Package 'antaresViz'

December 2, 2024

Type Package

Title Antares Visualizations

Version 0.18.3

Description Visualize results generated by Antares, a powerful open source software developed by RTE to simulate and study electric power systems (more information about 'Antares' here: <https://github.com/AntaresSimulatorTeam/Antares_Simulator>). This package provides functions that create interactive charts to help 'Antares' users visually explore the results of their simulations.

URL https://github.com/rte-antares-rpackage/antaresViz

BugReports https://github.com/rte-antares-rpackage/antaresViz/issues

License GPL (>= 3)

Encoding UTF-8

Depends antaresRead (>= 2.2.9), antaresProcessing (>= 0.13.0), spMaps (>= 0.5.0)

Imports dygraphs (>= 1.1.1.6), shiny (>= 0.13.0), plotly (>= 4.5.6), htmltools, htmlwidgets (>= 0.7.0), manipulateWidget (>= 0.10.0), leaflet (>= 1.1.0), sp (>= 2.0-0), sf, webshot, data.table, methods, lubridate, geojsonio, graphics, stats, leaflet.minicharts (>= 0.5.3), assertthat, rAmCharts, utils, lifecycle

RoxygenNote 7.2.2

Suggests testthat, covr, ggplot2, knitr, visNetwork, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Tatiana Vargas [aut, cre], Jalal-Edine Zawam [ctb], Francois Guillem [ctb], Benoit Thieurmel [ctb], Titouan Robert [ctb], Paul Plessiez [ctb],

Baptiste Seguinot [ctb],
RTE [cph]

Maintainer Tatiana Vargas <tatiana.vargas@rte-france.com>

Repository CRAN

Date/Publication 2024-12-02 10:00:19 UTC

Contents

`addShadows` *Add a shadow to map layers*

Description

This function adds a shadow to every svg element added to a leaflet map. It can greatly improve the lisibility of the map.

Usage

```
addShadows(map)
```

Arguments

`map`	A leaflet map object.

Value

The modified map object

Examples

```
require(leaflet)
require(leaflet.minicharts)

leaflet() %>%
  addTiles() %>%
  addFlows(0, 0, 1, 0, col= gray(0.9)) %>%
  addCircleMarkers(c(0, 1), c(0, 0), color = "white", fillOpacity = 1, stroke = FALSE) %>%
  addShadows()
```

`exchangesStack` *Plot the exchanges of an area*

Description

This function draws a stack representing the evolution of the exchanges of an area with its neighbours. Positive values denotes exports and negative values imports. User can either plot all flows from/to an area using the `default` stack or use a custom one. User can see available stacks with `exchangesStackAliases` and create new ones with `setExchangesStackAlias`.

Usage

```
exchangesStack(
  x,
  stack = "default",
  area = NULL,
  mcYear = "average",
  dateRange = NULL,
  colors = NULL,
  yMin = NULL,
  yMax = NULL,
  customTicks = NULL,
  main = NULL,
  ylab = NULL,
  unit = c("MWh", "GWh", "TWh"),
  compare = NULL,
  compareOpts = list(),
  interactive = getInteractivity(),
  legend = TRUE,
  legendId = sample(1e+09, 1),
  groupId = legendId,
  updateLegendOnMouseOver = TRUE,
```

```
  legendItemsPerRow = 5,
  width = NULL,
  height = NULL,
  xyCompare = c("union", "intersect"),
  h5requestFiltering = deprecated(),
  stepPlot = FALSE,
  drawPoints = FALSE,
  timeSteph5 = deprecated(),
  mcYearh5 = deprecated(),
  tablesh5 = deprecated(),
  language = "en",
  hidden = NULL,
  refStudy = NULL,
  ...
)

exchangesStackAliases()

setExchangesStackAlias(
  name,
  variables,
  colors,
  lines = NULL,
  lineColors = NULL,
  lineWidth = 3,
  description = NULL
)
```

Arguments

x	Object of class antaresData created with function [antaresRead::readAntares]. It is required to contain link data. If it also contains area data with column 'ROW BAL.', then exchanges with the rest of the world are also displayed on the chart.
stack	Name of the stack to use. If default, all flows available will be plotted. One can visualize available stacks with exchangesStackAliases
area	Name of a single area. The flows from/to this area will be drawn by the function.
mcYear	If x, contains multiple Monte-Carlo scenarios, this parameter determine which scenario is displayed. Must be an integer representing the index of the scenario or the word "average". In this case data are averaged.
dateRange	A vector of two dates. Only data points between these two dates are displayed. If NULL, then all data is displayed.
colors	Vector of colors with same length as parameter variables. If variables is an alias, then this argument should be NULL in order to use default colors.
yMin	numeric, the minimum value to be displayed on all y Axis. If NULL, the min value is automatically set
yMax	numeric, the maximum value to be displayed on all y Axis. If NULL, the max value is automatically set

customTicks numeric vector of the custom ticks values to be displayed on the y Axis. If NULL, the ticks are automatically generated

main Title of the graph.

ylab Title of the Y-axis.

unit Unit used in the graph. Possible values are "MWh", "GWh" or "TWh".

compare An optional character vector containing names of parameters. When it is set, two charts are outputed with their own input controls. Alternatively, it can be a named list with names corresponding to parameter names and values being list with the initial values of the given parameter for each chart. See details if you are drawing a map.

compareOpts List of options that indicates the number of charts to create and their position. Check out the documentation of [manipulateWidget::compareOptions] to see available options.

interactive LogicalValue. If TRUE, then a shiny gadget is launched that lets the user interactively choose the areas or districts to display.

legend Logical value indicating if a legend should be drawn. This argument is usefull when one wants to create a shared legend with [prodStackLegend()]

legendId Id of the legend linked to the graph. This argument is usefull when one wants to create a shared legend with [prodStackLegend()]

groupId Parameter that can be used to synchronize the horizontal zoom of multiple charts. All charts that need to be synchronized must have the same group.

updateLegendOnMouseOver
LogicalValue. If TRUE the legend will be updated when the mouse is over a stack. If FALSE the legend will be updated on a click

legendItemsPerRow
Number of elements to put in each row of the legend.

width Width of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.

height Height of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.

xyCompare Use when you compare studies, can be "union" or "intersect". If union, all of mcYears in one of studies will be selectable. If intersect, only mcYears in all studies will be selectable.

h5requestFiltering
Contains arguments used by default for h5 request, typically h5requestFiltering = list(links = getLinks(areas = myArea), mcYears = myMcYear)

stepPlot boolean, step style for curves.

drawPoints boolean, add points on graph

timeSteph5 character timeStep to read in h5 file. Only for Non interactive mode.

mcYearh5 numeric mcYear to read for h5. Only for Non interactive mode.

tablesh5 character tables for h5 ("areas" "links", "clusters" or "disticts"). Only for Non interactive mode.

language character language use for label. Defaut to 'en'. Can be 'fr'.

hidden	logical Names of input to hide. Defaut to NULL
refStudy	An object of class antaresData created with function [antaresRead::readAntares()] containing data for areas and or districts. Can also contains an opts who refer to a h5 file.
...	Other arguments for [manipulateWidget::manipulateWidget]
name	name of the stack to create or update
variables	A named list of expressions created with [base::alist]. The name of each element is the name of the variable to draw in the stacked graph. The element itself is an expression explaining how to compute the variable (see examples).
lines	A named list of expressions created with [base::alist] indicating how to compute the curves to display on top of the stacked graph. It should be NULL if there is no curve to trace or if parameter variables is an alias.
lineColors	Vector of colors with same length as parameter lines. This argument should be NULL if there is no curve to trace or if parameter variables is an alias.
lineWidth	Optionnal. Defaut to 3. Vector of width with same length as parameter lines (or only one value).
description	Description of the stack. It is displayed by function exchangesStackAliases.

