Package 'antaresRead'

October 12, 2022

Type Package

Title Import, Manipulate and Explore the Results of an 'Antares' Simulation

Version 2.3.0

Description Import, manipulate and explore results generated by 'Antares', a powerful open source software developed by RTE (Réseau de Transport d'Électricité) to simulate and study electric power systems
(more information about 'Antares' here : <https://antares-simulator.org/>).

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

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

License GPL (>= 2) | file LICENSE

Imports data.table (>= 1.9.6), bit64, lubridate (>= 1.7.1), plyr, methods, stats, stringr, shiny, pbapply, doParallel, jsonlite, httr, utils

Suggests rhdf5 (>= 2.24.0), testthat, covr, knitr, rmarkdown, foreach, parallel, htmltools

RoxygenNote 7.2.1

VignetteBuilder knitr

Encoding UTF-8

biocViews Infrastructure, DataImport

NeedsCompilation no

Author Veronique Bachelier [aut, cre], Jalal-Edine ZAWAM [aut], Frederic Breant [ctb], Francois Guillem [aut], Benoit Thieurmel [aut], Titouan Robert [aut], Victor Perrier [ctb], Etienne Sanchez [ctb], RTE [cph]

Maintainer Veronique Bachelier <veronique.bachelier@rte-france.com>

Repository CRAN

Date/Publication 2022-10-10 08:10:02 UTC

R topics documented:

.writeIni *Write ini file from list obtain by antaresRead:::readIniFile and modify by user*

Description

Write ini file from list obtain by antaresRead:::readIniFile and modify by user

Usage

```
.writeIni(listData, pathIni, overwrite = FALSE)
```

Arguments

listData	list, modified list obtained by antaresRead:::readIniFile.
pathIni	Character, Path to ini file.
overwrite	logical, should file be overwritten if already exist?

Examples

```
## Not run:
pathIni <- "D:/exemple_test/settings/generaldata.ini"
generalSetting <- antaresRead:::readIniFile(pathIni)
generalSetting$output$synthesis <- FALSE
writeIni(generalSetting, pathIni)

## End(Not run)
```

API-methods *API methods*

Description

API methods

Usage

```
api_get(opts, endpoint, ..., default_endpoint = "v1/studies")

api_post(opts, endpoint, ..., default_endpoint = "v1/studies")

api_put(opts, endpoint, ..., default_endpoint = "v1/studies")

api_delete(opts, endpoint, ..., default_endpoint = "v1/studies")
```

Arguments

opts	Antares simulation options or a ‘list‘ with an ‘host = ‘ slot.
endpoint	API endpoint to interrogate, it will be added after ‘default_endpoint‘. Can be a full URL (by wrapping ìn [I()]), in that case ‘default_endpoint‘ is ignored.
...	Additional arguments passed to API method.
default_endpoint	Default endpoint to use.

Value

Response from the API.

Examples

```
## Not run:

# List studies with local API
api_get(
  opts = list(host = "http://0.0.0.0:8080"),
  endpoint = NULL
)


## End(Not run)
```

---

as.antaresDataList *Convert objects to antaresDataTable*

Description

This function converts a list of tables or table into an `antaresDataList` object.

An `antaresDataList` is a list of tables of class`antaresDataTable`. It also has attributes that store the time step, the type of data and the simulation options.

Usage

```
as.antaresDataList(x, ...)

## S3 method for class 'antaresDataTable'
as.antaresDataList(x, name = NULL, ...)

## S3 method for class 'data.frame'
as.antaresDataList(
  x,
  synthesis,
  timeStep,
```

```
  type,
  opts = simOptions(),
  name = type,
  ...
)
```

Arguments

x	Data.frame or data.table to convert to a an antaresDataTable.
...	Arguments to be passed to methods.
name	name of the table in the final object. If NULL, the type of the data is used.
synthesis	Does the table contain synthetic results ?
timeStep	Time step of the data. One of "hourly", "daily", "weekly", "monthly" or "annual".
type	type of data: for instance "areas", "links", "clusters", etc.
opts	Simulation options.

Value

antaresDataList object.

---

as.antaresDataTable *Convert objects to antaresDataTable*

Description

This function converts a data.frame or a data.table into an antaresDataTable object.

An antaresDataTable is simply a data.table with additional attributes recording the time step, the type of data and the simulation options.

Usage

```
as.antaresDataTable(x, ...)

## S3 method for class 'data.frame'
as.antaresDataTable(x, synthesis, timeStep, type, opts = simOptions(), ...)
```

Arguments

x	object to convert to a an antaresDataList.
...	Arguments to be passed to methods.
synthesis	Does the table contain synthetic results ?
timeStep	Time step of the data. One of "hourly", "daily", "weekly", "monthly" or "annual".
type	type of data: for instance "areas", "links", "clusters", etc.
opts	Simulation options.

Value

antaresDataTable object.

---

changeTimeStep *Change the timestep of an output*

Description

This function changes the timestep of a table or an antaresData object and performs the required aggregation or desaggregation. We can specify (des)aggregate functions by columns, see the param 'fun'.

Usage

```
changeTimeStep(x, newTimeStep, oldTimeStep, fun = "sum", opts = simOptions())
```

Arguments

x	data.table with a column "timeId" or an object of class "antaresDataList"
newTimeStep	Desired time step.The possible values are hourly, daily, weekly, monthly and annual.
oldTimeStep	Current time step of the data. This argument is optional for an object of class antaresData because the time step of the data is stored inside the object
fun	Character vector with one element per column to (des)aggregate indicating the function to use ("sum", "mean", "min" or "max") for this column. It can be a single element, in that case the same function is applied to every columns.
opts	list of simulation parameters returned by the function setSimulationPath

Value

Either a data.table or an object of class "antaresDataList" depending on the class of x

Examples

```
## Not run:
setSimulationPath()

areasH <- readAntares(select = "LOAD", synthesis = FALSE, mcYears = 1)
areasD <- readAntares(select = "LOAD", synthesis = FALSE, mcYears = 1, timeStep ="daily")

areasDAgg <- changeTimeStep(areasH, "daily", "hourly")

all.equal(areasDAgg$LOAD, areasD$LOAD)

# Use different aggregation functions
mydata <- readAntares(select = c("LOAD", "MRG. PRICE"), timeStep = "monthly")
changeTimeStep(mydata, "annual", fun = c("sum", "mean"))
```

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

`copyToClipboard` *Copy data to the clipboard*

Description

`copyToClipboard` is a utility function that copies data to the clipboard. The data can then be copied in another program like excel.

