 2, sync = "none", sync.cursor = FALSE,
  no.initial.sync = TRUE)

latticeview(...)

sync(..., ncol = 2, sync = "all", sync.cursor = TRUE,
  no.initial.sync = TRUE)
```

Arguments

...	any number of mapview or leaflet objects or a list thereof
ncol	how many columns should be plotted
sync	whether to synchronise zoom and pan for certain elements. Possible values are "all" (default) to sync all maps, "none" to disable synchronisation or a list of panel numbers, e.g. `list(c(1, 3), c(2, 4))` will synchronise panels 1 & 3 and panels 2 & 4. Panels are drawn from top right to bottom left.
sync.cursor	whether to show cursor position in synced panels (default TRUE).
no.initial.sync	whether to sync the initial view (default TRUE).

Functions

- latticeview: alias for ease of typing
- sync: convenience function for syncing maps

Examples

```
## Not run:
library(sp)
library(raster)

data(meuse)
coordinates(meuse) <- ~x+y
proj4string(meuse) <- CRS("+init=epsg:28992")

## view different aspects of same data set
m1 <- mapview(meuse, zcol = "soil", burst = TRUE)
m2 <- mapview(meuse, zcol = "lead")
m3 <- mapview(meuse, zcol = "landuse", map.types = "Esri.WorldImagery")
m4 <- mapview(meuse, zcol = "dist.m")

latticeView(m1, m2, m3, m4) # 4 panels
sync(m1, m2, m3, m4) # 4 panels synchronised
latticeView(m1, m2) # 2 panels, split vertical
latticeView(m1, m2, ncol = 1) # 2 panels split horizontal
sync(m1, m2, ncol = 1) # same but synchronised
sync(m1, m2, m3, m4, sync = list(c(1, 2), c(3, 4))) # individual syncing
sync(m1, m2, m3, m4, sync = list(c(1, 2, 4)))

## view all layers of raster stack
map_list <- lapply(seq(nlayers(poppendorf)), function(i) {
  mapview(poppendorf[[i]], layer.name = names(poppendorf)[i])
})

latticeView(map_list, ncol = 5)

## view multiple data sets
m1 <- mapview(breweries, zcol = "founded")
m2 <- mapview(trails, zcol = "district", burst = TRUE)
m3 <- mapview(poppendorf[[5]], use.layer.names = TRUE)
m4 <- mapview(franconia, col.regions = "black")

latticeView(m1, m2, m3, m4) # not synced
sync(m1, m2, m3, m4) # synced
sync(m1, m2, m3, m4, no.initial.sync = FALSE) # all maps zoomed to m4 extent

## End(Not run)
```

`mapshot` *Save mapview or leaflet map as HTML and/or image*

Description

Save a mapview or leaflet map as `.html` index file or `.png`, `.pdf`, or `.jpeg` image.

Usage

```
mapshot(x, url = NULL, file = NULL, remove_url = TRUE,
  remove_controls = c("zoomControl", "layersControl", "homeButton",
  "scaleBar"), ...)
```

Arguments

x	mapview or leaflet object.
url	Output .html file. If not supplied and 'file' is specified, a temporary index file will be created.
file	Output .png, .pdf, or .jpeg file.
remove_url	logical. If TRUE (default), the .html file is removed once processing is completed. Only applies if 'url' is not specified.
remove_controls	character vector of control buttons to be removed from the map when saving to file. Any combination of "zoomControl", "layersControl", "homeButton", "scaleBar". If set to NULL nothing will be removed.
...	Further arguments passed on to webshot.

Details

mapshot can be used to save both leaflet and mapview maps as html or png files or both.

NOTE 1: In case you want to save larger maps produced with mapview (i.e. if you see the following warning: "the supplied feature layer has more points/vertices than the set threshold. using special rendering function, hence things may not behave as expected from a standard leaflet map") mapshot is likely to fail. Try setting `selfcontained = FALSE` to avoid errors and create a valid local html file.

NOTE 2: In case you want to save a map with popupGraphs or popupImages the respective graph/image files will be located one level above the specified target location. In case you want to move the html file, make sure to also move the respective *-graphs folder one level above.

See Also

webshot, saveWidget.

Examples

```
## Not run:
m <- mapview(breweries)

## create standalone .html
mapshot(m, url = paste0(getwd(), "/map.html"))

## create standalone .png; temporary .html is removed automatically unless
## 'remove_url = FALSE' is specified
mapshot(m, file = paste0(getwd(), "/map.png"))
mapshot(m, file = paste0(getwd(), "/map.png"),
        remove_controls = c("homeButton", "layersControl"))
```

```
## create .html and .png
mapshot(m, url = paste0(getwd(), "/map.html"),
        file = paste0(getwd(), "/map.png"))

## End(Not run)
```

---

mapView *View spatial objects interactively*

Description

this function produces an interactive view of the specified spatial object(s) on top of the specified base maps.

Usage

```
## S4 method for signature 'RasterLayer'
mapView(x, map = NULL,
  maxpixels = mapviewGetOption("mapview.maxpixels"),
  col.regions = mapviewGetOption("raster.palette")(256), at = NULL,
  na.color = mapviewGetOption("na.color"), use.layer.names = FALSE,
  values = NULL, map.types = mapviewGetOption("basemaps"),
  alpha.regions = 0.8, legend = mapviewGetOption("legend"),
  legend.opacity = 1, trim = TRUE, verbose = mapviewGetOption("verbose"),
  layer.name = NULL, homebutton = TRUE, native.crs = FALSE,
  method = c("bilinear", "ngb"), label = TRUE, query.type = c("mousemove",
  "click"), query.digits, query.position = "topright",
  query.prefix = "Layer", ...)

