Package: map (via r-universe)

March 12, 2025

Type Package

Title Defines a meta class of geographical objects, the 'map' class, and associated tools

Description The 'map' class is a collection of geospatial objects (e.g., from the 'sp', 'raster', 'sf', and 'terra' packages), with a number of metadata additions to enable powerful methods, e.g., for leaflet, reproducible GIS etc.

URL https://map.predictiveecology.org, https://github.com/PredictiveEcology/map

Date 2025-02-11

Version 1.1.0.9003

Depends R (>= 4.2)

Imports backports, data.table (>= 1.10.4), digest, methods, parallelly, pemisc (>= 0.0.4.9008), quickPlot, raster (>= 2.8.4), reproducible (>= 2.0.8.9001), sf, stats, terra, tiler (>= 0.3.0), utils, withr

Suggests covr, sp, SpaDES.tools (>= 2.0.4.9002), testthat (>= 3.0.0), usethis

Remotes ropensci/tiler, PredictiveEcology/pemisc@development, PredictiveEcology/quickPlot@development, PredictiveEcology/reproducible@development, cran/rgdal, cran/rgeos, PredictiveEcology/SpaDES.tools@development

Encoding UTF-8

Language en-CA

License GPL-3

BugReports https://github.com/PredictiveEcology/map/issues

ByteCompile yes

Collate 'GIS.R' 'buildMetadata.R' 'makeTiles.R' 'map-Defunct.R' 'map-analysis.R' 'map-class.R' 'map-package.R' 'map.R' 'rbindlistAG.R' 'zzz.R'

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev make default-jdk libpng-dev libssl-dev libproj-dev python3 libsqlite3-dev libudunits2-dev

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

RemoteUrl https://github.com/PredictiveEcology/map

RemoteRef development

RemoteSha a4d099df5b20325075495e57e4adf44fea31ac9e

Contents

.rasterToMemory .rasterToMemory

Description

.rasterToMemory

Usage

```
.rasterToMemory(x, ...)

## S3 method for class 'Raster'
.rasterToMemory(x, ...)

## S3 method for class 'SpatRaster'
.rasterToMemory(x, ...)
```

Arguments

x	A Raster or SpatRaster object
...	Additional arguments passed to raster read function

addColumnNameForLabels *Add* shinyLabel *column (attribute) to spatial vectors*

Description

Add shinyLabel column (attribute) to spatial vectors

Usage

```
addColumnNameForLabels(x, columnNameForLabels, ...)

## Default S3 method:
addColumnNameForLabels(x, columnNameForLabels, ...)

## S3 method for class 'list'
addColumnNameForLabels(x, columnNameForLabels, ...)

## S3 method for class 'sf'
addColumnNameForLabels(x, columnNameForLabels, ...)

## S3 method for class 'SpatialPolygonsDataFrame'
addColumnNameForLabels(x, columnNameForLabels, ...)
```

```
## S3 method for class 'SpatVector'
addColumnNameForLabels(x, columnNameForLabels, ...)
```

Arguments

x	a spatial vector object (sf, SpatialPolygons, SpatVector)
columnNameForLabels	character or integer identifying an existing column (attribute) to use as the shinyLabel.
...	Additional arguments passed to other methods (not used)

Value

a modified object with the same class as x

---

areaAndPolyValue *Determine the area of each zone in a raster*

Description

Polygonize a raster and calculate the area of each polygon.