Details

Compare argument can take following values :

- "mcYear"
- "main"
- "unit"
- "area"
- "legend"
- "stepPlot"
- "drawPoints"

Value

A htmlwidget of class dygraph. It can be modified with functions from package dygraphs.

Examples

```
library(antaresRead)
# with study test for example (study is in package antaresRead)
sourcedir <- system.file("testdata", package = "antaresRead")

# untar study in temp dir
path_latest <- file.path(tempdir(), "latest")
untar(file.path(sourcedir, "antares-test-study.tar.gz"), exdir = path_latest)

study_path <- file.path(path_latest, "test_case")
```

```
# set path to your Antares simulation
opts <- setSimulationPath(study_path)

if(interactive()){
  mydata <- readAntares(links = "all", timeStep = "daily")
  exchangesStack(mydata)

  # Also display exchanges with the rest of the world
  mydata <- readAntares(areas = "all", links = "all", timeStep = "daily")
  exchangesStack(mydata)

  # Use compare :
  exchangesStack(mydata, compare = "mcYear")
  exchangesStack(mydata, compare = "area")
  exchangesStack(mydata, compare = "unit")
  exchangesStack(mydata, compare = "legend")
  # Compare studies with refStudy argument
  exchangesStack(x = myData1, refStudy = myData2)
  exchangesStack(x = myData1, refStudy = myData2, interactive = FALSE)
  exchangesStack(x = list(myData2, myData3, myData4), refStudy = myData1)
 exchangesStack(x = list(myData2, myData3, myData4), refStudy = myData1, interactive = FALSE)


  # Compare 2 studies
  exchangesStack(x = list(opts, opts))

  # Compare 2 studies with argument refStudy
  exchangesStack(x = opts, refStudy = opts)
}
```

`limitSizeGraph` *Use to change limit size of graph (in Mb)*

Description

Use to change limit size of graph (in Mb)

Usage

```
limitSizeGraph(size)
```

Arguments

`size`	`numeric` widget size autorized in modules (default 200)

Examples

```
## Not run:
limitSizeGraph(500)

## End(Not run)
```

`mapLayout` *Place areas of a study on a map*

Description

This function launches an interactive application that let the user place areas of a study on a map. the GPS coordinates of the areas are then returned and can be used in functions. This function should be used only once per study. The result should then be saved in an external file and be reused.

Usage

```
mapLayout(
  layout,
  what = c("areas", "districts"),
  map = getSpMaps(),
  map_builder = TRUE
)
```

Arguments

`layout`	object returned by function [antaresRead::readLayout()]
`what`	Either "areas" or "districts". Indicates what type of object to place on the map.
`map`	An optional [sp::SpatialPolygons()] or [sp::SpatialPolygonsDataFrame()] object. See [spMaps::getSpMaps()]
`map_builder`	`logical` Add inputs for build custom map ? Defaut to TRUE.

Details

With `map_builder` option, you can build a quiet custom map using `spMaps` package. This package help you to build [sp::SpatialPolygons()] on Europe. Moreover, you can use two options in the module :

- "Merge countries" : Some countries like UK or Belgium are firstly rendered in multiple and diffrent area. You can so choose to finally use this countries as one single area on the map
- "Merge states" : If you need states details but not having one area per state, the map will be incomplete for some countries, plotting only states with area. So you can choose to aggregate the states of the countries. This is done using a nearest states algorithm. The result is available only after layout validation.

Value

An object of class `mapLayout`.

See Also

[plotMapLayout()]

Examples

```
## Not run:
# Read the coordinates of the areas in the Antares interface, then convert it
# in a map layout.
layout <- readLayout()
ml <- mapLayout(layout)

# visualize mapLayout
plotMapLayout(ml)

# Save the result for future use
save(ml, file = "ml.rda")


## End(Not run)
```

`modRpart` *Make rpart from antares data*

Description

Make rpart from antares data

Usage

```
modRpart(data)
```

Arguments

`data`	an antaresData after use of [antaresProcessing::mergeAllAntaresData]

Examples

```
## Not run:
setSimulationPath("Mystud", 1)
mydata <- readAntares(areas = "all", select = "OIL")
mydata <- mergeAllAntaresData(mydata)
modRpart(mydata)

## End(Not run)
```

modXY *Make X-Y bockey plot, interactive version*

Description

Make X-Y bockey plot, interactive version

Usage

```
modXY(x, xyCompare = c("union", "intersect"))
```

Arguments

x	optsH5 or list of optsH5
xyCompare	Use when you compare studies, can be "union" or "intersect". If union, all of mcYears in one of studies will be selectable. If intersect, only mcYears in all studies will be selectable.

Examples

```
## Not run:
opts <- setSimulationPath("h5File")
modXY(opts)
modXY(list(opts, opts))


## End(Not run)
```

placeGeoPoints-shiny *Shiny bindings for placeGeoPoints*

Description

Output and render functions for using placeGeoPoints within Shiny applications and interactive Rmd documents.

Usage

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

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

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
expr	An expression that generates a placeGeoPoints
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.

---

plot.mapLayout *Plot method for map layout*

Description

This method can be used to visualize the network of an antares study. It generates an interactive map with a visual representaiton of a map layout created with function [mapLayout()].

Usage

```
## S3 method for class 'mapLayout'
plot(
  x,
  colAreas = x$coords$color,
  dataAreas = 1,
  opacityArea = 1,
  areaMaxSize = 30,
  areaMaxHeight = 50,
  areaChartType = c("auto", "bar", "pie", "polar-area", "polar-radius"),
  labelArea = NULL,
  labelMinSize = 8,
  labelMaxSize = 8,
  colLinks = "#CCCCCC",
  sizeLinks = 3,
  opacityLinks = 1,
  dirLinks = 0,
  links = TRUE,
  areas = TRUE,
  tilesURL = defaultTilesURL(),
  preprocess = function(map) {
     map
 },
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

x	Object created with function [mapLayout()]
colAreas	Vector of colors for areas. By default, the colors used in the Antares software are used.
dataAreas	A numeric vector or a numeric matrix that is passed to function link[addMinicharts]. A single vector will produce circles with different radius. A matrix will produce bar charts or pie charts or polar charts, depending on the value of areaChartType
opacityArea	Opacity of areas. It has to be a numeric vector with values between 0 and 1.
areaMaxSize	Maximal width in pixels of the symbols that represent areas on the map.
areaMaxHeight	Maximal height of bars. Used only if a barchart representation is used.
areaChartType	Type of chart to use to represent areas.
labelArea	Character vector containing labels to display inside areas.
labelMinSize	minimal height of labels.
labelMaxSize	maximal height of labels.
colLinks	Vector of colors for links.
sizeLinks	Line width of the links, in pixels.
opacityLinks	Opacity of the links. It has to be a numeric vector with values between 0 and 1.
dirLinks	Single value or vector indicating the direction of the link. Possible values are 0, -1 and 1. If it equals 0, then links are repsented by a simple line. If it is equal to 1 or -1 it is represented by a line with an arrow pointing respectively the destination and the origin of the link.
links	Should links be drawn on the map ?
areas	Should areas be drawn on the map ?
tilesURL	URL template used to get map tiles. The followign site provides some URLs; https://leaflet-extras.github.io/leaflet-providers/preview/
preprocess	A function that takes as argument a map and that returns a modified version of this map. This parameter can be used to add extra information on a map.
width	Width of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.
height	Height of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.
...	Currently unused.

Value

The function generates an htmlwidget of class leaflet. It can be stored in a variable and modified with package [leaflet::leaflet()]

Examples