## S4 method for signature 'RasterStackBrick'
mapView(x, map = NULL,
  maxpixels = mapviewGetOption("mapview.maxpixels"),
  col.regions = mapviewGetOption("raster.palette")(256), at = NULL,
  na.color = mapviewGetOption("na.color"), use.layer.names = TRUE,
  values = NULL, map.types = mapviewGetOption("basemaps"),
  legend = mapviewGetOption("legend"), legend.opacity = 1, trim = TRUE,
  verbose = mapviewGetOption("verbose"), homebutton = TRUE,
  method = c("bilinear", "ngb"), label = TRUE, query.type = c("mousemove",
  "click"), query.digits, query.position = "topright",
  query.prefix = "Layer", ...)

## S4 method for signature 'Satellite'
mapView(x, map = NULL,
  maxpixels = mapviewGetOption("mapview.maxpixels"),
  col.regions = mapviewGetOption("raster.palette")(256), at = NULL,
```

```
  na.color = mapviewGetOption("na.color"), values = NULL,
  map.types = mapviewGetOption("basemaps"),
  legend = mapviewGetOption("legend"), legend.opacity = 1, trim = TRUE,
  verbose = mapviewGetOption("verbose"), homebutton = TRUE,
  method = c("bilinear", "ngb"), label = TRUE, ...)

## S4 method for signature 'sf'
mapView(x, map = NULL, pane = "auto", canvas = FALSE,
  zcol = NULL, burst = FALSE, color = mapviewGetOption("vector.palette"),
  col.regions = mapviewGetOption("vector.palette"), at = NULL,
  na.color = mapviewGetOption("na.color"), cex = 6, lwd = lineWidth(x),
  alpha = 0.9, alpha.regions = regionOpacity(x),
  na.alpha = regionOpacity(x), map.types = NULL,
  verbose = mapviewGetOption("verbose"), popup = popupTable(x),
  layer.name = NULL, label = makeLabels(x, zcol),
  legend = mapviewGetOption("legend"), legend.opacity = 1,
  homebutton = TRUE, native.crs = FALSE,
  highlight = mapviewHighlightOptions(x, alpha.regions, alpha, lwd),
  maxpoints = getMaxFeatures(x), ...)

## S4 method for signature 'sfc'
mapView(x, map = NULL, pane = "auto", canvas = FALSE,
  color = standardColor(x), col.regions = standardColRegions(x),
  at = NULL, na.color = mapviewGetOption("na.color"), cex = 6,
  lwd = lineWidth(x), alpha = 0.9, alpha.regions = regionOpacity(x),
  map.types = NULL, verbose = mapviewGetOption("verbose"), popup = NULL,
  layer.name = deparse(substitute(x, env = parent.frame(2))),
  label = makeLabels(x), legend = mapviewGetOption("legend"),
  legend.opacity = 1, homebutton = TRUE, native.crs = FALSE,
  highlight = mapviewHighlightOptions(x, alpha.regions, alpha, lwd),
  maxpoints = getMaxFeatures(x), ...)

## S4 method for signature 'numeric'
mapView(x, y, type = "p", grid = TRUE, label, ...)

## S4 method for signature 'data.frame'
mapView(x, xcol, ycol, grid = TRUE, aspect = 1,
  popup = popupTable(x), label, crs = NA, ...)

## S4 method for signature 'XY'
mapView(x, map = NULL, pane = "auto", canvas = FALSE,
  color = standardColor(x), col.regions = standardColRegions(x),
  at = NULL, na.color = mapviewGetOption("na.color"), cex = 6,
  lwd = lineWidth(x), alpha = 0.9, alpha.regions = regionOpacity(x),
  map.types = NULL, verbose = mapviewGetOption("verbose"), popup = NULL,
  layer.name = deparse(substitute(x, env = parent.frame(1))),
  label = makeLabels(x), legend = mapviewGetOption("legend"),
  legend.opacity = 1, homebutton = TRUE, native.crs = FALSE,
```

```
  highlight = mapviewHighlightOptions(x, alpha.regions, alpha, lwd),
  maxpoints = getMaxFeatures(x), ...)

## S4 method for signature 'XYZ'
mapView(x, layer.name = deparse(substitute(x, env =
  parent.frame(1))), ...)

## S4 method for signature 'XYM'
mapView(x, layer.name = deparse(substitute(x, env =
  parent.frame(1))), ...)

## S4 method for signature 'XYZM'
mapView(x, layer.name = deparse(substitute(x, env =
  parent.frame(1))), ...)

## S4 method for signature 'sfc_POINT'
mapView(x, ...)

## S4 method for signature 'sfc_MULTIPOINT'
mapView(x, ...)

## S4 method for signature 'sfc_LINESTRING'
mapView(x, ...)

## S4 method for signature 'sfc_MULTILINESTRING'
mapView(x, ...)

## S4 method for signature 'sfc_POLYGON'
mapView(x, ...)

## S4 method for signature 'sfc_MULTIPOLYGON'
mapView(x, ...)

## S4 method for signature 'sfc_GEOMETRY'
mapView(x, ...)

## S4 method for signature 'bbox'
mapView(x, layer.name = deparse(substitute(x, env =
  parent.frame(1))), alpha.regions = 0.2, ...)

## S4 method for signature 'missing'
mapView(map.types = mapviewGetOption("basemaps"), ...)

## S4 method for signature 'list'
mapView(x, map = NULL, zcol = NULL, burst = FALSE,
  color = mapviewGetOption("vector.palette"),
  col.regions = mapviewGetOption("vector.palette"), at = NULL,
  na.color = mapviewGetOption("na.color"), cex = 6, lwd = lapply(x,
```

```
  lineWidth), alpha = lapply(seq(x), function(i) 0.9),
  alpha.regions = lapply(seq(x), function(i) 0.6),
  map.types = mapviewGetOption("basemaps"),
  verbose = mapviewGetOption("verbose"), popup = lapply(seq(x), function(i)
  {     popupTable(x[[i]]) }), layer.name = deparse(substitute(x, env =
  parent.frame())), label = lapply(seq(x), function(i) {
  makeLabels(x[[i]], zcol = zcol[[i]]) }),
  legend = mapviewGetOption("legend"), legend.opacity = 1,
  homebutton = TRUE, native.crs = FALSE, maxpoints = NULL, ...)