Usage

```
areaAndPolyValue(ras, ...)

## S3 method for class 'Raster'
areaAndPolyValue(ras, ...)

## S3 method for class 'SpatRaster'
areaAndPolyValue(ras, ...)
```

Arguments

ras	A Raster or SpatRaster object
...	Additional arguments (not used)

Value

list containing: sizeInHa, the area; and polyID, the polygon ID.

buildMetadata *Build* map *object metadata table*

Description

Build map object metadata table

Usage

```
buildMetadata(
  metadata,
  isStudyArea,
  isRasterToMatch,
  layerName,
  obj,
  columnNameForLabels,
  objHash,
  leaflet,
  envir,
  ...
)
```

Arguments

metadata	metadata data.table.
isStudyArea	Logical indicating whether obj a study area.
isRasterToMatch	Logical indicating whether obj is a rasterToMatch layer. If TRUE, then this layer can be accessed by rasterToMatch(map).
layerName	Character specifying a label for this layer.
obj	The geospatial object being added to the map object.
columnNameForLabels	Character specifying the column name to use for labels.
objHash	Character specifying the digest hash of the object.
leaflet	Logical or Character vector of path(s) to write tiles. If TRUE or a character vector, then this layer will be added to a leaflet map. For RasterLayer or SpatRaster object, this will make tiles. If path is not specified, it will be the current path. The tile base file path will be paste0(layerName, "_", rndstr(1, 6)).
envir	TODO: description needed
...	Additional arguments.

crs,map-method *Extract the CRS of a* map

Description

Extract the CRS of a map

Usage

```
## S4 method for signature 'map'
crs(x, ...)
```

Arguments

x	Raster* or Spatial object
...	additional arguments. None implemented

See Also

Other mapMethods: mapRm(), rasterToMatch,map-method, rasters(), studyArea(), studyAreaName()

fasterize2 *Rasterize following crop and reproject*

Description

A simple wrapper around terra::rasterize().

Usage

```
fasterize2(emptyRaster, polygonToFasterize, field)
```

Arguments

emptyRaster	An empty RasterLayer orSpatRaster to use as a template.
polygonToFasterize	an sf or SpatVector object, which will be cropped first if extent(emptyRaster) < extent(polygonToFasterize).
field	character or numeric. If field is a character, it should a variable name in x. If field is numeric it typically is a single number or a vector of length nrow(x). The values are recycled to nrow(x)

Value

an object of the same class as emptyRaster

Examples

```
## using sf + raster
f1 <- system.file("external/lux.shp", package = "raster")
v1 <- sf::st_read(f1)
r1 <- raster::raster(v1, ncols = 75, nrows = 100)
raster::crs(r1) <- "epsg:4326"
z1 <- fasterize2(r1, v1, "NAME_2")

## using terra
f2 <- system.file("ex/lux.shp", package = "terra")
v2 <- terra::vect(f2)
r2 <- terra::rast(v2, ncols = 75, nrows = 100)
z2 <- fasterize2(r2, v2, "NAME_2")

terra::compareGeom(terra::rast(z1), z2)
```

---

gdal_polygonizeR *Polygonize a raster*

Description

Polygonize a raster

Usage

```
gdal_polygonizeR(
  x,
  outshape = NULL,
  gdalformat = "ESRI Shapefile",
  pypath = NULL,
  readpoly = TRUE,
  quiet = TRUE
)
```

Arguments

x	a RasterLayer, SpatRaster, or character giving the filepath to a raster.
outshape	character giving the filepath for the output shapefile.
gdalformat	GDAL driver to use. See terra::gdal(drivers=TRUE).
pypath, quiet	deprecated. maintained for backwards compatibility only (not used).
readpoly	logical indicating whether the polygons object should be returned (this was previously using gdal_polygonize on disk and required reading the file)

Value

if readpoly = TRUE (default), a SpatVector object; otherwise, NULL.

leafletTiles *Extract leaflet tile paths from a* map *obj*

Description

Extract leaflet tile paths from a map obj

Usage

```
leafletTiles(map)
```

Arguments

map	A map class obj

Value

A vector of paths indicating the relative paths. Any layers that don't have leaflet tiles will return NA.

makeTiles *Make tiles (pyramids) using* gdal2tiles

Description

Make tiles (pyramids) using gdal2tiles

Usage

```
makeTiles(tilePath, obj, overwrite = FALSE, ...)
```

Arguments

tilePath	A director to write tiles
obj	A raster object with or without file-backing
overwrite	Logical. If FALSE, and the director exists, then it will not overwrite any files.
...	Arguments passed to reproducible::projectInputs() (e.g., useGDAL).

map-class *The* map *class*

Description

Contains a common system for organizing geospatial vector and raster data, principally for use with **leaflet** and **shiny**.

