

Package: map (via r-universe)

October 9, 2026

Type Package

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

Version 1.1.2.9002

Date 2026-03-26

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.

License GPL-3

URL <https://predictiveecology.github.io/map/>,
<https://github.com/PredictiveEcology/map>

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

Depends R (>= 4.3)

Imports data.table (>= 1.10.4), digest, methods, parallelly, pemisc
(>= 0.0.4.9016), 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 PredictiveEcology/pemisc@development,
PredictiveEcology/reproducible@development,
PredictiveEcology/SpaDES.tools@development, ropensci/tiler

ByteCompile yes

Config/testthat/edition 3

Encoding UTF-8

Language en-CA

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

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

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make default-jdk libpng-dev libuv1-dev libssl-dev libproj-dev python3 libsqlite3-dev libudunits2-dev

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

Date/Publication 2026-09-09 19:45:07 UTC

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

RemoteRef development

RemoteSha a896da06f948933e324917875885daf84b269147

Contents

.rasterToMemory	3
addColumnNamesForLabels	3
areaAndPolyValue	4
buildMetadata	5
crs,map-method	6
fasterize2	6
gdal_polygonizeR	7
leafletTiles	8
makeTiles	8
map-class	9
mapAdd	9
mapAddAnalysis	13
mapAddPostHocAnalysis	14
mapAnalysis	15
mapRm	16
metadata	17
rasters	18
rasterToMatch,map-method	20
rbindlistAG	20
runMapAnalyses	21
show,map-method	21
studyArea	22
studyAreaName	23

Index	24
--------------	-----------

.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

addColumnNamesForLabels	<i>Add shinyLabel column (attribute) to spatial vectors</i>
-------------------------	---

Description

Add shinyLabel column (attribute) to spatial vectors

Usage

```
addColumnNamesForLabels(x, columnNamesForLabels, ...)

## Default S3 method:
addColumnNamesForLabels(x, columnNamesForLabels, ...)

## S3 method for class 'list'
addColumnNamesForLabels(x, columnNamesForLabels, ...)

## S3 method for class 'sf'
addColumnNamesForLabels(x, columnNamesForLabels, ...)

## S3 method for class 'SpatialPolygonsDataFrame'
addColumnNamesForLabels(x, columnNamesForLabels, ...)
```

```
## S3 method for class 'SpatVector'
addColumnForLabels(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	<i>Determine the area of each zone in a raster</i>
------------------	--

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	<i>Build map object metadata table</i>
---------------	--

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	<i>Extract the CRS of a map</i>
-----------------	---------------------------------

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	<i>Rasterize following crop and reproject</i>
------------	---

Description

A simple wrapper around [terra::rasterize\(\)](#).

Usage

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

Arguments

emptyRaster	An empty RasterLayer or SpatRaster 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 be 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

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

Value

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

leafletTiles	<i>Extract leaflet tile paths from a map obj</i>
--------------	--

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	<i>Make raster tiles (pyramids)</i>
-----------	-------------------------------------

Description

If python with GDAL support is available, uses gdal2tiles via [tiler::tile\(\)](#) to create tiles for use with e.g. leaflet.

Usage

```
makeTiles(
  tilePath = getOption("map.tilePath", "tiles"),
  obj,
  overwrite = FALSE,
  ...
)

canMakeTiles()
```

Arguments

tilePath	A directory 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).

Value

- `makeTiles()` is invoked for its side-effect of creating tiles in `tilePath`;
- `canMakeTiles()` returns a logical indicating the availability of python and GDAL support;

Note

Requires python with GDAL support (see `tiler::tiler_options()`). Windows users should install 'OSGeo4W' (<https://trac.osgeo.org/osgeo4w/>).

map-class	<i>The map class</i>
-----------	----------------------

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 list of `data.table` or `data.frame` of the results of the analyses.

mapAdd	<i>Append a spatial object to map</i>
--------	---------------------------------------

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 <code>reproducible::prepInputs()</code> (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 <code>getOption("map.overwrite")</code> .
...	Additional arguments passed to: <code>reproducible::postProcess()</code> , <code>reproducible::projectInputs()</code> , <code>reproducible::fixErrors()</code> , and <code>reproducible::prepInputs()</code> .
columnNameForLabels	A character string indicating which column to use for labels. This is currently only used if the object is a <code>sp::SpatialPolygonsDataFrame</code> .
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 <code>gdal2tiles</code> , making tiles. If path is not specified, it will be the current path. The tile base file path will be <code>paste0(layerName, "_", rndstr(1, 6))</code> .
isStudyArea	Logical. If TRUE, this will be assigned the label "StudyArea", and will be passed into <code>reproducible::prepInputs()</code> 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 <code>reproducible::Cache()</code> 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 `pemisc::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

```
ex_dir <- withr::local_tempdir("example_mapAdd_")
withr::local_dir(ex_dir)

withr::local_options(list(
  map.tilePath = file.path(ex_dir, "tiles"),
  map.useParallel = FALSE
))

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")

m1 <- mapAdd(
  StudyArea,
  isStudyArea = TRUE,
  layerName = "Small Study Area",
  poly = TRUE,
  analysisGroup2 = "Small Study Area"
)

if (require("SpaDES.tools", quietly = TRUE)) {
  smallStudyArea <- SpaDES.tools::randomPolygon(studyArea(m1), 1e5)
  smallStudyArea$ID <- 1L
  smallStudyArea$shinyLabel <- "zone2"

  m1 <- mapAdd(
    smallStudyArea,
    m1,
    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(m1)), 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")
  )