## S4 method for signature 'ANY'
mapview(...)

## S4 method for signature 'SpatialPixelsDataFrame'
mapView(x, map = NULL, zcol = NULL,
  maxpixels = mapviewGetOption("mapview.maxpixels"),
  col.regions = mapviewGetOption("raster.palette")(256), at = NULL,
  na.color = mapviewGetOption("na.color"), use.layer.names = FALSE,
  values = NULL, map.types = mapviewGetOption("basemaps"),
  alpha.regions = 0.8, legend = mapviewGetOption("legend"),
  legend.opacity = 1, trim = TRUE, verbose = mapviewGetOption("verbose"),
  layer.name = NULL, homebutton = TRUE, native.crs = FALSE,
  method = c("bilinear", "ngb"), label = TRUE, query.type = c("mousemove",
  "click"), query.digits, query.position = "topright",
  query.prefix = "Layer", ...)

## S4 method for signature 'SpatialGridDataFrame'
mapView(x, map = NULL, zcol = NULL,
  maxpixels = mapviewGetOption("mapview.maxpixels"),
  col.regions = mapviewGetOption("raster.palette")(256), at = NULL,
  na.color = mapviewGetOption("na.color"), use.layer.names = FALSE,
  values = NULL, map.types = mapviewGetOption("basemaps"),
  alpha.regions = 0.8, legend = mapviewGetOption("legend"),
  legend.opacity = 1, trim = TRUE, verbose = mapviewGetOption("verbose"),
  layer.name = NULL, homebutton = TRUE, native.crs = FALSE,
  method = c("bilinear", "ngb"), label = TRUE, query.type = c("mousemove",
  "click"), query.digits, query.position = "topright",
  query.prefix = "Layer", ...)

## S4 method for signature 'SpatialPointsDataFrame'
mapView(x, zcol = NULL,
  layer.name = NULL, ...)

## S4 method for signature 'SpatialPoints'
mapView(x, zcol = NULL, layer.name = NULL, ...)

## S4 method for signature 'SpatialPolygonsDataFrame'
mapView(x, zcol = NULL,
```

```
  layer.name = NULL, ...)

## S4 method for signature 'SpatialPolygons'
mapView(x, zcol = NULL, layer.name = NULL, ...)

## S4 method for signature 'SpatialLinesDataFrame'
mapView(x, zcol = NULL, layer.name = NULL,
  ...)

## S4 method for signature 'SpatialLines'
mapView(x, zcol = NULL, layer.name = NULL, ...)
```

Arguments

x	a Raster* or Spatial* or Satellite or sf object or a list of any combination of those. Furthermore, this can also be a data.frame or a numeric vector. If missing, a blank map will be drawn.
map	an optional existing map to be updated/added to
maxpixels	integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), sampleRegular is used before plotting.
col.regions	color (palette) pixels. See levelplot for details.
at	the breakpoints used for the visualisation. See levelplot for details.
na.color	color for missing values
use.layer.names	should layer names of the Raster* object be used?
values	a vector of values for the visualisation of the layers. Per default these are calculated based on the supplied raster* object.
map.types	character spcifications for the base maps. see http://leaflet-extras.github.io/leaflet-providers/preview/ for available options.
alpha.regions	opacity of the fills of points, polygons or raster layer(s)
legend	should a legend be plotted
legend.opacity	opacity of the legend
trim	should the raster be trimmed in case there are NAs on the egdes
verbose	should some details be printed during the process
layer.name	the name of the layer to be shown on the map
homebutton	logical, whether to add a zoom-to-layer button to the map. Defaults to TRUE
native.crs	logical whether to reproject to web map coordinate reference system (web mercator - epsg:3857) or render using native CRS of the supplied data (can also be NA). Default is FALSE which will render in web mercator. If set to TRUE now background maps will be drawn (but rendering may be much quicker as no reprojecting is necessary). Currently only works for simple features.
method	for raster data only (raster/stars). Method used to compute values for the resampled layer that is passed on to leaflet. mapview does projection on-the-fly

	to ensure correct display and therefore needs to know how to do this projection. The default is 'bilinear' (bilinear interpolation), which is appropriate for continuous variables. The other option, 'ngb' (nearest neighbor), is useful for categorical variables. Ignored if the raster layer is of class `factor` in which case "ngb" is used.
`label`	For vector data (sf/sp) a character vector of labels to be shown on mouseover. See `addControl` for details. For raster data (Raster*/stars) a logical indicating whether to add image query.
`query.type`	for raster methods only. Whether to show raster value query on `'mousemove'` or `'click'`. Ignored if `label = FALSE`.
`query.digits`	for raster methods only. The amount of digits to be shown by raster value query. Ignored if `label = FALSE`.
`query.position`	for raster methods only. The position of the raster value query info box. See `position` argument of `addLegend` for possible values. Ignored if `label = FALSE`.
`query.prefix`	for raster methods only. a character string to be shown as prefix for the layerId. Ignored if `label = FALSE`.
`...`	additional arguments passed on to repective functions. See `addRasterImage`, `addCircles`, `addPolygons`, `addPolylines` for details
`pane`	name of the map pane in which to render features. See `addMapPane` for details. Currently only supported for vector layers. Ignored if `canvas = TRUE`. The default `"auto"` will create different panes for points, lines and polygons such that points overlay lines overlay polygons. Set to `NULL` to get default leaflet behaviour where allfeatures are rendered in the same pane and layer order is determined by automatically/sequencially.
`canvas`	whether to use canvas rendering rather than svg. May help performance with larger data. See `http://leafletjs.com/reference-1.3.0.html#canvas` for more information.
`zcol`	attribute name(s) or column number(s) in attribute table of the column(s) to be rendered. See also Details.
`burst`	whether to show all (TRUE) or only one (FALSE) layer(s). See also Details.
`color`	color (palette) for points/polygons/lines
`cex`	attribute name(s) or column number(s) in attribute table of the column(s) to be used for defining the size of circles
`lwd`	line width
`alpha`	opacity of lines
`na.alpha`	opacity of missing values
`popup`	a `list` of HTML strings with the popup contents, usually created from `popupTable`. See `addControl` for details.
`highlight`	either `FALSE`, `NULL` or a list of styling options for feature highlighting on mouse hover. See `highlightOptions` for details.
`maxpoints`	the maximum number of points making up the geometry. In case of lines and polygons this refers to the number of vertices. See Details for more information.
`y`	numeric vector.

type	whether to render the numeric vector x as a point "p" or line "l" plot.
grid	whether to plot a (scatter plot) xy-grid to aid interpretation of the visualisation. Only relevant for the data.frame method.
xcol	the column to be mapped to the x-axis. Only relevant for the data.frame method.
ycol	the column to be mapped to the y-axis. Only relevant for the data.frame method.
aspect	the ratio of x/y axis corrdinates to adjust the plotting space to fit the screen. Only relevant for the data.frame method.
crs	an optional crs specification for the provided data to enable rendering on a basemap. See argument description in st_sf for details.

Details

If zcol is not NULL but a length one character vector (referring to a column name of the attribute table) and burst is TRUE, one layer for each unique value of zcol will be drawn. The same will happen if burst is a length one character vector (again referring to a column of the attribute table).

NOTE: if XYZ or XYM or XYZM data from package sf is passed to mapview, domensions Z and M will be stripped to ensure smooth rendering even though the popup will potentially still say something like "POLYGON Z".

maxpoints is taken to determine when to switch rendering from svg to canvas overlay for perfomance. The threshold calculation is done as follows:
if the number of points (in case of point data) or vertices (in case of polygon or line data) > maxpoints then render using special render function. Within this render function we approximate the complexity of fetures by