Slots

`metadata` `data.table` with columns describing metadata of objects in the `map`.

`.xData` Named environment of geospatial data objects (e.g., `sf`, `Raster*`, `Spatial*`). Each entry may also be simply an environment, which indicates where to find the object, i.e., via `get(layerName, envir = environment)`.

`CRS` The common CRS of all layers.

`paths` A named list of file paths. The default is a list of length 2 specifying `dataPath` and `tilePath`.

`analyses` A `data.table` or `data.frame` of the types of analyses to perform.

`analysesData` A `data.table` or `data.frame` of the results of the analyses.

mapAdd *Append a spatial object to map*

Description

If `isStudyArea = TRUE`, then several things will be triggered:

1. This layer will be added to metadata with `studyArea` set to `max(studyArea(map)) + 1`.
2. update CRS slot to be the CRS of the study area.

Usage

```
mapAdd(obj, map, layerName, overwrite = getOption("map.overwrite", FALSE), ...)

## Default S3 method:
mapAdd(
  obj = NULL,
  map = new("map"),
  layerName = NULL,
  overwrite = getOption("map.overwrite"),
  columnNameForLabels = 1,
  leaflet = FALSE,
  isStudyArea = FALSE,
  isRasterToMatch = FALSE,
```

```
  envir = NULL,
  useCache = getOption("reproducible.useCache", TRUE),
  useParallel = getOption("map.useParallel", FALSE),
  ...
)
```

Arguments

obj	Optional spatial object, currently RasterLayer, SpatialPolygons.
map	Optional map object. If not provided, then one will be created. If provided, then the present obj or options passed to reproducible::prepInputs() (e.g., url), will be appended to this map.
layerName	Required. A label for this map layer. This can be the same as the object name.
overwrite	Logical. If TRUE and this layerName exists in the map, then it will replace the existing object. Default is getOption("map.overwrite").
...	Additional arguments passed to: reproducible::postProcess(), reproducible::projectInputs(), reproducible::fixErrors(), and reproducible::prepInputs().
columnNameForLabels	A character string indicating which column to use for labels. This is currently only used if the object is a sp::SpatialPolygonsDataFrame.
leaflet	Logical or Character vector of path(s) to write tiles. If TRUE or a character vector, then this layer will be added to a leaflet map. For RasterLayer object, this will trigger a call to gdal2tiles, making tiles. If path is not specified, it will be the current path. The tile base file path will be paste0(layerName, "_", rndstr(1, 6)).
isStudyArea	Logical. If TRUE, this will be assigned the label "StudyArea", and will be passed into reproducible::prepInputs() for any future layers added.
isRasterToMatch	Logical indicating whether the object to be added should be considered a rasterToMatch.
envir	An optional environment. If supplied, then the obj will not be placed "into" the maps slot, rather the environment label will be placed into the maps slot.
useCache	Logical. If TRUE, internal calls to reproducible::Cache() will be used.
useParallel	Logical. If TRUE, then if there is more than one calculation to do at any stage, it will create and use a parallel cluster via makeOptimalCluster(). If running analyses in parallel, it may be useful to pass a function (via .clInit) to be run on each of the nodes immediately upon cluster creation (e.g., to set options).