Usage

```
copyToClipboard(x, ...)

## S3 method for class 'antaresDataList'
copyToClipboard(x, what, ...)
```

Arguments

`x`	an object used to select a method.
`...`	arguments passed to `write.table`
`what`	character or numeric indicating which element to copy to clipboard (areas, links, clusters or districts)

Value

The function does not return anything. It is only used to interact with the clipboard

Note

The function is useful only for small data objects: for a table, only the 50000 rows are copied to clipboard. If the table to copy is longer, either use filters to reduce the number of rows or write the table in text file with `write.table`

Examples

```
 # This only works on Windows systems
## Not run:
x <- data.frame(a = sample(10), b = sample(10))

copyToClipboard(x)

# Try to open excel and use CTRL + V to copy the data in a spreadsheet.

## End(Not run)
```

extractDataList *Format data PPSE-style*

Description

This function converts an "readAntares" object in the data structure used by PPSE : instead of having one table for areas, one for links and one for clusters, the function creates a list with one element per area. Each element is a data.table containing the data about the area and one column per cluster of the area containing the production of this cluster.

Usage

```
extractDataList(x, areas = NULL)
```

Arguments

x	object of class "antaresData" or "antaresTable" created by the function readAntares
areas	character vector containing the name of areas to keep in the final object. If NULL, all areas are kept in the final object.

Value

a list of data.tables with one element per area. The list also contains an element named "areaList" containing the name of areas in the object and a table called "infos" that contains for each area the number of variables of different type (values, details, link).

getAreas *Select and exclude areas*

Description

getAreas and getDistricts are utility functions that builds list of areas or districts by using regular expressions to select and/or exclude areas/districts

Usage

```
getAreas(
  select = NULL,
  exclude = NULL,
  withClustersOnly = FALSE,
  regexpSelect = TRUE,
  regexpExclude = TRUE,
  opts = simOptions(),
  ignore.case = TRUE,
  districts = NULL
```

```
)

getDistricts(
  select = NULL,
  exclude = NULL,
  regexpSelect = TRUE,
  regexpExclude = TRUE,
  opts = simOptions(),
  ignore.case = TRUE
)
```

Arguments

select	Character vector. If regexpSelect is TRUE, this vector is interpreted as a list of regular expressions. Else it is interpreted as a list of area names. If NULL, all areas are selected
exclude	Character vector. If regexpExclude is TRUE, this vector is interpreted as a list of regular expressions and each area validating one of them is excluded. Else it is interpreted as list of area names to exclude. If NULL, not any area is excluded.
withClustersOnly	Should the function return only nodes containing clusters ?
regexpSelect	Is select a list of regular expressions ?
regexpExclude	Is exclude a list of regular expressions ?
opts	list of simulation parameters returned by the function setSimulationPath
ignore.case	Should the case be ignored when evaluating the regular expressions ?
districts	Names of districts. If this argument is not null, only areas belonging to the specified districts are returned.

Value

A character vector containing the name of the areas/districts satisfying the rules defined by the parameters.

See Also

getLinks

---

getIdCols *get Id columns*

Description

getIdCols return the id columns of an AntaresDataTable

Usage

```
getIdCols(x = NULL)
```

Arguments

x	an AntaresDataTable.

Value

A character vector containing the name of the id columns of an antaresDataTable

---

getLinks *Retrieve links connected to a set of areas*

Description

This function finds the names of the links connected to a set of areas.

Usage

```
getLinks(
  areas = NULL,
  exclude = NULL,
  opts = simOptions(),
  internalOnly = FALSE,
  namesOnly = TRUE,
  withDirection = FALSE
)
```

Arguments

areas	Vector containing area names. It represents the set of areas we are interested in. If NULL, all areas of the study are used.
exclude	Vector containing area names. If not NULL, all links connected to one of these areas are omitted.
opts	list of simulation parameters returned by the function setSimulationPath
internalOnly	If TRUE, only links that connect two areas from parameter areas are returned. If not, the function also returns all the links that connect an area from the list with an area outside the list.
namesOnly	If TRUE, the function returns a vector with link names, else it returns a table containing the name, the origin and the destination of each selected link.
withDirection	Used only if namesOnly = FALSE. If FALSE, then the function returns a table with one line per link, containing the link name, the origin and the destination of the link. If TRUE, then it returns a table with columns area, link, to and direction which is equal is equal to 1 if the link connects area to to and -1 if it connects to to area. The column area contains only areas that are compatible with parameters areas and exclude. Note that the same link can appear twice in the table with different directions.

Value

If `namesOnly = TRUE` the function returns a vector containing link names

If `namesOnly = FALSE` and `withDirection = FALSE`, it returns a `data.table` with **exactly one line per link** and with three columns:

`link`	Link name
`from`	First area connected to the link
`to`	Second area connected to the link

If `namesOnly = FALSE` and `withDirection = TRUE`, it returns a `data.table` with **one or two lines per link** and with four columns:

`area`	Area name
`link`	Link name
`to`	Area connected to `area` by `link`
`direction`	1 if the link connects `area` to `to` else -1

Examples

```
## Not run:

# Get all links of a study
getLinks()

# Get all links with their origins and destinations
getLinks(namesOnly = FALSE)

# Get all links connected to French areas (assuming their names contain "fr")
getLinks(getAreas("fr"))

# Same but with only links connecting two French areas
getLinks(getAreas("fr"), internalOnly = TRUE)

# Exclude links connecting real areas with pumped storage virtual areas
# (assuming their names contain "psp")
getLinks(getAreas("fr"), exclude = getAreas("psp"))


## End(Not run)
```

`hvdcModification` *hvdc straitement*

Description

usage for hvdc

Usage

```
hvdcModification(data, removeHvdcAreas = TRUE, reafectLinks = FALSE)
```

Arguments

data	antaresDataList data to apply straitement
removeHvdcAreas	boolean remove HVDC areas.
reafectLinks	boolean .