```
maxFeatures <- maxfeatures / (npts(data) / length(data))
```

where npts determines the umber of points/vertices and length the number of features (points, lines or polygons). When the number of fetures in the current view window is larger than maxFeatures then features are rendered on the canvas, otherwise they are rendered as svg objects and fully queriable.

Methods (by class)

- RasterStackBrick: stack / brick
- Satellite: satellite
- sf: st_sf
- sfc: st_sfc
- numeric: numeric
- data.frame: data.frame
- XY: st_sfc
- XYZ: st_sfc
- XYM: st_sfc
- XYZM: st_sfc

- sfc_POINT: st_sfc
- sfc_MULTIPOINT: st_sfc
- sfc_LINESTRING: st_sfc
- sfc_MULTILINESTRING: st_sfc
- sfc_POLYGON: st_sfc
- sfc_MULTIPOLYGON: st_sfc
- sfc_GEOMETRY: st_sfc
- bbox: st_bbox
- missing: initiate a map without an object
- list: list
- ANY: alias for ease of typing
- SpatialPixelsDataFrame: SpatialPixelsDataFrame
- SpatialGridDataFrame: SpatialGridDataFrame
- SpatialPointsDataFrame: SpatialPointsDataFrame
- SpatialPoints: SpatialPoints
- SpatialPolygonsDataFrame: SpatialPolygonsDataFrame
- SpatialPolygons: SpatialPolygons
- SpatialLinesDataFrame: SpatialLinesDataFrame
- SpatialLines: SpatialLines

Author(s)

Tim Appelhans

Examples

```
## Not run:
mapview()

## simple features ====================================================
library(sf)

# sf
mapview(breweries)
mapview(franconia)

# sfc
mapview(st_geometry(breweries)) # no popup

# sfg / XY - taken from ?sf::st_point
outer = matrix(c(0,0,10,0,10,10,0,10,0,0),ncol=2, byrow=TRUE)
hole1 = matrix(c(1,1,1,2,2,2,1,1,1),ncol=2, byrow=TRUE)
hole2 = matrix(c(5,5,5,6,6,6,5,5,5),ncol=2, byrow=TRUE)
pts = list(outer, hole1, hole2)
(pl1 = st_polygon(pts))
```

```
mapview(pl1)

## raster ======================================================================
mapview(poppendorf[[5]])

## spatial objects =============================================================
mapview(leaflet::gadmCHE)
mapview(leaflet::atlStorms2005)


## styling options & legends ===================================================
mapview(franconia, color = "white", col.regions = "red")
mapview(franconia, color = "magenta", col.regions = "white")

mapview(breweries, zcol = "founded")
mapview(breweries, zcol = "founded", at = seq(1400, 2200, 200), legend = TRUE)
mapview(franconia, zcol = "district", legend = TRUE)

clrs <- sf.colors
mapview(franconia, zcol = "district", col.regions = clrs, legend = TRUE)

### multiple layers ===========================================================
mapview(franconia) + breweries
mapview(list(breweries, franconia))
mapview(franconia) + mapview(breweries) + trails

mapview(franconia, zcol = "district") + mapview(breweries, zcol = "village")
mapview(list(franconia, breweries),
        zcol = list("district", NULL),
        legend = list(TRUE, FALSE))


### burst =====================================================================
mapview(franconia, burst = TRUE)
mapview(franconia, burst = TRUE, hide = TRUE)
mapview(franconia, zcol = "district", burst = TRUE)


### ceci constitue la fin du pipe =============================================
library(dplyr)
library(sf)

franconia %>%
  sf::st_union() %>%
  mapview()

franconia %>%
  group_by(district) %>%
  summarize() %>%
  mapview(zcol = "district")

franconia %>%
  group_by(district) %>%
```

```
  summarize() %>%
  mutate(area = st_area(.) / 1e6) %>%
  mapview(zcol = "area")

franconia %>%
  mutate(area = sf::st_area(.)) %>%
  mapview(zcol = "area", legend = TRUE)

breweries %>%
  st_intersection(franconia) %>%
  mapview(zcol = "district")

franconia %>%
  mutate(count = lengths(st_contains(., breweries))) %>%
  mapview(zcol = "count")

franconia %>%
  mutate(count = lengths(st_contains(., breweries)),
         density = count / st_area(.)) %>%
  mapview(zcol = "density")