  ## need python + gdal to generate leaflet tiles
  doLeaflet <- canMakeTiles()

  m1 <- mapAdd(
    tsf,
    m1,
    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 = doLeaflet, ## to column in map@metadata; used for visualizing in leaflet
    overwrite = TRUE
  )
  m1 <- mapAdd(
    vtm,
    m1,
    filename2 = "vtm1.grd",
    layerName = "vtm1",
    vtm = "vtm1.grd",
    analysisGroup1 = "tsf1_vtm1",
    leaflet = doLeaflet,
    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
  # m1 <- mapAddAnalysis(m1, 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
  # m1 <- mapAddAnalysis(m1, functionName = "LargePatches", ageClasses = ageClasses,
  #   id = "1", labelColumn = "shinyLabel",
  #   ageClassCutOffs = ageClassCutOffs)

  ## Add a second polygon, trigger
  # smallStudyArea2 <- randomPolygon(studyArea(m1), 1e5)
  # smallStudyArea2$ID <- 1L
  # smallStudyArea2$shinyLabel <- "zone2"

  ## 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
# m1 <- mapAdd(smallStudyArea2, m1, 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(m1), 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
# m1 <- mapAdd(smallStudyArea2, m1, 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(m1)), 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")
# )

# m1 <- mapAdd(tsf2, m1, filename2 = "tsf2.tif", layerName = "tsf2",
#             tsf = "tsf2.tif",
#             analysisGroup1 = "tsf2_vtm2", leaflet = doLeaflet, overwrite = TRUE)
# m1 <- mapAdd(vtm2, m1, filename2 = "vtm2.grd", layerName = "vtm2",
#             vtm = "vtm2.grd",
#             analysisGroup1 = "tsf2_vtm2", leaflet = doLeaflet, overwrite = TRUE)

## post hoc analysis of data
## use or create a specialized function that can handle the analysesData slot
# m1 <- mapAddPostHocAnalysis(map = m1, functionName = "rbindlistAG",
#                             postHocAnalysisGroups = "analysisGroup2",
#                             postHocAnalyses = "all")
}

## cleanup
withr::deferred_run()

```

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 <code>getOption("map.useParallel", FALSE)</code> .
...	Additional arguments passed to <code>functionName</code> .

`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 <code>reproducible::prepInputs()</code> (e.g., <code>url</code>), will be appended to this map.
functionName	A function that is designed for post hoc analysis of map class objects, e.g., <code>rbindlistAG</code> .

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 <code>pemisc::makeOptimalCluster()</code> . If running analyses in parallel, it may be useful to pass a function (via <code>.clInit</code>) to be run on each of the nodes immediately upon cluster creation (e.g., to set options).
...	Optional arguments to pass into functionName

mapAnalysis	<i>Generic analysis for map objects</i>
-------------	---

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 <code>reproducible::prepInputs()</code> (e.g., <code>url</code>), 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 <code>analysisGroup2</code> column of the <code>map@metadata</code> or a <code>functionName</code>).
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 <code>pemisc::makeOptimalCluster()</code> . If running analyses in parallel, it may be useful to pass a function (via <code>.clInit</code>) to be run on each of the nodes immediately upon cluster creation (e.g., to set options).
...	Additional arguments passed to: <code>reproducible::postProcess()</code> , <code>reproducible::projectInputs()</code> , <code>reproducible::fixErrors()</code> , and <code>reproducible::prepInputs()</code> .

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	<i>Remove objects from a map</i>
-------	----------------------------------

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	<i>Extract metadata</i>
----------	-------------------------

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, ...)
```

 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	<i>Extract the study area(s) from a map</i>
-----------	---

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	<i>Map class methods</i>
---------------	--------------------------

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

- * **mapMethods**
 - crs, map-method, 6
 - mapRm, 16
 - rasters, 18
 - rasterToMatch, map-method, 20
 - studyArea, 22
 - studyAreaName, 23
- .rasterToMemory, 3
- addColumnNamesForLabels, 3
- areaAndPolyValue, 4
- buildMetadata, 5
- canMakeTiles (makeTiles), 8
- crs, map-method, 6
- fasterize2, 6
- gdal_polygonizeR, 7
- leafletTiles, 8
- makeTiles, 8
- map-class, 9
- mapAdd, 9
- mapAddAnalysis, 13
- mapAddPostHocAnalysis, 14
- mapAnalysis, 15
- mapRm, 6, 16, 19, 20, 22, 23
- maps (rasters), 18
- metadata, 17
- pemisc::makeOptimalCluster(), 11, 15
- rasters, 6, 16, 18, 20, 22, 23
- rasterToMatch, map-method, 20
- rbindlist-analysisGroups (rbindlistAG), 20
- rbindlistAG, 20
- reproducible::Cache(), 10
- reproducible::fixErrors(), 10, 15
- reproducible::postProcess(), 10, 15
- reproducible::prepInputs(), 10, 14, 15, 20
- reproducible::projectInputs(), 8, 10, 15
- runMapAnalyses, 21
- sf (rasters), 18
- show, map-method, 21
- sp (rasters), 18
- sp::SpatialPolygonsDataFrame, 10
- spatialPoints (rasters), 18
- spatialPolygons (rasters), 18
- spatRasters (rasters), 18
- spatVectors (rasters), 18
- studyArea, 6, 16, 19, 20, 22, 23
- studyArea, ANY-method (studyArea), 22
- studyArea, map-method (studyArea), 22
- studyArea<- (studyArea), 22
- studyArea<- , map-method (studyArea), 22
- studyAreaName, 6, 16, 19, 20, 22, 23
- terra::rasterize(), 6
- tiler::tile(), 8
- tiler::tiler_options(), 9