Examples

```
## Not run:

data <- readAntares(areas = 'all', links = 'all')
data <- setHvdcAreas(data, "psp in")
data <- hvdcModification(data)


## End(Not run)
```

---

isH5Opts *Test if opts is h5*

Description

Test if the value returned by setSimulationPath() is referring to an h5 file

Usage

```
isH5Opts(opts)
```

Arguments

opts	, opts

parAggregateMCall *Creation of Mc_all (only antares > V6)*

Description

Creation of Mc_all (only antares > V6)

Usage

```
parAggregateMCall(
  opts,
  nbcl = 8,
  verbose = 1,
  timestep = c("annual", "daily", "hourly", "monthly", "weekly"),
  writeOutput = TRUE,
  mcWeights = NULL,
  mcYears = NULL
)

aggregateResult(
  opts,
  verbose = 1,
  filtering = FALSE,
  selected = NULL,
  timestep = c("annual", "daily", "hourly", "monthly", "weekly"),
  writeOutput = FALSE,
  mcWeights = NULL,
  mcYears = NULL
)
```

Arguments

opts	list of simulation parameters returned by the function setSimulationPath
nbcl	numeric Number of parralel process
verbose	numeric show log in console. Defaut to 1 • 0 : No log • 1 : Short log • 2 : Long log
timestep	character antares timestep
writeOutput	boolean write result or not.
mcWeights	numeric vector of weigth for mcYears.
mcYears	numeric mcYears to load.
filtering	boolean filtering control
selected	list named list (pass to antaresRead) : list(areas = 'a', links = 'a - e')

Examples

```
## Not run:
   aggregateResult(opts)


## End(Not run)
```

---

ponderateMcAggregation

Mcyear aggregation weigthed by wd

Description

Mcyear aggregation weigthed by wd

Usage

```
ponderateMcAggregation(x, fun = weighted.mean, ...)
```

Arguments

x	antaresData data import with antaresRead
fun	function function to use
...	args others args pass to fun

Examples

```
## Not run:
  data <- readAntares(areas = 'all', mcYears = 'all')
  ponderateMcAggregation(data, fun = weighted.mean, w = c(.1, .9))


## End(Not run)
```

read-ini *Read configuration options from file or API*

Description

Read configuration options from file or API

Usage

```
readIni(pathIni, opts = antaresRead::simOptions(), default_ext = ".ini")

readIniFile(file, stringsAsFactors = FALSE)

readIniAPI(study_id, path, host, token = NULL)
```

Arguments

pathIni	Path to config/ini file to read.
opts	List of simulation parameters returned by the function [antaresRead::setSimulationPath()]
default_ext	Default extension used for config files.
file	File path.
stringsAsFactors	logical: should character vectors be converted to factors?
study_id	Study's identifier.
path	Path of configuration object to read.
host	Host of AntaREST server API.
token	API personnal access token.

Value

A list with an element for each section of the .ini file.

Examples

```
## Not run:
library(antaresRead)
library(antaresEditObject)

# With physical study:
setSimulationPath("../tests-studies/Study_V8.2/", simulation = "input")
readIni("settings/generaldata")

# With API
setSimulationPathAPI(
  host = "http://localhost:8080",
  study_id = "73427ae1-be83-44e0-b04f-d5127e53424c",
```

```
  token = NULL,
  simulation = "input"
)
readIni("settings/generaldata")


## End(Not run)
```

---

readAntares *Read the data of an Antares simulation*

Description

readAntares is a swiss-army-knife function used to read almost every possible time series of an antares Project at any desired time resolution (hourly, daily, weekly, monthly or annual).

It was first designed to read output time series, but it can also read input time series. The input time series are processed by the function to fit the query of the user (timeStep, synthetic results or Monte-Carlo simulation, etc.). The few data that are not read by readAntares can generally by read with other functions of the package starting with "read" (readClusterDesc, readLayout, readBindingConstraints)

Usage

```
readAntares(
  areas = NULL,
  links = NULL,
  clusters = NULL,
  districts = NULL,
  clustersRes = NULL,
  misc = FALSE,
  thermalAvailabilities = FALSE,
  hydroStorage = FALSE,
  hydroStorageMaxPower = FALSE,
  reserve = FALSE,
  linkCapacity = FALSE,
  mustRun = FALSE,
  thermalModulation = FALSE,
  select = NULL,
  mcYears = NULL,
  timeStep = c("hourly", "daily", "weekly", "monthly", "annual"),
  mcWeights = NULL,
  opts = simOptions(),
  parallel = FALSE,
  simplify = TRUE,
  showProgress = TRUE
)
```

Arguments

areas	Vector containing the names of the areas to import. If NULL no area is imported. The special value "all" tells the function to import all areas. By default, the value is "all" when no other argument is enter and "NULL" when other arguments are enter.
links	Vector containing the name of links to import. If NULL no area is imported. The special value "all" tells the function to import all areas. Use function getLinks to import all links connected to some areas.
clusters	Vector containing the name of the areas for which you want to import results at thermal cluster level. If NULL no cluster is imported. The special value "all" tells the function to import thermal clusters from all areas.
districts	Vector containing the names of the districts to import. If NULL, no district is imported. The special value "all" tells the function to import all districts.
clustersRes	Vector containing the name of the areas for which you want to import results at renewable cluster level. If NULL no cluster is imported. The special value "all" tells the function to import renewable clusters from all areas.
misc	Vector containing the name of the areas for which you want to import misc.
thermalAvailabilities	Should thermal availabilities of clusters be imported ? If TRUE, the column "thermalAvailability" is added to the result and a new column "availableUnits" containing the number of available units in a cluster is created.If synthesis is set to TRUE then "availableUnits" contain the mean of avaible units on all MC Years.
hydroStorage	Should hydro storage be imported ?
hydroStorageMaxPower	Should hydro storage maximum power be imported ?
reserve	Should reserve be imported ?
linkCapacity	Should link capacities be imported ?
mustRun	Should must run productions be added to the result? If TRUE, then four columns are added: mustRun contains the production of clusters that are in complete must run mode; mustRunPartial contains the partial must run production of clusters; mustRunTotal is the sum of the two previous columns. Finally thermalPmin is similar to mustRunTotal except it also takes into account the production induced by the minimum stable power of the units of a cluster. More precisely, for a given cluster and a given time step, it is equal to min(NODU x min.stable.power, mustRunTotal).
thermalModulation	Should thermal modulation time series be imported ? If TRUE, the columns "marginalCostModulation", "marketBidModulation", "capacityModulation" and "minGenModulation" are added to the cluster data.
select	Character vector containing the name of the columns to import. If this argument is NULL, all variables are imported. Special names "allAreas" and "allLinks" indicate to the function to import all variables for areas or for links. Since version 1.0, values "misc", "thermalAvailabilities", "hydroStorage", "hydroStorageMaxPower", "reserve", "linkCapacity", "mustRun", "thermalModulation"

are also accepted and can replace the corresponding arguments. The list of available variables can be seen with the command `simOptions()$variables`. Id variables like `area`, `link` or `timeId` are automatically imported. Note that `select` is *not* taken into account when importing cluster data.