## End(Not run)
```

`mapview-class` *Class mapview*

Description

Class mapview

Slots

`object` the spatial object

`map` the leaflet map object

`mapviewColors` *mapview version of leaflet::color* functions*

Description

mapview version of leaflet::color* functions

Color palettes for mapview

Usage

```
mapviewColors(x, zcol = NULL, colors = mapviewGetOption("vector.palette"),
  at = NULL, na.color = mapviewGetOption("na.color"), ...)

mapviewPalette(name = "mapviewVectorColors")

mapViewPalette(name)
```

Arguments

x	Spatial* or Raster* object
zcol	the column to be colored
colors	color vector to be used for coloring the levels specified in at
at	numeric vector giving the breakpoints for the colors
na.color	the color for NA values.
...	additional arguments passed on to level.colors
name	Name of the color palette to be used. One of "mapviewVectorColors" (default), "mapviewRasterColors", "mapviewSpectralColors" or "mapviewTopoColors".

Author(s)

Tim Appelhans

See Also

level.colors

colorRampPalette

---

mapviewOptions *Global options for the mapview package*

Description

To permanently set any of these options, you can add them to <your R installation>/etc/Rprofile.site>. For example, to change the default number of pixels to be visualised for Raster* objects, add a line like this: options(mapviewMaxPixels = 700000) to that file.

query single mapviewOption parameters

Usage

```
mapviewOptions(platform, basemaps, raster.size, mapview.maxpixels,
  plainview.maxpixels, maxpoints, maxpolygons, maxlines, raster.palette,
  vector.palette, verbose, na.color, legend, legend.pos, layers.control.pos,
  default = FALSE, console = TRUE, leafletWidth, leafletHeight)

mapviewGetOption(param)
```

Arguments

platform	character. The platform to be used. Current options are "leaflet" and "quickmapr".
basemaps	character. The basemaps to be used for rendering data. See http://leaflet-extras.github.io/leaflet-providers/preview/ for possible values
raster.size	numeric. see the maxBytes argument in addRasterImage
mapview.maxpixels	numeric. The maximum amount of pixels allowed for Raster* objects to be rendered with mapview. Defaults to 500000. Set this higher if you have a potent machine or are patient enough to wait a little.
plainview.maxpixels	numeric. The maximum amount of pixels allowed for Raster* objects to be rendered with plainview. Defaults to 10000000. Set this higher if you have a potent machine or are patient enough to wait a little.
maxpoints	numeric. Maximum number of points allowed for leaflet overlay rendering. If this number is exceeded rendering will be done using special functionality which will provide much more speed and better handling. This means that standard functionality is reduced. For example adding layers via "+" is not possible anymore.
maxpolygons	numeric. Maximum number of polygons allowed for leaflet overlay rendering. If this number is exceeded rendering will be done using special functionality which will provide much more speed and better handling. This means that standard functionality is reduced. For example adding layers via "+" is not possible anymore.
maxlines	numeric. Maximum number of lines allowed for leaflet overlay rendering. If this number is exceeded rendering will be done using special functionality which will provide much more speed and better handling. This means that standard functionality is reduced. For example adding layers via "+" is not possible anymore.
raster.palette	a color palette function for raster visualisation. Should be a function that takes an integer as input and returns a vector of colors. See colorRampPalette for details.
vector.palette	a color palette function for vector visualisation. Should be a function that takes an integer as input and returns a vector of colors. See colorRampPalette for details.
verbose	logical. Many functions in mapview provide details about their behaviour. Set this to TRUE if you want to see these printed to the console.
na.color	character. The default color to be used for NA values.
legend	logical. Whether or not to show a legend for the layer(s).
legend.pos	Where should the legend be placed? One of "topleft", "topright", "bottomleft", "bottomright".
layers.control.pos	character. Where should the layer control be placed? One of "topleft", "topright", "bottomleft", "bottomright".
default	logical. If TRUE all options are set to their default values

console	logical. Should the options be printed to the console
leafletWidth, leafletHeight	height and width of the htmlwidget in px.
param	character. parameter to be queried.

Value

list of the current options (invisibly). If no arguments are provided the options are printed.

Functions

- mapviewGetOption: query single mapviewOption parameters

Author(s)

Tim Appelhans

See Also

rasterOptions, options

Examples

```
mapviewOptions()
mapviewOptions(na.color = "pink")
mapviewOptions()

mapviewGetOption("platform")

mapviewOptions(default = TRUE)
mapviewOptions()
```

---

mapviewOutput *Create a mapview UI element for use with shiny*

Description

Create a mapview UI element for use with shiny

Usage

```
mapviewOutput(outputId, width = "100%", height = 400)
```

Arguments

outputId	Output variable to read from
width, height	the width and height of the map (see shinyWidgetOutput)

npts *count the number of points/vertices/nodes of sf objects*

Description

count the number of points/vertices/nodes of sf objects

Usage

```
npts(x)
```

Arguments

x	an sf/sfc object

Examples

```
npts(franconia)
npts(sf::st_geometry(franconia[1, ])) # first polygon

npts(breweries) # is the same as
nrow(breweries)
```

plainView *View raster objects interactively without background map but in any CRS*

Description

this function produces an interactive view of the specified raster object(s) on a plain grey background but for any CRS.

Usage

```
## S4 method for signature 'RasterLayer'
plainView(x,
  maxpixels = mapviewGetOption("plainview.maxpixels"),
  col.regions = mapviewGetOption("raster.palette")(256), at,
  na.color = mapviewGetOption("na.color"), legend = TRUE,
  verbose = mapviewGetOption("verbose"), layer.name = deparse(substitute(x,
  env = parent.frame())), gdal = TRUE, ...)