```
## Not run:
# Read the coordinates of the areas in the Antares interface, then convert it
# in a map layout.
layout <- readLayout()
ml <- mapLayout(layout)

# Save the result for future use
save(ml, file = "ml.rda")

# Plot the network on an interactive map
plot(ml)

# change style
plot(ml, colAreas = gray(0.5), colLinks = "orange")

# Use polar area charts to represent multiple values for each area.
nareas <- nrow(ml$coords)
fakeData <- matrix(runif(nareas * 3), ncol = 3)
plot(ml, sizeAreas = fakeData)

# Store the result in a variable to change it with functions from leaflet
# package
library(leaflet)

center <- c(mean(ml$coords$x), mean(ml$coords$y))

p <- plot(ml)
p %>%
  addCircleMarker(center[1], center[2], color = "red",
                  popup = "I'm the center !")

## End(Not run)
```

---

plotMap *Display results of a simulation on a map*

Description

This function generates an interactive map that let the user visually explore the results of an Antares simulation. By default the function starts a Shiny gadget that let the user which variables to represent.

Usage

```
plotMap(
  x,
  refStudy = NULL,
```

```
  mapLayout,
  colAreaVar = "none",
  sizeAreaVars = c(),
  areaChartType = c("bar", "pie", "polar-area", "polar-radius"),
  uniqueScale = FALSE,
  showLabels = FALSE,
  popupAreaVars = c(),
  labelAreaVar = "none",
  colLinkVar = "none",
  sizeLinkVar = "none",
  popupLinkVars = c(),
  closePopupOnClick = TRUE,
  type = c("detail", "avg"),
  timeId = NULL,
  mcYear = "average",
  main = "",
  typeSizeAreaVars = FALSE,
  aliasSizeAreaVars = c(),
  compare = NULL,
  compareOpts = list(),
  interactive = getInteractivity(),
  options = plotMapOptions(),
  width = NULL,
  height = NULL,
  dateRange = NULL,
  xyCompare = c("union", "intersect"),
  h5requestFiltering = deprecated(),
  timeSteph5 = deprecated(),
  mcYearh5 = deprecated(),
  tablesh5 = deprecated(),
  sizeMiniPlot = FALSE,
  language = "en",
  hidden = NULL,
  ...
)
```

Arguments

x	Object of class antaresDataList created with [antaresRead::readAntares()] and containing areas and links data. It can be a list of antaresData objects. In this case, one chart is created for each object.
refStudy	An object of class antaresData created with function [antaresRead::readAntares()] containing data for areas and or districts. Can also contains an opts who refer to a h5 file.
mapLayout	Object created with function [antaresViz::mapLayout()]
colAreaVar	Name of a variable present in x$areas. The values of this variable are represented by the color of the areas on the map. If "none", then the default color is used for all areas.

`sizeAreaVars` Vector of variables present in `x$areas` to associate with the size of areas on the map. If this parameter has length equal to 0, all areas have the same size. If it has length equal to one, then the radius of the areas change depending on the values of the variable choosen. If it has length greater than 1 then areas are represented by a polar area chart where the size of each section depends on the values of each variable.

`areaChartType` If parameter `sizeAreaVars` contains multiple variables, this parameter determines the type of representation. Possible values are `"bar"` for bar charts, `"pie"` for pie charts, `"polar-area"` and `"polar-radius"` for polar area charts where the values are represented respectively by the area or the radius of the slices.

`uniqueScale` If the map contains polar or bar charts, should the different variables represented use the same scale or should each variable have its own scale ? This parameter should be TRUE only if the variables have the same unit and are comparable : for instance production variables.

`showLabels` Used only when `sizeAreaVars` contains multiple variables. If it is `TRUE`, then values of each variable are displayed.

`popupAreaVars` Vector of variables to display when user clicks on an area.

`labelAreaVar` Variable to display inside the areas. This parameter is used only if parameter `sizeAreaVars` contains zero or one variable.

`colLinkVar` Name of a variable present in `x$links`. The values of this variable are represented by the color of the links on the map. If `"none"`, then the default color is used for all links

`sizeLinkVar` Name of a variable present in `x$links`. Its values are represented by the line width of the links on the map.

`popupLinkVars` Vector of variables to display when user clicks on a link

`closePopupOnClick`
`LogicalValue`, if `TRUE` the popups will automatically be closed with each click. If `FALSE`, the popups will stay open.

`type` If `type="avg"`, the data is averaged by area/and or link and represented on the map. If it is equal to `"detail"`, only one time step at a time. In interactive mode, an input control permits to choose the time step shown.

`timeId` time id present in the data.

`mcYear` If `x`, contains multiple Monte-Carlo scenarios, this parameter determine which scenario is displayed. Must be an integer representing the index of the scenario or the word "average". In this case data are averaged.

`main` Title of the map.

`typeSizeAreaVars`
`logical`. Select `sizeAreaVars` using alias ? Default to `FALSE`

`aliasSizeAreaVars`
If `typeSizeAreaVars` is set to TRUE, name of alias. You can find the list of alias with the function [antaresRead::showAliases()]

`compare` An optional character vector containing names of parameters. When it is set, two charts are outputed with their own input controls. Alternatively, it can be a named list with names corresponding to parameter names and values being list

with the initial values of the given parameter for each chart. See details if you are drawing a map.

`compareOpts` List of options that indicates the number of charts to create and their position. Check out the documentation of [manipulateWidget::compareOptions] to see available options.

`interactive` LogicalValue. If `TRUE`, then a shiny gadget is launched that lets the user interactively choose the areas or districts to display.

`options` List of parameters that override some default visual settings. See the help of [antaresViz::plotMapOptions()].

`width` Width of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.

`height` Height of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.

`dateRange` A vector of two dates. Only data points between these two dates are displayed. If NULL, then all data is displayed.

`xyCompare` Use when you compare studies, can be "union" or "intersect". If union, all of mcYears in one of studies will be selectable. If intersect, only mcYears in all studies will be selectable.

`h5requestFiltering`
Contains arguments used by default for h5 request, typically h5requestFiltering = list(mcYears = 3)

`timeSteph5` `character` timeStep to read in h5 file. Only for Non interactive mode.

`mcYearh5` `numeric` mcYear to read for h5. Only for Non interactive mode.

`tablesh5` `character` tables for h5 ("areas" "links", "clusters" or "disticts"). Only for Non interactive mode.

`sizeMiniPlot` `boolean` variable size for miniplot

`language` `character` language use for label. Defaut to 'en'. Can be 'fr'.

`hidden` `logical` Names of input to hide. Defaut to NULL

`...` Other arguments for [manipulateWidget::manipulateWidget]

Details

compare argument can take following values :

- "mcYear"
- "type"
- "colAreaVar"
- "sizeAreaVars"
- "areaChartType"
- "showLabels"
- "popupAreaVars"
- "labelAreaVar"

- "colLinkVar"
- "sizeLinkVar"
- "popupLinkVars"
- "typeSizeAreaVars"
- "aliasSizeAreaVars"

Value

An htmlwidget of class "leaflet". It can be modified with package `leaflet`. By default the function starts a shiny gadget that lets the user play with most of the parameters of the function. The function returns a leaflet map when the user clicks on the button `"OK"`.

Examples