`mcYears` Index of the Monte-Carlo years to import. If `NULL`, synthetic results are read, else the specified Monte-Carlo simulations are imported. The special value `all` tells the function to import all Monte-Carlo simulations.

`timeStep` Resolution of the data to import: hourly (default), daily, weekly, monthly or annual.

`mcWeights` Vector of weights to apply to the specified mcYears. If not `NULL`, the vector must be the same length as the vector provided in the `mcYear` parameter. The function `readAntares` will then return the weighted synthetic results for the specified years, with the specified weights.

`opts` list of simulation parameters returned by the function `setSimulationPath`

`parallel` Should the importation be parallelized ? (See details)

`simplify` If TRUE and only one type of output is imported then a data.table is returned. If FALSE, the result will always be a list of class "antaresData".

`showProgress` If TRUE the function displays information about the progress of the importation.

Details

If parameters `areas`, `links`, `clusters` and `districts` are all `NULL`, `readAntares` will read output for all areas. By default the function reads synthetic results if they are available.

`readAntares` is able to read input time series, but when they are not stored in output, these time series may have changed since a simulation has been run. In such a case the function will remind you this danger with a warning.

When individual Monte-Carlo simulations are read, the function may crash because of insufficient memory. In such a case, it is necessary to reduce size of the output. Different strategies are available depending on your objective:

- Use a larger time step (parameter `timeStep`)
- Filter the elements to import (parameters `areas`,`links`, `clusters` and `districts`)
- Select only a few columns (parameter `select`)
- read only a subset of Monte-Carlo simulations (parameter `mcYears`). For instance one can import a random sample of 100 simulations with `mcYears = sample(simOptions()$mcYears, 100)`

Value

If `simplify = TRUE` and only one type of output is imported then the result is a data.table.

Else an object of class "antaresDataList" is returned. It is a list of data.tables, each element representing one type of element (areas, links, clusters)

Parallelization

If you import several elements of the same type (areas, links, clusters), you can use parallelized importation to improve performance. Setting the parameter `parallel = TRUE` is not enough to parallelize the importation, you also have to install the package foreach and a package that provides a parallel backend (for instance the package doParallel).

Before running the function with argument `parallel=TRUE`, you need to register your parallel backend. For instance, if you use package "doParallel" you need to use the function `registerDoParallel` once per session.

See Also

setSimulationPath, getAreas, getLinks, getDistricts

Examples

```
## Not run:
# Import areas and links separately

areas <- readAntares() # equivalent to readAntares(areas="all")
links <- readAntares(links="all")

# Import areas and links at same time

output <- readAntares(areas = "all", links = "all")

# Add input time series to the object returned by the function
areas <- readAntares(areas = "all", misc = TRUE, reserve = TRUE)

# Get all output for one area

myArea <- sample(simOptions()$areaList, 1)
myArea

myAreaOutput <- readAntares(area = myArea,
                            links = getLinks(myArea, regexpSelect=FALSE),
                            clusters = myArea)

# Or equivalently:
myAreaOutput <- readAntaresAreas(myArea)

# Use parameter "select" to read only some columns.

areas <- readAntares(select = c("LOAD", "OV. COST"))

# Aliases can be used to select frequent groups of columns. use showAliases()
# to view a list of available aliases

areas <- readAntares(select="economy")


## End(Not run)
```

`readAntaresAreas` *Read output for a list of areas*

Description

This a function is a wrapper for "antaresData" that reads all data for a list of areas.

Usage

```
readAntaresAreas(
  areas,
  links = TRUE,
  clusters = TRUE,
  clustersRes = TRUE,
  internalOnly = FALSE,
  opts = simOptions(),
  ...
)
```

Arguments

`areas`	Vector containing area names. It represents the set of areas we are interested in. If `NULL`, all areas of the study are used.
`links`	should links connected to the areas be imported ?
`clusters`	should the thermal clusters of the areas be imported ?
`clustersRes`	should the renewable clusters of the areas be imported ?
`internalOnly`	If `TRUE`, only links that connect two areas from parameter `areas` are returned. If not, the function also returns all the links that connect an area from the list with an area outside the list.
`opts`	list of simulation parameters returned by the function `setSimulationPath`
`...`	Other arguments passed to the function `readAntares`

Value

If `simplify = TRUE` and only one type of output is imported then the result is a data.table.

Else an object of class "antaresData" is returned. It is a list of data.tables, each element representing one type of element (areas, links, clusters)

Examples

```
## Not run:
myarea <- simOptions()$areaList[1]
data <- readAntaresAreas(myarea)

# Equivalent but more concise than:
data2 <- readAntares(myarea, links = getLinks(myarea), clusters = myarea)
```

```
all.equal(data, data2)

## End(Not run)
```

`readBindingConstraints` *Read binding constraints*

Description

This function reads the binding constraints of an Antares project.

Be aware that binding constraints are read in the input files of a study. So they may have changed since a simulation has been run.

Usage

```
readBindingConstraints(opts = simOptions())

## S3 method for class 'bindingConstraints'
summary(object, ...)
```

Arguments

`opts`	list of simulation parameters returned by the function `setSimulationPath`
`object`	Object returned by readBindingConstraints
`...`	Unused

Value

`readBindingConstraints` returns an object of class `bindingConstraints`. It is a named list with one element per read constraint. Each element is itself a list with the following elements:

`enabled`	is the constraint enabled ?
`timeStep`	time step the constraint applies to
`operator`	type of constraint: equality, inequality on one side or both sides
`coefficients`	elements containing the coefficients used by the constraint
`values`	values used by the constraint. It contains one line per time step and three columns "less", "greater" and "equal"

The `summary` method returns a data.frame with one line per constraint.

Examples

```
## Not run:
setSimulationPath()

constraints <- readBindingConstraints()
summary(constraints)


## End(Not run)
```

`readClusterDesc` *Import clusters description*

Description

This function reads in the input files of an antares study the characteristics of each cluster.

Be aware that clusters descriptions are read in the input files so they may have changed since a simulation has been run.