Examples

```
withr::local_tempdir("example_mapAdd_") |>
  withr::local_dir()

StudyArea <- list(cbind(
  x = c(-122.98, -116.1, -99.2, -106, -122.98),
  y = c(59.9, 65.73, 63.58, 54.79, 59.9)
)) |>
```

```
  sf::st_polygon() |>
  sf::st_sfc() |>
  sf::st_sf(geometry = _, ID = 1L, shinyLabel = "zone2", crs = "epsg:4326")

ml <- mapAdd(StudyArea, isStudyArea = TRUE, layerName = "Small Study Area",
             poly = TRUE, analysisGroup2 = "Small Study Area")

if (require("SpaDES.tools", quietly = TRUE)) {
  withr::local_options(list(
    map.tilePath = withr::local_tempdir("tiles_"),
    map.useParallel = FALSE
  ))
  smallStudyArea <- SpaDES.tools::randomPolygon(studyArea(ml), 1e5)
  smallStudyArea$ID <- 1L
  smallStudyArea$shinyLabel <- "zone2"

  ml <- mapAdd(smallStudyArea, ml, isStudyArea = TRUE, filename2 = NULL,
               analysisGroup2 = "Smaller Study Area",
               poly = TRUE,
               layerName = "Smaller Study Area") # adds a second studyArea within 1st

  rasTemplate <- terra::rast(terra::ext(studyArea(ml)), resolution = 0.001)
  tsf <- SpaDES.tools::randomPolygons(rasTemplate, numTypes = 8)*30
  vtm <- SpaDES.tools::randomPolygons(tsf, numTypes = 4)
  levels(vtm) <- data.frame(
    ID = sort(unique(vtm[])),
    Factor = c("black spruce", "white spruce", "aspen", "fir")
  )

  ml <- mapAdd(tsf, ml, layerName = "tsf1",
               filename2 = "tsf1.tif", # to postProcess
               # to map object
               tsf = "tsf1.tif", # to column in map@metadata
               analysisGroup1 = "tsf1_vtm1",  # this is the label for analysisGroup1
              leaflet = TRUE, # to column in map@metadata; used for visualizing in leaflet
               overwrite = TRUE)
  ml <- mapAdd(vtm, ml, filename2 = "vtm1.grd",
               layerName = "vtm1",
               vtm = "vtm1.grd",
               analysisGroup1 = "tsf1_vtm1", leaflet = TRUE, overwrite = TRUE)

 ## these map analyses are in `LandWebUtils` package, which is reverse dependency of this one
  ageClasses <- c("Young", "Immature", "Mature", "Old")
  ageClassCutOffs <- c(0, 40, 80, 120)

 ## add an analysis -- this will trigger analyses because there are already objects in the map
  ##    This will trigger 2 analyses:
  ##    LeadingVegTypeByAgeClass on each raster x polygon combo (only 1 currently)
  ##    so there is 1 raster group, 2 polygon groups, 1 analyses - Total 2, 2 run now
  # ml <- mapAddAnalysis(ml, functionName = "LeadingVegTypeByAgeClass",
  #                      ageClasses = ageClasses, ageClassCutOffs = ageClassCutOffs)

 ## add an analysis -- this will trigger analyses because there are already objects in the map
```

```
##    This will trigger 2 more analyses:
##    largePatches on each raster x polygon combo (only 1 currently)
##   so there is 1 raster group, 2 polygon groups, 2 analyses - Total 4, only 2 run now
# ml <- mapAddAnalysis(ml, functionName = "LargePatches", ageClasses = ageClasses,
#                      id = "1", labelColumn = "shinyLabel",
#                      ageClassCutOffs = ageClassCutOffs)

## Add a second polygon, trigger
# smallStudyArea2 <- randomPolygon(studyArea(ml), 1e5)
# smallStudyArea2$ID <- 1L
# smallStudyArea2$shinyLabel <- "zone2"

## add a new layer -- this will trigger analyses because there are already analyese in the map
##   This will trigger 2 more analyses ... largePatches on each *new* raster x polygon combo
##   (now there are 2) -- so there is 1 raster group, 3 polygon groups, 2 analyses - Total 6
# ml <- mapAdd(smallStudyArea2, ml, isStudyArea = FALSE, filename2 = NULL, overwrite = TRUE,
#              analysisGroup2 = "Smaller Study Area 2",
#              poly = TRUE,
#             layerName = "Smaller Study Area 2") # adds a second studyArea within 1st

## Add a *different* second polygon, via overwrite. This should trigger new analyses.
# smallStudyArea2 <- randomPolygon(studyArea(ml), 1e5)
# smallStudyArea2$ID <- 1
# smallStudyArea2$shinyLabel = "zone1"

## add a new layer -- this will trigger analyses because there are already analyses in the map
##   This will trigger 2 more analyses ... largePatches on each *new* raster x polygon combo
##   (now there are 2) -- so there is 1 raster group, 3 polygon groups, 2 analyses - Total 6
# ml <- mapAdd(smallStudyArea2, ml, isStudyArea = FALSE, filename2 = NULL, overwrite = TRUE,
#              analysisGroup2 = "Smaller Study Area 2",
#              poly = TRUE,
#             layerName = "Smaller Study Area 2") # adds a second studyArea within 1st

## Add a 2nd pair of rasters
# rasTemplate <- rast(ext(studyArea(ml)), res = 0.001)
# tsf2 <- randomPolygons(rasTemplate, numTypes = 8)*30
# vtm2 <- randomPolygons(tsf2, numTypes = 4)
# levels(vtm2) <- data.frame(
#   ID = sort(unique(vtm2[])),
#   Factor = c("black spruce", "white spruce", "aspen", "fir")
# )

# ml <- mapAdd(tsf2, ml, filename2 = "tsf2.tif", layerName = "tsf2",
#              tsf = "tsf2.tif",
#              analysisGroup1 = "tsf2_vtm2", leaflet = TRUE, overwrite = TRUE)
# ml <- mapAdd(vtm2, ml, filename2 = "vtm2.grd", layerName = "vtm2",
#              vtm = "vtm2.grd",
#              analysisGroup1 = "tsf2_vtm2", leaflet = TRUE, overwrite = TRUE)

## post hoc analysis of data
##  use or create a specialized function that can handle the analysesData slot
# ml <- mapAddPostHocAnalysis(map = ml, functionName = "rbindlistAG",
#                             postHocAnalysisGroups = "analysisGroup2",
```

```
  #                               postHocAnalyses = "all")
}

## cleanup
withr::deferred_run()
```