## S4 method for signature 'RasterStackBrick'
plainView(x, r = 3, g = 2, b = 1,
  na.color = mapviewGetOption("na.color"),
```

```
  maxpixels = mapviewGetOption("plainview.maxpixels"),
  layer.name = deparse(substitute(x, env = parent.frame())), ...)

## S4 method for signature 'SpatialPixelsDataFrame'
plainView(x, zcol = 1, ...)

## S4 method for signature 'ANY'
plainview(...)
```

Arguments

x	a raster* object
maxpixels	integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), sampleRegular is used before plotting.
col.regions	color (palette).See levelplot for details.
at	the breakpoints used for the visualisation. See levelplot for details.
na.color	color for missing values.
legend	logical, whether to draw a legend for the raster layer.
verbose	should some details be printed during the process
layer.name	the name of the layer to be shown on the map
gdal	logical. If TRUE (default) gdalUtils::gdal_translate is used to create the png file for display when possible. See details for further information.
...	additional arguments passed on to repective functions. See addRasterImage, addCircles, addPolygons, addPolylines for details
r	integer. Index of the Red channel, between 1 and nlayers(x)
g	integer. Index of the Green channel, between 1 and nlayers(x)
b	integer. Index of the Blue channel, between 1 and nlayers(x)
zcol	attribute name or column number in attribute table of the column to be rendered

Details

If the raster object is not in memory (i.e. if raster::inMemory is FLASE) and argument gdal is set to TRUE (default) gdalUtils::gdal_translate is used to translate the rsater object to a png file to be rendered in the viewer/browser. This is fast for large rasters. In this case, argument maxpixels is not used, instead the image is rendered in original resolution. However, this means that RasterLayers will be shown in greyscale. If you want to set a color palette manually, use gdal = FALSE and (optionally provide) col.regions.

For plainView there are a few keyboard shortcuts defined:

- plus/minus - zoom in/out
- space - toggle antialiasing
- esc - zoom to layer extent
- enter - set zoom to 1
- ctrl - increase panning speed by 10

Methods (by class)

- RasterStackBrick: stack / brick
- SpatialPixelsDataFrame: SpatialPixelsDataFrame
- ANY: alias for ease of typing

Author(s)

Stephan Woellauer

Tim Appelhans

Examples

```
### raster data ###
library(sp)
library(raster)

data(meuse.grid)
coordinates(meuse.grid) = ~x+y
proj4string(meuse.grid) <- CRS("+init=epsg:28992")
gridded(meuse.grid) = TRUE
meuse_rst <- stack(meuse.grid)

# SpatialPixelsDataFrame
plainView(meuse.grid, zcol = "dist")


# raster layer
m1 <- plainView(poppendorf[[5]])
m1

# raster stack - true color
plainview(poppendorf, 4, 3, 2)
```

---

poppendorf *Landsat 8 detail of Franconian Switzerland centered on Poppendorf*

Description

Landsat 8 detail of Franconian Switzerland centered on Poppendorf

Format

"RasterBrick-class" with 5 bands (bands 1 to 5).

Details

Use of this data requires your agreement to the USGS regulations on using Landsat data.

Source

https://earthexplorer.usgs.gov

---

popupTable *Create HTML strings for popups*

Description

Create HTML strings for popup tables used as input for mapview or leaflet. This optionally allows the user to include only a subset of feature attributes.

Create HTML strings for popup images used as input for mapview or leaflet.

Create HTML strings for popup graphs used as input for mapview or leaflet.

Usage

```
popupTable(x, zcol, row.numbers = TRUE)

popupImage(img, src = c("local", "remote"), ...)

popupGraph(graphs, type = c("png", "svg", "html"), width = 300,
  height = 300, ...)
```

Arguments

x	A Spatial* object.
zcol	numeric or character vector indicating the columns included in the output popup table. If missing, all columns are displayed.
row.numbers	logical whether to include row numbers in the popup table.
img	A character vector of file path(s) or web-URL(s) to any sort of image file(s).
src	Whether the source is "local" (i.e. valid file path(s)) or "remote" (i.e. valid URL(s)).
...	further arguments passed on to underlying methods such as height and width.
graphs	A list of figures associated with x.
type	Output filetype, one of "png" (default), "svg" or "html".
width	popup width in pixels.
height	popup height in pixels.

Details

Type svg uses native svg encoding via readLines. height and width are set via ... and passed on to svg
Type png embeds via "<img src = ...". height and width are set via ... and passed on to png
Type html embeds via "<iframe src = ...". height and width are set directly in pixels.

Value

A `list` of HTML strings required to create feature popup tables.

A `list` of HTML strings required to create popup graphs.

A `list` of HTML strings required to create popup graphs.

Examples

```
library(leaflet)

## include columns 1 and 2 only
mapview(franconia, popup = popupTable(franconia, zcol = 1:2))
mapview(breweries, zcol = "founded", legend = TRUE,
        popup = popupTable(breweries, zcol = c("founded", "village")))
leaflet() %>% addCircleMarkers(data = breweries)
leaflet() %>% addCircleMarkers(data = breweries,
                               popup = popupTable(breweries))

## Not run:
## remote images -----
### one image
library(sf)

pnt = st_as_sf(data.frame(x = 174.764474, y = -36.877245),
                coords = c("x", "y"),
                crs = 4326)

img = "http://bit.ly/1TVwRiR"

mapview(pnt, popup = popupImage(img, src = "remote"))