Usage

```
readClusterDesc(opts = simOptions())

readClusterResDesc(opts = simOptions())
```

Arguments

`opts`	list of simulation parameters returned by the function `setSimulationPath`

Value

A data.table with one line per cluster. The columns of the data.table may change between different projects, but there will always be the following columns:

`area`	Name of the area containing the cluster
`cluster`	Name of the cluster
`group`	Type of cluster (gaz, nuclear, etc.)
`unitcount`	number of production units
`nominalcapacity`	production capacity of each unit

The other present columns depends on the version of antares and the options that have been set: if an option is unset for all clusters, it will not appear in the table.

By default, the function reads the cluster description of the default antares study. You can use the argument `opts` to specify another study.

`readClusterDesc` : read thermal clusters

`readClusterResDesc` : read renewable clusters (Antares >= V8.1)

Examples

```
## Not run:

# thermal
readClusterDesc()

# renewable
readClusterResDesc()

# By default, the function reads cluster descriptions for the default study,
# but it is possible to specify another study with parameter "opts"
sim1 <- setSimulationPath()

#[... code that modifies the default antares study]

readClusterDesc(sim1)


## End(Not run)
```

`readInputTS` *Read Input time series*

Description

`readInputTS` is a function that reads time series from an antares project. But contrary to `readAntares`, it only reads time series stored in the input folder, so it can work in "input" mode.

Usage

```
readInputTS(
  load = NULL,
  thermalAvailabilities = NULL,
  ror = NULL,
  hydroStorage = NULL,
  hydroStorageMaxPower = NULL,
  wind = NULL,
  solar = NULL,
  misc = NULL,
  reserve = NULL,
  linkCapacity = NULL,
  resProduction = NULL,
  opts = simOptions(),
  timeStep = c("hourly", "daily", "weekly", "monthly", "annual"),
  simplify = TRUE,
  parallel = FALSE,
```

```
  showProgress = TRUE
)
```

Arguments

load	vector of areas names for which load time series must be read.
thermalAvailabilities	vector of areas names for which thermal availabilities of clusters must be read.
ror	vector of areas names for which run of river time series must be read.
hydroStorage	vector of areas names for which hydrolic storage time series must be read.
hydroStorageMaxPower	vector of areas names for which hydrolic storage maximum power time series must be read.
wind	vector of areas names for which wind time series must be read
solar	vector of areas names for which solar time series must be read
misc	vector of areas names for which misc time series must be read
reserve	vector of areas names for which reserve time series must be read
linkCapacity	vector of links names for which links characteristics time series must be read
resProduction	vector of areas names for which renewables clusters production time series must be read.
opts	list of simulation parameters returned by the function setSimulationPath
timeStep	Resolution of the data to import: hourly (default), daily, weekly, monthly or annual.
simplify	If TRUE and only one type of output is imported then a data.table is returned. If FALSE, the result will always be a list of class "antaresData".
parallel	Should the importation be parallelized ? (See details)
showProgress	If TRUE the function displays information about the progress of the importation.

Value

If simplify = TRUE and only one type of input is imported then the result is a data.table with class "antaresDataTable".

Else an object of class "antaresDataList" is returned. It is a list of data.tables, each element representing one type of element (load, wind, solar, etc.).

Note

All parameters expecting a vector of areas or links names also accept the special value "all". It indicates the function to read the desired time series for all areas or links.

See Also

setSimulationPath, readAntares, getAreas, getLinks

Examples

```
## Not run:
# Set an antares study in "input" mode. This is useful when one want to
# inspect input time series before running a simulation.
# Note that readAntares do not function in input mode, but readInputTS
# works with any mode.

setSimulationPath("path_to_the_study", "input")

# Read load time series
readInputTS(load = "all")

# Read hydrolic storage and maximum power in the same call:
readInputTS(hydroStorage = "all", hydroStorageMaxPower = "all")

# Use a different time step
myArea <- readInputTS(load= "myArea", timeStep = "monthly")

# Quick plot to visualize the variability of the series
matplot(myArea[, - (1:2), with = FALSE], type = "l")

## End(Not run)
```

---

readLayout *Read areas layout*

Description

This function reads in the input files of an antares study the current areas layout, ie. the position of the areas It may be useful for plotting the network.

Be aware that the layout is read in the input files so they may have changed since a simulation has been run.

Usage

```
readLayout(opts = simOptions(), xyCompare = c("union", "intersect"))
```

Arguments

opts	list of simulation parameters returned by the function setSimulationPath
xyCompare	Use when passing multiple opts, can be "union" or "intersect".

Value

A list with three elements:

areas:	A data.frame containing the name, the color and the coordinate of each area

`district:` A data.frame containing the name, the color and the coordinate of each district

`links:` A data.frame containing the name, the coordinates of the origin and the destination of each link

By default, `readLayout` reads the layout for the current default antares study. It is possible to specify another study with the parameter `opts`. And we can pass multiple studies using a `list` of opts.

Examples

```
## Not run:
readLayout()

# By default, the function reads layout for the default study,
# but it is possible to specify another study with parameter "opts"
sim1 <- setSimulationPath()

#[... code that modifies the default antares study]

readLayout(sim1)


## End(Not run)
```

---

readOptimCriteria *Read Optimization Criteria*

Description

This function can be used to read the value of the criteria optimized by ANTARES. Notice that these values are only available in "Xpansion" mode or when option "Export mps" is turned on.

Usage

```
readOptimCriteria(opts = simOptions())
```

Arguments

`opts` list of simulation parameters returned by the function `setSimulationPath`

Value

A table of class `antaresDataTable`. It contains the usual columns `timeID`, `mcYear`, `time` and two columns "criterion1" and "criterion2" containing the values of the criteria. Time step can be daily or weekly depending on the optimization options.

Examples

```
## Not run:
setSimulationPath()

optimCriteria <- readOptimCriteria()

## End(Not run)
```

`removeVirtualAreas` *Remove virtual areas*

Description

This function removes virtual areas from an `antaresDataList` object and corrects the data for the real areas. The `antaresDataList` object should contain area and link data to function correctly.