`mapAddAnalysis` *Add an analysis to a* `map` *object*

Description

TODO: description needed

Usage

```
mapAddAnalysis(
  map,
  functionName,
  useParallel = getOption("map.useParallel", FALSE),
  ...
)
```

Arguments

`map`	A `map` object
`functionName`	The name of the analysis function to add
`useParallel`	Logical indicating whether to use multiple threads. Defaults to `getOption("map.useParallel", FALSE)`.
`...`	Additional arguments passed to `functionName`.

`mapAddPostHocAnalysis` *Add a post hoc analysis function to a* `map` *object*

Description

Add a post hoc analysis function to a `map` object

Usage

```
mapAddPostHocAnalysis(
  map,
  functionName,
  postHocAnalysisGroups = NULL,
  postHocAnalyses = "all",
  useParallel = getOption("map.useParallel", FALSE),
  ...
)
```

Arguments

map	Optional map object. If not provided, then one will be created. If provided, then the present obj or options passed to reproducible::prepInputs() (e.g., url), will be appended to this map.
functionName	A function that is designed for post hoc analysis of map class objects, e.g., rbindlistAG.
postHocAnalysisGroups	Character string with one analysisGroups, i.e., "analysisGroup1" or "analysisGroup2".
postHocAnalyses	Character vector with "all", (which will do all analysisGroups; default), or 1 or more of the the functionNames that are in the analyses slot.
useParallel	Logical. If TRUE, then if there is more than one calculation to do at any stage, it will create and use a parallel cluster via makeOptimalCluster(). If running analyses in parallel, it may be useful to pass a function (via .clInit) to be run on each of the nodes immediately upon cluster creation (e.g., to set options).
...	Optional arguments to pass into functionName

---

mapAnalysis *Generic analysis for map objects*

Description

This is the workhorse function that runs any analyses described in map@analyses. It uses hashing, and will not rerun any analysis that already ran on identical inputs.