```
## Not run:
mydata <- readAntares(areas = "all", links = "all", timeStep = "daily",
                      select = "nostat")

# Place areas on a map. Ths has to be done once for a given study. Then the
# object returned by "mapLayout" may be saved and reloaded with
# functions save and load

layout <- readLayout()
ml <- mapLayout(layout = layout)
save("ml", file = "ml.rda")

plotMap(x = mydata, mapLayout = ml)

# Specify the variables to use to control the color or size of elements.
plotMap(mydata, mapLayout =  ml,
        sizeAreaVars = c("WIND", "SOLAR", "H. ROR"),
        sizeLinkVar = "FLOW LIN.")

# Change default graphical properties
plotMap(x = mydata, mapLayout = ml, options = list(colArea="red", colLink = "orange"))
plotMap(x = list(mydata, mydata), mapLayout =  ml)

# Use custom alias
setAlias("custom_alias", "short description", c("OIL", "GAS", "COAL"))
plotMap(x = mydata, mapLayout = ml, typeSizeAreaVars = TRUE,
    aliasSizeAreaVars = "custom_alias")

plotMap(x = mydata, mapLayout = ml, interactive = FALSE,
    language = "fr", aliasSizeAreaVars = "Renouvelable", typeSizeAreaVars = TRUE)

# Use h5 for dynamic request / exploration in a study
# Set path of simulaiton
setSimulationPath(path = path1)

# Convert your study in h5 format
writeAntaresH5(path = myNewPath)
```

```
# Redefine sim path with h5 file
opts <- setSimulationPath(path = myNewPath)
plotMap(x = opts, mapLayout = ml)

# Compare elements in a single study
plotMap(x = opts, mapLayout = ml,  .compare = "mcYear")

# Compare 2 studies
plotMap(x = list(opts, opts2), mapLayout = ml)

# Compare 2 studies with argument refStudies
plotMap(x = opts, refStudy = opts2, mapLayout = ml)
plotMap(x = opts, refStudy = opts2, mapLayout = ml, interactive = FALSE, mcYearh5 = 2)
plotMap(x = opts, refStudy = opts2, mapLayout = ml, h5requestFiltering =
list(mcYears = myMcYear))

## End(Not run)
```

`plotMapLayout` *Visualize mapLayout output.*

Description

Visualize mapLayout output.

Usage

```
plotMapLayout(mapLayout)
```

Arguments

`mapLayout`	object returned by function [mapLayout()]

See Also

[mapLayout()]

Examples

```
## Not run:
# Read the coordinates of the areas in the Antares interface, then convert it
# in a map layout.
layout <- readLayout()
ml <- mapLayout(layout)

# visualize mapLayout
plotMapLayout(ml)
```

```
## End(Not run)
```

---

plotMapOptions *Graphical options for plotMap*

Description

These functions get and set options that control some graphical aspects of maps created with [plotMap()].

Usage

```
plotMapOptions(
  areaDefaultCol = "#DDDDE5",
  areaDefaultSize = 30,
  areaMaxSize = 50,
  areaMaxHeight = 50,
  areaChartColors = NULL,
  areaColorScaleOpts = colorScaleOptions(),
  labelMinSize = 8,
  labelMaxSize = 24,
  linkDefaultCol = "#BEBECE",
  linkDefaultSize = 3,
  linkMaxSize = 15,
  linkColorScaleOpts = colorScaleOptions(),
  legend = c("choose", "visible", "hidden"),
  tilesURL = defaultTilesURL(),
  preprocess = function(map) {
     map
 }
)

defaultTilesURL()

colorScaleOptions(
  breaks = 5,
  domain = NULL,
  negCol = "#FF0000",
  zeroCol = "#FAFAFA",
  posCol = "#0000FF",
  naCol = "#EEEEEE",
  zeroTol = NULL,
  colors = NULL,
  levels = NULL
)
```

Arguments

areaDefaultCol	default color of areas.
areaDefaultSize	default size of areas.
areaMaxSize	maximal size of an area when it represents the value of some variable.
areaMaxHeight	Maximal height of bars. Used only if a barchart representation is used.
areaChartColors	Vector of colors to use in polar area charts and bar charts
areaColorScaleOpts	List of options used to construct a continuous color scale. This list should be generated with function colorScaleOptions.
labelMinSize	minimal height of labels.
labelMaxSize	maximal height of labels.
linkDefaultCol	Default color of links.
linkDefaultSize	Default line width of links.
linkMaxSize	Maximal line width of a link when it represents the value of some variable.
linkColorScaleOpts	List of options used to construct a continuous color scale. This list should be generated with function colorScaleOptions.
legend	Should the legend be displayed or not ? Default is to mask the legend but add a button to display it. Other values are "visible" to make the legend always visible and "hidden" to mask it.
tilesURL	URL template used to get map tiles. The followign site provides some URLs; https://leaflet-extras.github.io/leaflet-providers/preview/
preprocess	A function that takes as argument a map and that returns a modified version of this map. This parameter can be used to add extra information on a map.
breaks	Either a single number indicating the approximate number of colors to use, or a vector of values at which values to change color. In the first case, the function tries to cut the data nicely, so the real number of colors used may vary.
domain	Range of the data, ie. the range of possible values. If NULL, the the range of the data is used
negCol	color of the extreme negative value.
zeroCol	color of the 0 value.
posCol	Color of the extreme positive value.
naCol	Color for missing values
zeroTol	All values in the interval \[-zeroTol, +zeroTol\] are mapped to the zeroCol color. If NULL, the function tries to pick a nice value that is approximately equal to 1% of the maximal value.
colors	Vector of colors. If it is set and if user manually sets break points, then these colors are used instead of the colors defined by parameters negCol, zeroCol and posCol.
levels	Vector of the distinct values a variable can take. Only used when the variable to represent is a categorical variable.

Value

A list with the values of the different graphical parameters.

Examples

```
## Not run:

# Example : Change color for area variables

library(antaresViz)

studyPath <- "path/to/study"
setSimulationPath(path = studyPath, simulation = -1)
myData<-readAntares(areas = "all", links = "all")

ml<-readRDS(file = "path/to/mapLayout.rds")

myOption<-plotMapOptions(areaChartColors = c("yellow", "violetred"))

plotMap(myData,
        ml,
        sizeAreaVars = c("SOLAR", "WIND"),
        type="avg",
        interactive = FALSE,
       options = myOption
)

# for pie chart
plotMap(myData,
        ml,
        sizeAreaVars = c("SOLAR", "WIND"),
        type="avg",
        interactive = FALSE,
        options = myOption,
        areaChartType = "pie",
        sizeMiniPlot = TRUE
)

# Example : Change color for link and area variables
myOption <- plotMapOptions(areaChartColors = c("yellow", "violetred"), linkDefaultCol = "green")
plotMap(myData,
        ml,
        type="avg",
        sizeAreaVars = c("SOLAR", "WIND"),
        interactive = FALSE,
        options = myOption
)

# Change default area color
myOption <- plotMapOptions(areaDefaultCol = "green")
plotMap(myData,
        ml,
```

```
        interactive = FALSE,
        options = myOption
)

# Change the scale
plotMap(myData,
        ml,
        colAreaVar = "MRG. PRICE",
        options = plotMapOptions(
          areaColorScaleOpts = colorScaleOptions(
            breaks = c(-1000, 100, 200, 20000),
            colors = c("green", "orange", "red")
          )
        ),
        interactive = FALSE
)

## End(Not run)
```

`plotThermalGroupCapacities` *Plot for Thermal Group Capacities*

Description

Plot for Thermal Group Capacities

Usage

```
plotThermalGroupCapacities(
  data,
  area = "all",
  main = "Thermal group capacities"
)
```

Arguments

`data`	data.table of Thermal Group capacities
`area`	areas to select, default all
`main`	title

Examples

```
## Not run:
opts <- setSimulationPath(getwd())
plotThermalGroupCapacities( thermalGroupCapacities(opts))

## End(Not run)
```

plotXY *Plot density between X et Y with ggplot2 and plotly*

Description

Plot density between X et Y with ggplot2 and plotly

Usage

```
plotXY(
  data,
  x,
  y,
  precision = 30,
  sizeOnCount = FALSE,
  outLine = TRUE,
  transform = NULL
)
```

Arguments

data	data.frame can be antaresData object
x	character, x variable
y	character, y variable
precision	Deprecated.
sizeOnCount	Deprecated.
outLine	Deprecated.
transform	Deprecated.

Examples

```
## Not run:

setSimulationPath("myStudy")
myData <- readAntares()

plotXY(myData, "NODU", "LOAD", precision = 50,
         sizeOnCount = FALSE)

myData <- readAntares(areas = "all", links = "all")
myData <- mergeAllAntaresData(myData)
plotXY(myData, "OP. COST_max_b", "OP. COST_max_c", precision = 50,
         sizeOnCount = FALSE)



## End(Not run)
```

prodStackAliases *Visualize the production stack of an area*

Description

prodStack draws the production stack for a set of areas or districts. User can see available stacks with prodStackAliases and create new ones with setProdStackAlias.

Usage

```
prodStackAliases()

setProdStackAlias(
  name,
  variables,
  colors,
  lines = NULL,
  lineColors = NULL,
  lineWidth = 3,
  description = NULL
)

prodStack(
  x,
  stack = "eco2mix",
  areas = NULL,
  mcYear = "average",
  dateRange = NULL,
  yMin = NULL,
  yMax = NULL,
  customTicks = NULL,
  main = .getLabelLanguage("Production stack", language),
  unit = c("MWh", "GWh", "TWh"),
  compare = NULL,
  compareOpts = list(),
  interactive = getInteractivity(),
  legend = TRUE,
  legendId = sample(1e+09, 1),
  groupId = legendId,
  updateLegendOnMouseOver = TRUE,
  legendItemsPerRow = 5,
  width = NULL,
  height = NULL,
  xyCompare = c("union", "intersect"),
  h5requestFiltering = deprecated(),
  stepPlot = FALSE,
  drawPoints = FALSE,
```

```
  timeSteph5 = deprecated(),
  mcYearh5 = deprecated(),
  tablesh5 = deprecated(),
  language = "en",
  hidden = NULL,
  refStudy = NULL,
  ...
)
```

Arguments

name	name of the stack to create or update
variables	A named list of expressions created with [base::alist]. The name of each element is the name of the variable to draw in the stacked graph. The element itself is an expression explaining how to compute the variable (see examples).
colors	Vector of colors with same length as parameter variables. If variables is an alias, then this argument should be NULL in order to use default colors.
lines	A named list of expressions created with [base::alist] indicating how to compute the curves to display on top of the stacked graph. It should be NULL if there is no curve to trace or if parameter variables is an alias.
lineColors	Vector of colors with same length as parameter lines. This argument should be NULL if there is no curve to trace or if parameter variables is an alias.
lineWidth	Optionnal. Defaut to 3. Vector of width with same length as parameter lines (or only one value).
description	Description of the stack. It is displayed by function prodStackAliases.
x	An object of class antaresData created with function [antaresRead::readAntares()] containing data for areas and or districts. it can be a list of antaresData objects. In this case, one chart is created for each object. Can also contains opts who refer to a h5 file or list of opts.
stack	Name of the stack to use. One can visualize available stacks with prodStackAliases
areas	Vector of area or district names. The data of these areas or districts is aggregated by the function to construct the production stack.
mcYear	If x, contains multiple Monte-Carlo scenarios, this parameter determine which scenario is displayed. Must be an integer representing the index of the scenario or the word "average". In this case data are averaged.
dateRange	A vector of two dates. Only data points between these two dates are displayed. If NULL, then all data is displayed.
yMin	numeric, the minimum value to be displayed on all y Axis. If NULL, the min value is automatically set
yMax	numeric, the maximum value to be displayed on all y Axis. If NULL, the max value is automatically set
customTicks	numeric vector of the custom ticks values to be displayed on the y Axis. If NULL, the ticks are automatically generated
main	Title of the graph.

`unit` Unit used in the graph. Possible values are "MWh", "GWh" or "TWh".

`compare` An optional character vector containing names of parameters. When it is set, two charts are outputed with their own input controls. Alternatively, it can be a named list with names corresponding to parameter names and values being list with the initial values of the given parameter for each chart. See details if you are drawing a map.

`compareOpts` List of options that indicates the number of charts to create and their position. Check out the documentation of [manipulateWidget::compareOptions] to see available options.

`interactive` LogicalValue. If `TRUE`, then a shiny gadget is launched that lets the user interactively choose the areas or districts to display.

`legend` Logical value indicating if a legend should be drawn. This argument is usefull when one wants to create a shared legend with [prodStackLegend()]

`legendId` Id of the legend linked to the graph. This argument is usefull when one wants to create a shared legend with [prodStackLegend()]

`groupId` Parameter that can be used to synchronize the horizontal zoom of multiple charts. All charts that need to be synchronized must have the same group.

`updateLegendOnMouseOver`
`LogicalValue`. If `TRUE` the legend will be updated when the mouse is over a stack. If `FALSE` the legend will be updated on a click

`legendItemsPerRow`
Number of elements to put in each row of the legend.

`width` Width of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.

`height` Height of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.

`xyCompare` Use when you compare studies, can be "union" or "intersect". If union, all of mcYears in one of studies will be selectable. If intersect, only mcYears in all studies will be selectable.

`h5requestFiltering`
Contains arguments used by default for h5 request, typically h5requestFiltering = list(areas = "a", mcYears = 2)

`stepPlot` `boolean`, step style for curves.

`drawPoints` `boolean`, add points on graph

`timeSteph5` `character` timeStep to read in h5 file. Only for Non interactive mode.

`mcYearh5` `numeric` mcYear to read for h5. Only for Non interactive mode.

`tablesh5` `character` tables for h5 ("areas" "links", "clusters" "disticts"). Only for Non interactive mode.

`language` `character` language use for label. Defaut to 'en'. Can be 'fr'.

`hidden` `logical` Names of input to hide. Defaut to NULL

`refStudy` An object of class `antaresData` created with function [antaresRead::readAntares()] containing data for areas and or districts. Can also contains an opts who refer to a h5 file.

`...` Other arguments for [manipulateWidget::manipulateWidget]

Details

compare argument can take following values :

- "mcYear"
- "main"
- "unit"
- "areas"
- "legend"
- "stack"
- "stepPlot"
- "drawPoints"

Value

`prodStack` returns an interactive html graphic. If argument `interactive` is `TRUE`, then a shiny gadget is started and the function returns an interactive html graphic when the user clicks on button "Done".

`prodStackAliases` displays the list of available aliases.

`setProdStackAlias` creates or updates a stack alias.

See Also

[prodStackLegend()]

Examples