Usage

```
removeVirtualAreas(
  x,
  storageFlexibility = NULL,
  production = NULL,
  reassignCosts = FALSE,
  newCols = TRUE,
  rowBal = TRUE,
  prodVars = getAlias("rmVA_production"),
  costsVars = c("OV. COST", "OP. COST", "CO2 EMIS.", "NP COST"),
  costsOn = c("both", "storageFlexibility", "production")
)
```

Arguments

`x`	An object of class `antaresDataList` with at least components `areas` and `links`.
`storageFlexibility`	A vector containing the names of the virtual storage/flexibility areas. Can also be a named list. Names are columns to add and elements the virtual areas to group.
`production`	A vector containing the names of the virtual production areas.
`reassignCosts`	If TRUE, the production costs of the virtual areas are reallocated to the real areas they are connected to. If the virtual areas are connected to a virtual hub, their costs are first reallocated to the hub and then the costs of the hub are reallocated to the real areas.
`newCols`	If `TRUE`, new columns containing the production of the virtual areas are added. If FALSE their production is added to the production of the real areas they are connected to.

rowBal	If TRUE, then BALANCE will be corrected by ROW. BAL: BALANCE := BALANCE - "ROW. BAL"
prodVars	Virtual productions columns to add to real area. Default to getAlias("rmVA_production")
costsVars	If parameter reassignCosts is TRUE, affected columns. Default to OV. COST, OP. COST, CO2 EMIS. and NP COST
costsOn	If parameter reassignCosts is TRUE, then the costs of the virtual areas are reassigned to the real areas they are connected to. You can choose to reassigned production & storageFlexibility virtuals areas ("both", default), or only "production" or "storageFlexibility" virtuals areas

Details

Two types of virtual areas have been defined corresponding to different types of modeling in Antares and different types of post-treatment to do:

- Flexibility/storage areas are areas created to model pumping unit or any other flexibility that behave as a storage. For those virtual areas, the important results are flows on the links.
- Production areas are areas created to isolate some generation from the "real" areas. They can be isolate for several reasons: to distinguish time-series (for example wind onshore/offshore), to select some specific unit to participate to day-ahead reserve, etc.

removeVirtualAreas performs different corrections:

- Correct the balance of the real areas (and districts) by removing the flows to or from virtual areas.
- If parameter reassignCosts is TRUE, then the costs of the virtual areas are reassigned to the real areas they are connected to. The default affected columns are OV. COST, OP. COST, CO2 EMIS. and NP COST. If a virtual area is connected to a single real area, all its costs are attributed to the real area. If it is connected to several real areas, then costs at a given time step are divided between them proportionally to the flows between them and the virtual area. An aggregation is done at the end to correct districts costs.
- For each storage/flexibility area, a column named like the area is created. It contains the values of the flow between the virtual area and the real areas. This column is interpreted as a production of electricity: it is positive if the flow from the virtual area to the real area is positive and negative otherwise. If parameter newCols is FALSE, the values are added to the variable PSP and the columns is removed. An aggregation is done at the end to add virtual storage/flexibility to districts.
- If the parameter production is specified, then the non null productions of the virtual areas are either added to the ones of the real areas they are connected to if newCols = FALSE or put in new columns if newCols = TRUE. In the second case the columns are named *_virtual where "*" is a type of production (wind, solar, nuclear, ...). Productions that are zero for all virtual areas are omited. If virtual production areas contains clusters then they will be move to the real area. An aggregation is done at the end to add virtual production to districts.
- Finally, virtual areas and the links connected to them are removed from the data.

The functions makes a few assumptions about the network. If they are violated it will not act correctly:

- storage/flexibility areas can be connected to other storage/flexibility areas (hubs), but at least one of them is connected to a real area. That means that there is no group of virtual areas disconnected from the real network. If such a group exists, you can either remove them manually or simply not import them.
- production areas are connected to one and only one real area. They cannot be connected to virtual areas. But a real area may by connected to several production areas.

Value

An `antaresDataList` object in which virtual areas have been removed and data of the real has been corrected. See details for an explanation of the corrections.

Examples

```
## Not run:

# Assume we have a network with two virtual areas acting as pump storage and
# an area representing offshore production
#
#  offshore
#     |
# real area - psp in
#           \
#             psp out
#

data <- readAntares(areas="all", links="all")

# Remove pump storage virtual areas

correctedData <- removeVirtualAreas(
    x = data,
    storageFlexibility = c("psp in", "psp out"),
    production = "offshore"
)

correctedData_list <- removeVirtualAreas(
    x = data,
    storageFlexibility = list(PSP = c("psp in", "psp out")),
    production = "offshore"
)


correctedData_details <- removeVirtualAreas(
    x = data,
    storageFlexibility = list(PSP_IN = "psp in", PSP_OUT =  "psp out"),
    production = "offshore"
)


## End(Not run)
```

setHvdcAreas *Set hvdc areas*

Description

This function add hvdc attribute

Usage

```
setHvdcAreas(data, areas)
```

Arguments

data	antaresData or antaresDatalist data.
areas	character hvdc areas list.

Value

A list with three elements:

Examples

```
## Not run:

library(antaresRead)
opts <- setSimulationPath('mypath', 1)
myAreaOutput <- readAntares(areas = "all", links = "all")
myAreaOutput <- setHvdcAreas(myAreaOutput, "y_dsr")



## End(Not run)
```

setRam *Specify RAM limit*

Description

This function specify RAM limit (in Go) of the value returned by readAntares.

Usage

```
setRam(x)
```

Arguments

x	numeric RAM limit in Go

Examples

```
## Not run:
#Set maximum ram to used to 50 Go
setRam(50)

## End(Not run)
```

---

setSimulationPath *Set Path to an Antares simulation*

Description

This function has to be used before the read functions. It sets the path to the Antares simulation to work on and other useful options (list of areas, links, areas with clusters, variables, etc.). On local disk with setSimulationPath or on an AntaREST API with setSimulationPathAPI

Usage

```
setSimulationPath(path, simulation = NULL)

setSimulationPathAPI(
  host,
  study_id,
  token,
  simulation = NULL,
  timeout = 60,
  httr_config = list()
)
```

Arguments

path	(optional) Path to the simulation. It can either be the path to a directory containing an antares project or directly to the directory containing the output of a simulation. If missing, a window opens and lets the user choose the directory of the simulation interactively. Can also choose .h5 file, if rhdf5 is installed.
simulation	(optional) Only used if "path" represents the path of a study and not of the output of a simulation. It can be either the name of the simulation or a number indicating which simulation to use. It is possible to use negative values to select a simulation from the last one: for instance -1 will select the most recent simulation, -2 will the penultimate one, etc. There are two special values 0 and "input" that tells the function that the user is not interested by the results of any simulation, but only by the inputs. In such a case, the function readAntares is unavailable.

host	character host of AntaREST server API
study_id	character id of the target study on the API
token	character API personnal access token
timeout	numeric API timeout (seconds). Default to 60. See also setTimeoutAPI
httr_config	API httr configuration. See config

Details

The simulation chosen with setSimulationPath or setSimulationPathAPI becomes the default simulation for all functions of the package. This behavior is fine when working on only one simulation, but it may become problematic when working on multiple simulations at same time.