### multiple file (types)
library(sp)
images = c(img,
          "https://upload.wikimedia.org/wikipedia/commons/9/91/Octicons-mark-github.svg",
            "https://www.r-project.org/logo/Rlogo.png",
            "https://upload.wikimedia.org/wikipedia/commons/d/d6/MeanMonthlyP.gif")

pt4 = data.frame(x = jitter(rep(174.764474, 4), factor = 0.01),
                 y = jitter(rep(-36.877245, 4), factor = 0.01))
coordinates(pt4) = ~ x + y
proj4string(pt4) = "+init=epsg:4326"

mapview(pt4, popup = popupImage(images)) # NOTE the gif animation

## local images -----
pnt = st_as_sf(data.frame(x = 174.764474, y = -36.877245),
                coords = c("x", "y"), crs = 4326)
img = system.file("img","Rlogo.png",package="png")
mapview(pnt, popup = popupImage(img))

## End(Not run)
```

```
## Not run:
### example: svg -----

library(sp)

data(meuse)
coordinates(meuse) = ~ x + y
proj4string(meuse) = CRS("+init=epsg:28992")

## create plots with points colored according to feature id
library(lattice)
p = xyplot(copper ~ cadmium, data = meuse@data, col = "grey")
p = mget(rep("p", length(meuse)))

clr = rep("grey", length(meuse))
p = lapply(1:length(p), function(i) {
  clr[i] = "red"
  update(p[[i]], col = clr)
})

mapview(meuse, popup = popupGraph(p, type = "svg"))

### example: png -----
pt = data.frame(x = 174.764474, y = -36.877245)

coordinates(pt) = ~ x + y
proj4string(pt) = "+init=epsg:4326"

p2 = levelplot(t(volcano), col.regions = terrain.colors(100))

mapview(pt, popup = popupGraph(p2, width = 300, height = 400))

### example: html -----
mapview(breweries[1, ], map.types = "Esri.WorldImagery",
        popup = popupGraph(mapview(breweries[1, ])@map,
                           type = "html",
                           width = 500,
                           height = 400))

## End(Not run)
```

---

print,mapview-method *Method for printing mapview objects*

Description

Method for printing mapview objects

Usage

```
## S4 method for signature 'mapview'
print(x)
```

Arguments

x	a mapview object

---

renderCubeView *Widget render function for use in Shiny*

Description

Widget render function for use in Shiny

Usage

```
renderCubeView(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr	An expression that generates an HTML widget
env	The environment in which to evaluate expr
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable

---

renderMapview *Render a mapview widget in shiny*

Description

Render a mapview widget in shiny

Usage

```
renderMapview(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr	An expression that generates an HTML widget
env	The environment in which to evaluate expr
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable

show,mapview-method *Method for printing mapview objects (show)*

Description

Method for printing mapview objects (show)

Usage

```
## S4 method for signature 'mapview'
show(object)
```

Arguments

object	a mapview object

slideView *slideView*

Description

Two images are overlaid and a slider is provided to interactively compare the two images in a before-after like fashion. img1 and img2 can either be two RasterLayers, two RasterBricks/Stacks or two character strings. In the latter case it is assumed that these point to .png images on the disk.

NOTE: In case you want to include multiple slideviews in one page in a Rmd or flexdashboard we highly recommend using package widgetframe. Also, make sure to use different image names and/or labels for each of the RasterLayers/Bricks/Stacks. Otherwise things will likely not work properly.

This is a modified implementation of http://bl.ocks.org/rfriberg/8327361

Usage

```
## S4 method for signature 'RasterStackBrick,RasterStackBrick'
slideView(img1, img2,
  label1 = deparse(substitute(img1, env = parent.frame())),
  label2 = deparse(substitute(img2, env = parent.frame())), r = 3, g = 2,
  b = 1, na.color = mapviewGetOption("na.color"),
  maxpixels = mapviewGetOption("plainview.maxpixels"), ...)

## S4 method for signature 'RasterLayer,RasterLayer'
slideView(img1, img2,
  label1 = deparse(substitute(img1, env = parent.frame())),
  label2 = deparse(substitute(img2, env = parent.frame())), legend = TRUE,
  col.regions = mapviewGetOption("raster.palette")(256),
  na.color = mapviewGetOption("na.color"),
```

```
  maxpixels = mapviewGetOption("plainview.maxpixels"))

## S4 method for signature 'RasterStackBrick,RasterLayer'
slideView(img1, img2,
  label1 = deparse(substitute(img1, env = parent.frame())),
  label2 = deparse(substitute(img2, env = parent.frame())), legend = TRUE,
  r = 3, g = 2, b = 1,
  col.regions = mapviewGetOption("raster.palette")(256),
  na.color = mapviewGetOption("na.color"),
  maxpixels = mapviewGetOption("plainview.maxpixels"), ...)

## S4 method for signature 'RasterLayer,RasterStackBrick'
slideView(img1, img2,
  label1 = deparse(substitute(img1, env = parent.frame())),
  label2 = deparse(substitute(img2, env = parent.frame())), legend = TRUE,
  r = 3, g = 2, b = 1,
  col.regions = mapviewGetOption("raster.palette")(256),
  na.color = mapviewGetOption("na.color"),
  maxpixels = mapviewGetOption("plainview.maxpixels"), ...)

## S4 method for signature 'character,character'
slideView(img1, img2,
  label1 = deparse(substitute(img1, env = parent.frame())),
  label2 = deparse(substitute(img2, env = parent.frame())))

## S4 method for signature 'ANY'
slideview(...)
```

Arguments

img1	a RasterStack/Brick, RasterLayer or path to a .png file
img2	a RasterStack/Brick, RasterLayer or path to a .png file
label1	slider label for img1 (defaults to object name)
label2	slider label for img2 (defaults to object name)
r	integer. Index of the Red channel, between 1 and nlayers(x)
g	integer. Index of the Green channel, between 1 and nlayers(x)
b	integer. Index of the Blue channel, between 1 and nlayers(x)
na.color	the color to be used for NA pixels
maxpixels	integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), sampleRegular is used before plotting.
...	additional arguments passed on to repective functions.
legend	whether to plot legends for the two images (ignored for RatserStacks/*Bricks).
col.regions	color (palette).See levelplot for details.
color	the color palette to be used for visualising RasterLayers

Details

Compare two images trough interactive swiping overlay

For slideView there are a few keyboard shortcuts defined:

- space - toggle antialiasing
- esc - zoom to layer extent
- enter - set zoom to 1
- ctrl - increase panning speed by 10

Methods (by class)

- `img1 = RasterLayer,img2 = RasterLayer`: for RasterLayers
- `img1 = RasterStackBrick,img2 = RasterLayer`: for RasterStackBrick, RasterLayer
- `img1 = RasterLayer,img2 = RasterStackBrick`: for RasterLayer, RasterStackBrick
- `img1 = character,img2 = character`: for png files
- `ANY`: alias for ease of typing

Author(s)

Tim Appelhans

Stephan Woellauer

Examples