```
## Not run:
mydata <- readAntares(areas = "all", timeStep = "daily")

# Start a shiny gadget that permits to choose areas to display.
prodStack(x = mydata, unit = "GWh")

# Use in a non-interactive way
prodStack(x = mydata, unit = "GWh", areas = "fr", interactive = FALSE)

# Define a custom stack
setProdStackAlias(
  name = "Wind and solar",
  variables = alist(wind = WIND, solar = SOLAR),
  colors = c("green", "orange")
)

prodStack(x = mydata, unit = "GWh", stack = "Wind and solar")

# In a custom stack it is possible to use computed values
setProdStackAlias(
  name = "Renewable",
  variables = alist(
```

```
    renewable = WIND + SOLAR + `H. ROR` + `H. STOR` + `MISC. NDG`,
    thermal = NUCLEAR + LIGNITE + COAL + GAS + OIL + `MIX. FUEL` + `MISC. DTG`
  ),
  colors = c("green", gray(0.3)),
  lines = alist(goalRenewable = LOAD * 0.23),
  lineColors = "#42EB09"
)

prodStack(x = mydata, unit = "GWh", stack = "renewable")

# Use compare
prodStack(x = mydata, compare = "areas")
prodStack(x = mydata, unit = "GWh", compare = "mcYear")
prodStack(x = mydata, unit = "GWh", compare = "main")
prodStack(x = mydata, unit = "GWh", compare = "unit")
prodStack(x = mydata, unit = "GWh", compare = "areas")
prodStack(x = mydata, unit = "GWh", compare = "legend")
prodStack(x = mydata, unit = "GWh", compare = "stack")
prodStack(x = mydata, unit = "GWh", compare = c("mcYear", "areas"))


# Compare studies
prodStack(list(mydata, mydata))
# Compare studies with refStudy argument
prodStack(x = myData1, refStudy = myData2)
prodStack(x = myData1, refStudy = myData2, interactive = FALSE)
prodStack(x = list(myData2, myData3, myData4), refStudy = myData1)
prodStack(x = list(myData2, myData3, myData4), refStudy = myData1, interactive = FALSE)


# Use h5 opts
# Set path of simulaiton
setSimulationPath(path = path1)

# Convert your study in h5 format
writeAntaresH5(path = mynewpath)

# Redefine sim path with h5 file
opts <- setSimulationPath(path = mynewpath)
prodStack(x = opts)

# Compare elements in a single study
prodStack(x = opts, .compare = "mcYear")

# Compare 2 studies
prodStack(x = list(opts, opts2))

# Compare 2 studies with argument refStudies
prodStack(x = opts, refStudy = opts2)
prodStack(x = opts, refStudy = opts2, interactive = FALSE, mcYearh5 = 2, areas = myArea)
prodStack(x = opts, refStudy = opts2, h5requestFiltering = list(areas = myArea,
mcYears = 2))
```

```
## End(Not run)
```

`prodStackExchangesLegend` *Plot an interactive legend for time series plots*

Description

These functions create a nice looking legend that displays values when the user hovers a time series produced with plot this package. By default, the different functions already output a legend. This function is mostly useful to share a unique legend between two or more time series plots.

Usage

```
prodStackExchangesLegend(
  stack = "default",
  legendItemsPerRow = 5,
  legendId = "",
  language = "en"
)

prodStackLegend(
  stack = "eco2mix",
  legendItemsPerRow = 5,
  legendId = "",
  language = "en"
)

tsLegend(labels, colors, types = "line", legendItemsPerRow = 5, legendId = "")
```

Arguments

`stack`	Name of the stack to use. One can visualize available stacks with `prodStackAliases`
`legendItemsPerRow`	Number of elements to put in each row of the legend.
`legendId`	Id of the legend linked to the graph. This argument is usefull when one wants to create a shared legend with [prodStackLegend()]
`language`	`character` language use for label. Defaut to 'en'. Can be 'fr'.
`labels`	vector containing the names of the times series
`colors`	vector of colors. It must have the same length as parameter `labels`.
`types`	"line" or "area" or a vector with same length as `labels` containing these two values.

Details

Thes functions can be used to create a legend shared by multiple plots in a Shiny application or an interactive document created with Rmarkdown. For instance, let assume one wants to display four productions stacks in a 2x2 layout and have a unique legend below them in a Rmarkdown document. To do so, one can use the following chunck code:

```
```{R, echo = FALSE}
library(manipulateWidget)

combineWidgets(
  prodStack(mydata, areas = "fr",
            main = "Production stack in France", unit = "GWh",
            legend = FALSE, legendId = 1, height = "100%", width = "100%"),
  prodStack(mydata, areas = "de",
            main = "Production stack in Germany", unit = "GWh",
            legend = FALSE, legendId = 1, height = "100%", width = "100%"),
  prodStack(mydata, areas = "es",
            main = "Production stack in Spain", unit = "GWh",
            legend = FALSE, legendId = 1, height = "100%", width = "100%"),
  prodStack(mydata, areas = "be",
            main = "Production stack in Belgium", unit = "GWh",
            legend = FALSE, legendId = 1, height = "100%", width = "100%"),
  footer = prodStackLegend(legendId = 1)
)
```
```

`runAppAntaresViz` *Run app antaresViz*

Description

`runAppAntaresViz` run antaresViz App.

Usage

```
runAppAntaresViz()
```

Value

an App Shiny.

savePlotAsPng *Save interactive plot as a png image*

Description

This function saves an interactive plot generated with one of the functions of this package as a png image. The result can then be included in documents or presentations.

Usage