Usage

```
mapAnalysis(
  map,
  functionName = NULL,
  purgeAnalyses = NULL,
  useParallel = getOption("map.useParallel", FALSE),
  ...
)
```

Arguments

map	Optional map object. If not provided, then one will be created. If provided, then the present obj or options passed to reproducible::prepInputs() (e.g., url), will be appended to this map.
functionName	A function name that will be run on combinations of inputs in the map object. See details.
purgeAnalyses	A character string indicating which analysis group combination or part thereof (e.g., the name entered into the row under analysisGroup2 column of the map@metadata or a functionName.

useParallel	Logical. If TRUE, then if there is more than one calculation to do at any stage, it will create and use a parallel cluster via makeOptimalCluster(). If running analyses in parallel, it may be useful to pass a function (via .clInit) to be run on each of the nodes immediately upon cluster creation (e.g., to set options).
...	Additional arguments passed to: reproducible::postProcess(), reproducible::projectInputs(), reproducible::fixErrors(), and reproducible::prepInputs().

Details

This function will do a sequence of things. First, it will run expand.grid on any columns whose names start with analysisGroup, creating a factorial set of analyses as described by these columns. It will assess the combinations against the arguments used by the functionName. For any analysisGroup that does not provide the correct arguments for the functionName, these analysisGroups will be omitted for that particular function. For efficiency, the function will then assess if any of these has already been run. For those that have not been run, it will then run the functionName on arguments that it finds in the metadata slot of the map object, as well as any arguments passed in here in the .... In general, the arguments being passed in here should be fixed across all analyses, while any that vary by analysis should be entered into the metadata table at the time of adding the layer to the map, via mapAdd.

Value

TODO

---

mapRm *Remove objects from a* map

Description

Remove objects from a map

Usage

```
mapRm(map, layer, ask = TRUE, ...)

## S3 method for class 'map'
mapRm(map, layer = NULL, ask = TRUE, ...)
```

Arguments

map	TODO: document this
layer	TODO: document this
ask	TODO: document this
...	TODO: document this

See Also

Other mapMethods: crs,map-method, rasterToMatch,map-method, rasters(), studyArea(), studyAreaName()

Other mapMethods: crs,map-method, rasterToMatch,map-method, rasters(), studyArea(), studyAreaName()

Examples

```
if (require("SpaDES.tools", quietly = TRUE)) {
  p <- terra::vect(cbind(-120, 60), crs = "epsg:4326") |>
    SpaDES.tools::randomPolygon(area = 1e5) |>
    sf::st_as_sf() |>
    sf::as_Spatial()
  m <- mapAdd(p, layerName = "p")
  m

  m <- mapRm(m, "p")
  m
}
```

`metadata` *Extract metadata*

Description

Extract metadata

Usage

```
metadata(x, ...)

## S3 method for class 'map'
metadata(x, ...)

## S3 method for class 'Map'
metadata(x, ...)

## S3 method for class 'Raster'
metadata(x, ...)

## S3 method for class 'SpatRaster'
metadata(x, ...)
```

Arguments

`x`	A `map`, `Raster`, or `SpatRaster` object
`...`	Additional arguments passed to other methods (not used)

rasters *Extract objects of specific classes from a* map *object*

Description

- rasters() extracts RasterLayer objects;
- sf() extracts sf objects;
- sp() extracts Spatial objects;
- spatialPolygons() extracts SpatialPolygons objects;
- spatialPoints() extracts SpatialPoints objects;
- spatRasters() extracts SpatRaster objects;
- spatVectors() extracts SpatVector objects;

This will extract all objects in or pointed to within the map.