In such case, you can store the object returned by the function in a variable and pass this variable to the functions of the package (see examples).

Value

A list containing various information about the simulation, in particular:

studyPath	path of the Antares study
simPath	path of the simulation
inputPath	path of the input folder of the study
studyName	Name of the study
simDataPath	path of the folder containing the data of the simulation
name	name of the simulation
mode	type of simulation: economy, adequacy, draft or input
synthesis	Are synthetic results available ?
yearByYear	Are the results for each Monte Carlo simulation available ?
scenarios	Are the Monte-Carlo scenarii stored in output ? This is important to reconstruct some input time series that have been used in each Monte-Carlo simulation.
mcYears	Vector containing the number of the exported Monte-Carlo scenarios
antaresVersion	Version of Antares used to run the simulation.
areaList	Vector of the available areas.
districtList	Vector of the available districts.
linkList	Vector of the available links.
areasWithClusters	Vector of areas containing clusters.
variables	Available variables for areas, districts and links.
parameters	Other parameters of the simulation.
timeIdMin	Minimum time id of the simulation. It is generally equal to one but can be higher if working on a subperiod.
timeIdMax	maximum time id of the simulation.

start	Date of the first day of the year in the simulation. This date corresponds to timeId = 1.
firstWeekday	First day of the week.
districtsDef	data.table containing the specification of the districts.
energyCosts	list containing the cost of spilled and unsupplied energy.

See Also

simOptions, readAntares, readLayout, readClusterDesc, readBindingConstraints

Examples

```
## Not run:
# Select interactively a study. It only works on windows.

setSimulationPath()

# Specify path of the study. Note: if there are more than one simulation
# output in the study, the function will asks the user to interactively choose
# one simulation.

setSimulationPath("path_of_the_folder_of_the_study")

# Select the first simulation of a study

setSimulationPath("path_of_the_folder_of_the_study", 1)

# Select the last simulation of a study

setSimulationPath("path_of_the_folder_of_the_study", -1)

# Select a simulation by name

setSimulationPath("path_of_the_folder_of_the_study", "name of the simulation")

# Just need to read input data

setSimulationPath("path_of_the_folder_of_the_study", "input")
# or
setSimulationPath("path_of_the_folder_of_the_study", 0)

# Working with API
#--------------------------
setSimulationPathAPI(
    host = "http://antares_api_adress",
    study_id = "study_id_on_api",
    token = "token"
)

## Custom httr options ?
```

```
# global using httr package
require(httr)
set_config(verbose())
setSimulationPathAPI(
    host = "http://antares_api_adress",
    study_id = "study_id_on_api",
    token = "token"
)

reset_config()

# or in setSimulationPathAPI
setSimulationPathAPI(
    host = "http://antares_api_adress",
    study_id = "study_id_on_api",
    token = "token",
    httr_config = config(verbose = TRUE)
)

# disable ssl certificate checking ?
setSimulationPathAPI(
    host = "http://antares_api_adress",
    study_id = "study_id_on_api",
    token = "token",
    httr_config = config(ssl_verifypeer = FALSE)
)

# WORKING WITH MULTIPLE SIMULATIONS
#----------------------------------
# Let us assume ten simulations have been run and we want to collect the
# variable "LOAD" for each area. We can create a list containing options
# for each simulation and iterate through this list.

opts <- lapply(1:10, function(i) {
   setSimulationPath("path_of_the_folder_of_the_study", i)
})

output <- lapply(opts, function(o) {
  res <- readAntares(areas = "all", select = "LOAD", timeStep = "monthly", opts = o)
  # Add a column "simulation" containing the name of the simulation
  res$simulation <- o$name
  res
})

# Concatenate all the tables in one super table
output <- rbindlist(output)

# Reshape output for easier comparisons: one line per timeId and one column
# per simulation
output <- dcast(output, timeId + areaId ~ simulation, value.var = "LOAD")

output
```

```
# Quick visualization
matplot(output[area == area[1], !c("area", "timeId"), with = FALSE],
        type = "l")

## End(Not run)
```

`setTimeoutAPI` *Change API Timeout*

Description

Change API Timeout

Usage

```
setTimeoutAPI(opts, timeout)
```

Arguments

`opts`	list of simulation parameters returned by the function `setSimulationPathAPI`
`timeout`	`numeric` API timeout (seconds). Default to 60.

Examples

```
## Not run:
opts <- setTimeoutAPI(opts, timeout = 45)

## End(Not run)
```

`showAliases` *show aliases for variables*

Description

Aliases are short names that can be used in the `select` parameter in function `readAntares` to tell the function which columns and/or type of data to import.

`setAlias` can be used to create a new alias. It can be especially useful for package developers to help their users select the data required by their packages.

`getAlias` return character vector containing columns and/or types of data

`showAliases` lists available aliases

Usage

```
showAliases(names = NULL)

setAlias(name, desc, select)

getAlias(name)
```

Arguments

names	optional vector of alias names. If provided, the full list of columns selected by these aliases is displayed. Else only the name and a short description of all aliases is displayed.
name	Alias name
desc	Short description indicating why the new alias is interesting
select	character vector containing columns and/or types of data to import.

Value

`setAlias` is only used for its side effects. A data.frame with columns 'name', 'desc' and 'select'. `showAliases` invisibly returns a data.frame with columns "name", "desc" and "select".

Examples

```
# Display the short description of an alias
showAliases()

# Display the full description of an alias
showAliases("renewable")

getAlias("renewable")

## Not run:
# Create a new alias that imports flows
setAlias("test", "short description", c("links", "FLOW LIN."))
showAliases()

## End(Not run)
```

`simOptions` *Extract simulation options*

Description

The function `readAntares` stores in its output the options used to read some data (path of the study, area list, link list, start date, etc.).

Usage

```
simOptions(x = NULL)
```

Arguments

x	object of class antaresTable or antaresData

Details

simOptions extracts these options from an object of class antaresTable or antaresOutput. It can be useful when working on multiple simulations, either to check how some object has been created or to use it in some functions like getAreas or getLinks

If the parameter of the function is NULL, it returns the default simulation options, that is the options set by setSimulationPath the last time it was run.