```
savePlotAsPng(plot, file = "Rplot.png", width = 600, height = 480, ...)
```

Arguments

plot	A plot generated with one of the functions of this package.
file	The name of the output file
width	Width of the output file
height	height of the output file
...	Other parameters passed to function [webshot::webshot]

Value

The function only creates the required file. Nothing is returned

Examples

```
## Not run:
  mydata <- readAntares()
  myplot <- plot(mydata, variable = "MRG. PRICE", type = "density")
  savePlotAsPng(myplot, file = "myplot.png")

## End(Not run)
```

setInteractivity *Get and set interactivity mode*

Description

`setInteractivity` globally sets the interactivity mode of plot functions. This is useful to avoid repeating `interactive = FALSE` or `interactive = TRUE` in each function. `getInteractivity` gets the interactivity mode.

Usage

```
setInteractivity(interactive = "auto")

getInteractivity()
```

Arguments

interactive	Should plot functions generate a UI that lets users to interactively modify input data and graphical parameters of a chart? It should be TRUE or FALSE. The default behavior is to set it to TRUE if the R session is interactive and to FALSE otherwise (for instance in Rmarkdown document).

Value

`getInteractivity` returns a boolean indicating the interactivity mode of plot functions. `setInteractivity` is only used for its side effects.

---

stackMap *plot stack and map*

Description

plot stack and map

Usage

```
stackMap(x, mapLayout)
```

Arguments

x	`antaresDataList` antaresDataList contian areas ans links.
mapLayout	Object created with function [mapLayout()]

Examples

```
## Not run:
mydata <- readAntares(areas = "all", links = "all")

layout <- readLayout()
ml <- mapLayout(layout = layout)

stackMap(x = mydata, mapLayout = ml)

## End(Not run)
```

tsPlot *plot time series contained in an antaresData object*

Description

This function generates an interactive plot of an antares time series.

Usage

```
tsPlot(
  x,
  refStudy = NULL,
  table = NULL,
  variable = NULL,
  elements = NULL,
  variable2Axe = NULL,
  mcYear = "average",
  type = c("ts", "barplot", "monotone", "density", "cdf", "heatmap"),
  dateRange = NULL,
  typeConfInt = FALSE,
  confInt = 0,
  minValue = NULL,
  maxValue = NULL,
  aggregate = c("none", "mean", "sum", "mean by variable", "sum by variable"),
  compare = NULL,
  compareOpts = list(),
  interactive = getInteractivity(),
  colors = NULL,
  main = NULL,
  ylab = NULL,
  legend = TRUE,
  legendItemsPerRow = 5,
  colorScaleOpts = colorScaleOptions(20),
  width = NULL,
  height = NULL,
  xyCompare = c("union", "intersect"),
  h5requestFiltering = deprecated(),
  highlight = FALSE,
  stepPlot = FALSE,
  drawPoints = FALSE,
  secondAxis = FALSE,
  timeSteph5 = deprecated(),
  mcYearh5 = deprecated(),
  tablesh5 = deprecated(),
  language = "en",
  hidden = NULL,
  ...
```

```
)

## S3 method for class 'antaresData'
plot(
  x,
  refStudy = NULL,
  table = NULL,
  variable = NULL,
  elements = NULL,
  variable2Axe = NULL,
  mcYear = "average",
  type = c("ts", "barplot", "monotone", "density", "cdf", "heatmap"),
  dateRange = NULL,
  typeConfInt = FALSE,
  confInt = 0,
  minValue = NULL,
  maxValue = NULL,
  aggregate = c("none", "mean", "sum", "mean by variable", "sum by variable"),
  compare = NULL,
  compareOpts = list(),
  interactive = getInteractivity(),
  colors = NULL,
  main = NULL,
  ylab = NULL,
  legend = TRUE,
  legendItemsPerRow = 5,
  colorScaleOpts = colorScaleOptions(20),
  width = NULL,
  height = NULL,
  xyCompare = c("union", "intersect"),
  h5requestFiltering = deprecated(),
  highlight = FALSE,
  stepPlot = FALSE,
  drawPoints = FALSE,
  secondAxis = FALSE,
  timeSteph5 = deprecated(),
  mcYearh5 = deprecated(),
  tablesh5 = deprecated(),
  language = "en",
  hidden = NULL,
  ...
)

## S3 method for class 'simOptions'
plot(
  x,
  refStudy = NULL,
  table = NULL,
```

```
  variable = NULL,
  elements = NULL,
  variable2Axe = NULL,
  mcYear = "average",
  type = c("ts", "barplot", "monotone", "density", "cdf", "heatmap"),
  dateRange = NULL,
  typeConfInt = FALSE,
  confInt = 0,
  minValue = NULL,
  maxValue = NULL,
  aggregate = c("none", "mean", "sum", "mean by variable", "sum by variable"),
  compare = NULL,
  compareOpts = list(),
  interactive = getInteractivity(),
  colors = NULL,
  main = NULL,
  ylab = NULL,
  legend = TRUE,
  legendItemsPerRow = 5,
  colorScaleOpts = colorScaleOptions(20),
  width = NULL,
  height = NULL,
  xyCompare = c("union", "intersect"),
  h5requestFiltering = deprecated(),
  highlight = FALSE,
  stepPlot = FALSE,
  drawPoints = FALSE,
  secondAxis = FALSE,
  timeSteph5 = deprecated(),
  mcYearh5 = deprecated(),
  tablesh5 = deprecated(),
  language = "en",
  hidden = NULL,
  ...
)

## S3 method for class 'list'
plot(
  x,
  refStudy = NULL,
  table = NULL,
  variable = NULL,
  elements = NULL,
  variable2Axe = NULL,
  mcYear = "average",
  type = c("ts", "barplot", "monotone", "density", "cdf", "heatmap"),
  dateRange = NULL,
  typeConfInt = FALSE,
```