Usage

```
rasters(map, ...)

## S3 method for class 'map'
rasters(map, ...)

sp(map, ...)

## S3 method for class 'map'
sp(map, ...)

sf(map, ...)

## S3 method for class 'map'
sf(map, ...)

spatialPolygons(map, ...)

## S3 method for class 'map'
spatialPolygons(map, ...)

spatialPoints(map, ...)

## S3 method for class 'map'
spatialPoints(map, ...)

spatRasters(map, ...)

## S3 method for class 'map'
spatRasters(map, ...)
```

```
spatVectors(map, ...)

## S3 method for class 'map'
spatVectors(map, ...)

maps(map, class = NULL, layerName = NULL)
```

Arguments

map	A map object
...	Additional arguments passed to other methods (not used)
class	If supplied, this will be the class of objects returned. Default is NULL which is "all", meaning all objects in the map object.
layerName	character string giving the name(s) of layer(s) to extract.

Value

A list of geospatial objects of class class

See Also

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, studyArea(), studyAreaName()

rasterToMatch,map-method *Extract the* rasterToMatch*(s) from a* map *object*

Description

If layer is not provided and there is more than one studyArea, then this will extract the last one added.

Usage

```
## S4 method for signature 'map'
rasterToMatch(x, layer = 1)
```

Arguments

x	a map object
layer	an integer identifying the index of the rasterToMatch to extract

See Also

Other mapMethods: crs,map-method, mapRm(), rasters(), studyArea(), studyAreaName()

rbindlistAG *Utility functions for grouping analyses in a* map *object*

Description

Utility functions for grouping analyses in a map object

Usage

```
rbindlistAG(map, functionName, analysisGroups)
```

Arguments

map	Optional map object. If not provided, then one will be created. If provided, then the present obj or options passed to reproducible::prepInputs() (e.g., url), will be appended to this map.
functionName	TODO: description needed
analysisGroups	A character (length 1 currently), indicating which analysis group (e.g., "analysisGroup1") should be used to rbindlist. Can also specify "all" which will rbindlist all outputs.

Value

A list of data.tables.

runMapAnalyses runMapAnalyses

Description

TODO: description needed

Usage

```
runMapAnalyses(
  map,
  purgeAnalyses = NULL,
  useParallel = getOption("map.useParallel", FALSE),
  ...
)
```

Arguments

map	TODO
purgeAnalyses	TODO
useParallel	TODO
...	TODO

Value

TODO

show,map-method *Show method for map class objects*

Description

Show method for map class objects

Usage

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

Arguments

object	TODO: describe this

studyArea *Extract the studyArea(s) from a* map

Description

If layer is not provided and there is more than one studyArea, then this will extract the last one added.

Usage

```
studyArea(map, layer = NA, sorted = FALSE)

## S4 method for signature 'ANY'
studyArea(map, layer = NA, sorted = FALSE)

## S4 method for signature 'map'
studyArea(map, layer = NA, sorted = FALSE)

studyArea(map, layer = NA) <- value

## S4 replacement method for signature 'map'
studyArea(map, layer = NA) <- value
```

Arguments

map	TODO: document this
layer	TODO: document this
sorted	Logical. Should the numeric layer be referring to geographic area of the area or the order that the studyArea were placed into map object
value	The value to assign to the object.

See Also

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, rasters(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, rasters(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, rasters(), studyAreaName()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, rasters(), studyAreaName()

studyAreaName *Map class methods*

Description

Tools for getting objects and metadata in and out of a map class.

Usage

```
studyAreaName(x, layer, ...)

## S3 method for class 'map'
studyAreaName(x, layer = 1, ...)

## S3 method for class 'data.table'
studyAreaName(x, layer = 1, ...)
```

Arguments

x	TODO: document this
layer	TODO: document this
...	Additional arguments passed to other methods (not used)

See Also

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, rasters(), studyArea()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, rasters(), studyArea()

Other mapMethods: crs,map-method, mapRm(), rasterToMatch,map-method, rasters(), studyArea()

Index