Value

list of options used to read the data contained in an object or the last simulation options read by setSimulationPath if x is NULL

Examples

```
## Not run:
  setSimulationPath(study1)

  simOptions() # returns the options for study 1

  data <- readAntares()

  # Choose a different study
  setSimulationPath(study2)

  simOptions() # returns the options for study 2

  getAreas() # returns the areas of the secund study
  getAreas(opts = simOptions(data)) # returns the areas of the first study


## End(Not run)
```

---

subset.antaresDataList

Subset an antaresDataList

Description

Subset method for antaresDataList.

Usage

```
## S3 method for class 'antaresDataList'
subset(x, y = NULL, areas = NULL, timeIds = NULL, mcYears = NULL, ...)
```

Arguments

x	Object of class antaresDataList created with readAntares.
y	A table containing at least one of the columns "area", "timeId" or "mcYear". If it is not NULL, then only tuples (area, timeId,mcYear) present in this table are kept.
areas	Vector of area names to keep in the result. If NULL, all areas are kept.
timeIds	Vector of time ids to keep. If NULL, all time ids are kept.
mcYears	Vector of monte-carlo years to keep. If NULL, all time ids are kept.
...	Currently unused.

Value

A filtered antaresDataList.

Examples

```
## Not run:
#keep only the first year
mydata <- readAntares(areas = "all", links = "all", mcYears = "all")
mySubset<-subset(mydata, mcYears = 1)

#keep only the first year for areas a and b
mydata <- readAntares(areas = "all", links = "all", mcYears = "all")
mySubset<-subset(mydata, mcYears = 1, areas=c("a", "b"))

#' #keep only the first year for areas a and b and timeIds include in 5:16
mydata <- readAntares(areas = "all", links = "all", mcYears = "all")
mySubset<-subset(mydata, mcYears = 1, areas=c("a", "b"), timeIds=5:16)


## End(Not run)
```

---

viewAntares *View the content of an antares output*

Description

This function displays each element of an antaresData object in a spreadsheet-like viewer.

Usage

```
viewAntares(x, ...)
```

Arguments

x	An object of class antaresData, generated by the function readAntares.
...	Currently unused

Value

Invisible NULL.

Examples

```
## Not run:
setSimulationPath()

areas <-readAntares()
viewAntares(areas)

output <- studyAntares(areas="all", links = "all", clusters = "all")
viewAntares(output) # Opens three data viewers for each element of output

## End(Not run)
```

---

writeAntaresH5 *Convert antares output to h5 file*

Description

Convert antares output to h5 file

Usage

```
writeAntaresH5(
  path = NULL,
  timeSteps = c("hourly", "daily", "weekly", "monthly", "annual"),
  opts = simOptions(),
  writeMcAll = TRUE,
  compress = 1,
  misc = FALSE,
  thermalAvailabilities = FALSE,
  hydroStorage = FALSE,
  hydroStorageMaxPower = FALSE,
  reserve = FALSE,
  linkCapacity = FALSE,
  mustRun = FALSE,
  thermalModulation = FALSE,
  allData = FALSE,
  writeAllSimulations = FALSE,
```

```
  nbCores = 4,
  removeVirtualAreas = FALSE,
  storageFlexibility = NULL,
  production = NULL,
  reassignCosts = FALSE,
  newCols = TRUE,
  overwrite = FALSE,
  supressMessages = FALSE
)
```

Arguments

path	character folder where h5 file will be write (default NULL)
timeSteps	character timeSteps
opts	list of simulation parameters returned by the function setSimulationPath. Default to antaresRead::simOptions()
writeMcAll	boolean write mc-all
compress	numeric compress level
misc	boolean see readAntares
thermalAvailabilities	boolean see readAntares
hydroStorage	boolean see readAntares
hydroStorageMaxPower	boolean see readAntares
reserve	boolean see readAntares
linkCapacity	boolean see readAntares
mustRun	boolean see readAntares
thermalModulation	boolean see readAntares
allData	boolean add all data with a single call (writeMcAll, misc, thermalAvailabilities, hydroStorage, hydroStorageMaxPower reserve, linkCapacity, mustRun, thermalModulation).
writeAllSimulations	boolean, write all simulations of your antares study.
nbCores	numeric, number of cores to use, only used if writeAllSimulations is TRUE
removeVirtualAreas	boolean, remove virtual areas, see removeVirtualAreas
storageFlexibility	character or list, see removeVirtualAreas
production	character or list, see removeVirtualAreas
reassignCosts	boolean or list, see removeVirtualAreas
newCols	boolean or list, see removeVirtualAreas
overwrite	boolean or list, overwrite old file
supressMessages	boolean, supress messages from readAntares and removeVirtualAreas

Examples

```
## Not run:
# Write simulation one by one
setSimulationPath("C:/Users/MyUser/Mystudy", 1)
writeAntaresH5(path="PATH_TO_YOUR_STUDY")

# Write all simulations
setSimulationPath("C:/Users/MyUser/Mystudy")
writeAntaresH5(path="PATH_TO_YOUR_STUDY", writeAllSimulations = TRUE)

# Choose timestep to write
setSimulationPath("C:/Users/MyUser/Mystudy", 1)
writeAntaresH5(path="PATH_TO_YOUR_STUDY", timeSteps = "hourly")

# Write with additionnal information
writeAntaresH5(path="PATH_TO_YOUR_STUDY", timeSteps = "hourly",
   misc = TRUE, thermalAvailabilities = TRUE,
   hydroStorage = TRUE, hydroStorageMaxPower = TRUE, reserve = TRUE,
   linkCapacity = TRUE, mustRun = TRUE, thermalModulation = TRUE)

# Write all data with a shorcut
writeAntaresH5(path="PATH_TO_YOUR_STUDY", allData = TRUE)

#Remove virtuals areas

writeAntaresH5(path="PATH_TO_YOUR_STUDY", timeSteps = "hourly", overwrite = TRUE,
               writeMcAll = FALSE, removeVirtualAreas = TRUE,
               storageFlexibility =  "psp in-2",
               production =  NULL, reassignCosts =FALSE, newCols = TRUE)

#Remove virtuals areas more than one call
writeAntaresH5(
               path="PATH_TO_YOUR_STUDY",
               timeSteps = "hourly",
               overwrite = TRUE,
               writeMcAll = FALSE,
               removeVirtualAreas = TRUE,
               storageFlexibility = list("psp out", "psp in-2"),
               production = list(NULL, NULL),
               reassignCosts = list(TRUE, FALSE),
               newCols = list(FALSE, TRUE)
               )



## End(Not run)
```

Index