```
  confInt = 0,
  minValue = NULL,
  maxValue = NULL,
  aggregate = c("none", "mean", "sum", "mean by variable", "sum by variable"),
  compare = NULL,
  compareOpts = list(),
  interactive = getInteractivity(),
  colors = NULL,
  main = NULL,
  ylab = NULL,
  legend = TRUE,
  legendItemsPerRow = 5,
  colorScaleOpts = colorScaleOptions(20),
  width = NULL,
  height = NULL,
  xyCompare = c("union", "intersect"),
  h5requestFiltering = deprecated(),
  highlight = FALSE,
  stepPlot = FALSE,
  drawPoints = FALSE,
  secondAxis = FALSE,
  timeSteph5 = deprecated(),
  mcYearh5 = deprecated(),
  tablesh5 = deprecated(),
  language = "en",
  hidden = NULL,
  ...
)
```

Arguments

x	Object of class antaresData. Alternatively, it can be a list of antaresData objects. In this case, one chart is created for each object. Can also be opts object from h5 file or list of opts object from h5 file.
refStudy	An object of class antaresData created with function [antaresRead::readAntares()] containing data for areas and or districts. Can also contains an opts who refer to a h5 file.
table	Name of the table to display when x is an antaresDataList object.
variable	Name of the variable to plot. If this argument is missing, then the function starts a shiny gadget that let the user choose the variable to represent. When the user clicks on the "Done" button", the graphic is returned by the function.
elements	Vector of "element" names indicating for which elements of 'x' should the variable be plotted. For instance if the input data contains areas, then this parameter should be a vector of area names. If data contains clusters data, this parameter has to be the concatenation of the area name and the cluster name, separated by " > ". This is to prevent confusion when two clusters from different areas have the same name.

variable2Axe — character, variables on second axis.

mcYear — If x, contains multiple Monte-Carlo scenarios, this parameter determine which scenario is displayed. Must be an integer representing the index of the scenario or the word "average". In this case data are averaged.

type — Type of plot to draw. "ts" creates a time series plot, "barplot" creates a barplot with one bar per element representing the average value of the variable for this element. "monotone" draws the monotone curve of the variable for each element.

dateRange — A vector of two dates. Only data points between these two dates are displayed. If NULL, then all data is displayed.

typeConfInt — logical. If multiple Monte Carlo scenarios are present in the input data, see all curves (FALSE, Default), or mean and confidence interval (TRUE)

confInt — Number between 0 and 1 indicating the size of the confidence interval to display. If it equals to 0, then confidence interval is not computed nor displayed. Used only when multiple Monte Carlo scenarios are present in the input data.

minValue — Only used if parameter type is "density" or "cdf". If this parameter is set, all values that are less than minValue are removed from the graphic. This is useful to deal with variables containing a few extreme values (generally cost and price variables). If minValue is unset, all values are displayed.

maxValue — Only used if parameter type is "density" or "cdf". If this parameter is set, all values not in [-minValue, maxValue] are removed from the graphic. This is useful to deal with variables containing a few extreme values (generally cost and price variables). If maxValue is 0 or unset, all values are displayed.

aggregate — When multiple elements are selected, should the data be aggregated. If "none", each element is represented separetly. If "mean" values are averaged and if "sum" they are added. You can also compute mean and sum by variable.

compare — An optional character vector containing names of parameters. When it is set, two charts are outputed with their own input controls. Alternatively, it can be a named list with names corresponding to parameter names and values being list with the initial values of the given parameter for each chart. See details if you are drawing a map.

compareOpts — List of options that indicates the number of charts to create and their position. Check out the documentation of [manipulateWidget::compareOptions] to see available options.

interactive — LogicalValue. If TRUE, then a shiny gadget is launched that lets the user interactively choose the areas or districts to display.

colors — Vector of colors

main — Title of the graph.

ylab — Label of the Y axis.

legend — Logical value indicating if a legend should be drawn. This argument is usefull when one wants to create a shared legend with [prodStackLegend()]

legendItemsPerRow — Number of elements to put in each row of the legend.

colorScaleOpts	A list of parameters that control the creation of color scales. It is used only for heatmaps. See [colorScaleOptions()] for available parameters.
width	Width of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.
height	Height of the graph expressed in pixels or in percentage of the parent element. For instance "500px" and "100%" are valid values.
xyCompare	Use when you compare studies, can be "union" or "intersect". If union, all of mcYears in one of studies will be selectable. If intersect, only mcYears in all studies will be selectable.
h5requestFiltering	Contains arguments used by default for h5 request, typically h5requestFiltering = list(mcYears = 2)
highlight	highlight curve when mouse over
stepPlot	boolean, step style for curves.
drawPoints	boolean, add points on graph
secondAxis	add second axis to graph
timeSteph5	character timeStep to read in h5 file. Only for Non interactive mode.
mcYearh5	numeric mcYear to read for h5. Only for Non interactive mode.
tablesh5	character tables for h5 ("areas" "links", "clusters" or "disticts"). Only for Non interactive mode.
language	character language use for label. Defaut to 'en'. Can be 'fr'.
hidden	logical Names of input to hide. Defaut to NULL
...	Other arguments for [manipulateWidget::manipulateWidget]

Details

If the input data contains several Monte-Carlo scenarios, the function will display the evolution of the average value. Moreover it will represent a 95

If the input data has a annual time step, the function creates a barplot instead of a line chart.

compare argument can take following values :

- "mcYear"
- "main"
- "variable"
- "type"
- "typeConfInt"
- "confInt"
- "elements"
- "aggregate"
- "legend"
- "highlight"
- "stepPlot"
- "drawPoints"
- "secondAxis"

Value

The function returns an object of class "htmlwidget". It is generated by package highcharter if time step is annual or by dygraphs for any other time step.It can be directly displayed in the viewer or be stored in a variable for later use.

Examples

```
## Not run:
mydata <- readAntares(areas = "all", timeStep = "hourly")
plot(x = mydata)

# Plot only a few areas
plot(x = mydata[area %in% c("area1", "area2", "area3")])

# If data contains detailed results, then the function adds a confidence
# interval
dataDetailed <- readAntares(areas = "all", timeStep = "hourly", mcYears = 1:2)
plot(x = dataDetailed)

# If the time step is annual, the function creates a barplot instead of a
# linechart
dataAnnual <- readAntares(areas = "all", timeStep = "annual")
plot(x = dataAnnual)

# Compare two simulaitons
# Compare the results of two simulations
setSimulationPath(path1)
mydata1 <- readAntares(areas = "all", timeStep = "daily")
setSimulationPath(path2)
mydata2 <- readAntares(areas = "all", timeStep = "daily")

plot(x = list(mydata1, mydata2))

# When you compare studies, you have 2 ways to defind inputs, union or intersect.
# for example, if you chose union and you have mcYears 1 and 2 in the first study
# and mcYears 2 and 3 in the second, mcYear input will be worth c(1, 2, 3)
# In same initial condition (study 1 -> 1,2 ans study 2 -> 2, 3) if you choose intersect,
# mcYear input will be wort 2.
# You must specify union or intersect with xyCompare argument (default union).
plot(x = list(mydata1[area %in% c("a", "b")],
 mydata1[area %in% c("b", "c")]), xyCompare = "union")
plot(x = list(mydata1[area %in% c("a", "b")],
 mydata1[area %in% c("b", "c")]), xyCompare = "intersect")

# Compare data in a single simulation
# Compare two periods for the same simulation
plot(x = mydata1, compare = "dateRange")

# Compare two Monte-Carlo scenarios
detailedData <- readAntares(areas = "all", mcYears = "all")
plot(x = detailedData, .compare = "mcYear")
```

```
# Use h5 for dynamic request / exploration in a study
# Set path of simulaiton
setSimulationPath(path = path1)

# Convert your study in h5 format
writeAntaresH5(path = mynewpath)

# Redefine sim path with h5 file
opts <- setSimulationPath(path = mynewpath)
plot(x = opts)

# Compare elements in a single study
plot(x = opts, .compare = "mcYear")
# Compare 2 studies
plot(x = list(opts, opts2))

# Compare 2 studies with argument refStudy
plot(x = opts, refStudy = opts2)
plot(x = opts, refStudy = opts2, type = "ts", interactive = FALSE, mcYearh5 = 2)
plot(x = opts, refStudy = opts2, type = "ts",
    dateRange = DR,
    h5requestFiltering = list(mcYears = mcYears = mcYearToTest))



## End(Not run)
```

Index