```
### raster data ###
library(sp)
library(raster)

data(poppendorf)

stck1 <- subset(poppendorf, c(3, 4, 5))
stck2 <- subset(poppendorf, c(2, 3, 4))
slideView(stck1, stck2)

## Not run:
### example taken from
### http://www.news.com.au/technology/environment/nasa-images-reveal-
### aral-sea-is-shrinking-before-our-eyes/story-e6frflp0-1227074133835

library(jpeg)
library(raster)

web_img2000 <- "http://cdn.newsapi.com.au/image/v1/68565a36c0fccb1bc43c09d96e8fb029"

jpg2000 <- readJPEG(readBin(web_img2000, "raw", 1e6))

# Convert imagedata to raster
rst_blue2000 <- raster(jpg2000[, , 1])
```

```
rst_green2000 <- raster(jpg2000[, , 2])
rst_red2000 <- raster(jpg2000[, , 3])

img2000 <- brick(rst_red2000, rst_green2000, rst_blue2000)

web_img2013 <- "http://cdn.newsapi.com.au/image/v1/5707499d769db4b8ec76e8df61933f2a"

jpg2013 <- readJPEG(readBin(web_img2013, "raw", 1e6))

# Convert imagedata to raster
rst_blue2013 <- raster(jpg2013[, , 1])
rst_green2013 <- raster(jpg2013[, , 2])
rst_red2013 <- raster(jpg2013[, , 3])

img2013 <- brick(rst_red2013, rst_green2013, rst_blue2013)

slideView(img2000, img2013, label1 = "before", label2 = "after")

## End(Not run)
```

`trails` *Selected hiking trails in Franconia*

Description

Selected hiking trails in Franconia

Format

`sf feature collection MULTILINESTRING`

Details

These hiking trails were downloaded on 06/04/2017 from https://geoportal.bayern.de/bayernatlas
These data are published by the owner under Creative Commons Namensnennung 3.0 Deutschland,
see https://creativecommons.org/licenses/by/3.0/de/ for details.

Source

Datenquelle: Bayerische Vermessungsverwaltung - www.geodaten.bayern.de http://www.ldbv.bayern.de/produkte/weitere/opendata.html

viewExtent *View extent/bbox of spatial objects interactively*

Description

This function produces an interactive view of the extent/bbox of the supplied spatial object

Usage

```
viewExtent(x, map = NULL, popup = NULL, layer.name = NULL,
  alpha.regions = 0.2, label = NULL, ...)

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

Arguments

x	either a Raster*, sf* or Spatial* object
map	a leaflet map the extent should be added to. If NULL standard background layers are created.
popup	a list of HTML strings with the popup contents, usually created from popupTable. See addControl for details.
layer.name	the name of the layer to be shown on the map.
alpha.regions	opacity of the fills or the raster layer(s).
label	a character vector of labels to be shown on mouseover. See addControl for details.
...	additional arguments passed on to addRectangles
data	either a Raster*, sf* or Spatial* object

Functions

- addExtent: add extent/bbox of spatial/sf objects to a leaflet map

Author(s)

Tim Appelhans

Examples

```
library(leaflet)

viewExtent(poppendorf)
viewExtent(breweries)
viewExtent(franconia) + breweries
viewExtent(trails) + trails + breweries
leaflet() %>% addProviderTiles("OpenStreetMap") %>% addExtent(breweries)
```

viewRGB *Red-Green-Blue map view of a multi-layered Raster object*

Description

Make a Red-Green-Blue plot based on three layers (in a RasterBrick or RasterStack). 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 plagirized, i.e. copied from plotRGB.

Usage

```
## S4 method for signature 'RasterStackBrick'
viewRGB(x, r = 3, g = 2, b = 1,
  quantiles = c(0.02, 0.98), map = NULL,
  maxpixels = mapviewGetOption("mapview.maxpixels"),
  map.types = mapviewGetOption("basemaps"),
  na.color = mapviewGetOption("na.color"),
  layer.name = deparse(substitute(x, env = parent.frame())),
  method = c("bilinear", "ngb"), ...)
```

Arguments

x	a RasterBrick or RasterStack
r	integer. Index of the Red channel, between 1 and nlayers(x)
g	integer. Index of the Green channel, between 1 and nlayers(x)
b	integer. Index of the Blue channel, between 1 and nlayers(x)
quantiles	the upper and lower quantiles used for color stretching. If set to NULL, no stretching is applied.
map	the map to which the layer should be added
maxpixels	integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), sampleRegular is used before plotting.
map.types	character spcifications for the base maps. see http://leaflet-extras.github.io/leaflet-providers/preview/ for available options.
na.color	the color to be used for NA pixels
layer.name	the name of the layer to be shown on the map
method	Method used to compute values for the resampled layer that is passed on to leaflet. mapview does projection on-the-fly to ensure correct display and therefore needs to know how to do this projection. The default is 'bilinear' (bilinear interpolation), which is appropriate for continuous variables. The other option, 'ngb' (nearest neighbor), is useful for categorical variables.
...	additional arguments passed on to mapView

Author(s)

Tim Appelhans

Examples

```
library(raster)

viewRGB(poppendorf, 4, 3, 2) # true-color
viewRGB(poppendorf, 5, 4, 3) # false-color
```

Index