Package 'antaresEditObject'

April 19, 2024

Type Package

Title Edit an 'Antares' Simulation

Version 0.6.2

Description Edit an 'Antares' simulation before running it : create new areas, links, thermal clusters or binding constraints or edit existing ones. Update 'Antares' general & optimization settings.
'Antares' is an open source power system generator, more information available here : <https://antares-simulator.org/>.

License GPL (>= 2) | file LICENSE

URL https://github.com/rte-antares-rpackage/antaresEditObject, https://rte-antares-rpackage.github.io/antaresEditObject/

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

Encoding UTF-8

RoxygenNote 7.2.2

Depends antaresRead (>= 2.4.2)

Imports assertthat, cli, data.table, httr, grDevices, jsonlite, whisker, doParallel, doFuture, memuse, progressr, pbapply, parallel, future, plyr, yaml

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Tatiana Vargas [aut, cre],
Frederic Breant [aut],
Victor Perrier [aut],
Baptiste Seguinot [ctb],
Benoit Thieurmel [ctb],
Titouan Robert [ctb],
Jalal-Edine Zawam [ctb],
Etienne Sanchez [ctb],
Janus De Bondt [ctb],
Assil Mansouri [ctb],

Abdallah Mahoudi [ctb],
Clement Berthet [ctb],
Kamel Kemiha [ctb],
Nicolas Boitard [ctb],
RTE [cph]

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

Repository CRAN

Date/Publication 2024-04-19 12:22:53 UTC

R topics documented:

activateRES *Activate RES in an Antares study*

Description

Helper to activate Renewables Energy Sources. This will update `renewable.generation.modelling` parameter and create appropriate structure for RES clusters.

Usage

```
activateRES(opts = antaresRead::simOptions(), quietly = !interactive())
```

Arguments

`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath`
`quietly`	Display or not a message to the user if success.

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

library(antaresEditObject)
tmp <- tempfile()
createStudy(path = tmp)
opts <- antaresRead::setSimulationPath(tmp)
activateRES()

# then you can use createClusterRES()...


## End(Not run)
```

activateST *Activate st-storage in an Antares study*

Description

Activate st-storage in an Antares study

Usage

```
activateST(opts = antaresRead::simOptions(), quietly = !interactive())
```

Arguments

opts	List of simulation parameters returned by the function antaresRead::setSimulationPath
quietly	Display or not a message to the user if success.

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

library(antaresEditObject)
tmp <- tempfile()
createStudy(path = tmp)
opts <- antaresRead::setSimulationPath(tmp)
activateST()

# then you can use createClusterST()...


## End(Not run)
```

---

add_week_number_column_to_ts

Add week number column to a data.time of time series type

Description

If timeId column exists, add a week number column. A week is 168 consecutive hours (= 24 * 7).

Usage

```
add_week_number_column_to_ts(xts)
```

Arguments

xts	a data.table of time series type.

Value

the data.table xts with a new column week.

adequacyOptions *Adequacy patch parameters for creating an area*

Description

Adequacy patch parameters for creating an area

Usage

```
adequacyOptions(adequacy_patch_mode = "outside")
```

Arguments

adequacy_patch_mode	character, default to "outside"

Value

a named list

Examples

```
adequacyOptions()
```

backupStudy *Create a backup with an Antares Study*

Description

Antares API: **NO**

Save an Antares Study or only inputs in a `.tar.gz` or `.zip` file

Usage

```
backupStudy(
  backupfile,
  what = "study",
  opts = antaresRead::simOptions(),
  extension = ".zip"
)
```

Arguments

backupfile	Name of the backup, without extension. If missing, either the name of the study or 'input' according argument what.
what	Which folder to save, input for the input folder or study for the whole study.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath
extension	Defaut is .zip.

Value

The path of the backup

Examples

```
## Not run:
backupStudy()

## End(Not run)
```

---

check-version *Is study an Antares v7 study ?*

Description

Is study an Antares v7 study ?

Usage

```
is_antares_v7(opts = antaresRead::simOptions())

is_antares_v820(opts = antaresRead::simOptions())
```

Arguments

opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

a logical, TRUE if study is v7 or above, FALSE otherwise.

Examples

```
## Not run:
# setSimulationPath

is_antares_v7()
```

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

---

checkRemovedArea *Seek for a removed area*

Description

Check if it remains trace of a deleted area in the input folder

Usage

```
checkRemovedArea(area, all_files = TRUE, opts = antaresRead::simOptions())
```

Arguments

area	An area
all_files	Check files in study directory.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath

Value

a named list with two elements

Examples

```
## Not run:
checkRemovedArea("myarea")

## End(Not run)
```

---

check_consistency_reservoir_values
For a given area, check consistency between reservoir and reservoir capacity values

---

Description

For a given area, check consistency between reservoir and reservoir capacity values

Usage

```
check_consistency_reservoir_values(area, new_data, prev_data)
```

Arguments

area	The area where to run the check.
new_data	The new list of parameters.
prev_data	The previous data found in hydro.ini.

---

check_mingen_vs_hydro_storage

Check if mingen data and hydro storage data are consistent

Description

At each weekly/monthly/annual time step, mingen must be less or equal than hydro storage.

Usage

```
check_mingen_vs_hydro_storage(area, opts = antaresRead::simOptions())
```

Arguments

area	The area where to check the data.
opts	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`.

Value

a list containing the boolean if the check is ok and the message to display.

Note

Function called only for an **Antares version >= 860**.

---

check_mingen_vs_maxpower

Check if mingen data and maxpower data are consistent

Description

At each hourly time step, mingen must be less or equal than generatingMaxPower.

Usage

```
check_mingen_vs_maxpower(area, opts = antaresRead::simOptions())
```

Arguments

area	The area where to check the data.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

Value

a list containing the boolean if the check is ok and the message to display.

Note

Function called only for an **Antares version >= 860**.

---

cleanUpOutput *Clean up output based on geographic trimming*

Description

Clean up output based on geographic trimming

Usage

```
cleanUpOutput(areas = NULL, opts = simOptions())
```

Arguments

areas	Character. vector of areas (folders). Links will also be cleaned based on getLinks() results
opts	List. simulation options

---

computeOtherFromHourlyMulti *Compute daily, weekly, monthly and annual mc-ind from hourly data multiyear. (new)*

Description

Compute daily, weekly, monthly and annual mc-ind from hourly data multiyear. (new)

Usage

```
computeOtherFromHourlyMulti(
  opts = simOptions(),
  areas = "all",
  type = c("areas", "links", "clusters"),
  timeStep = c("daily", "monthly", "annual", "weekly"),
  mcYears = simOptions()$mcYears,
  writeOutput = F,
  nbcl = 8,
  verbose = F
)
```

Arguments

opts	study opts
areas	vector of areas
type	type of aggregation
timeStep	timestep of aggregation (daily, monthly and annual, NO weekly)
mcYears	vector of years to compute
writeOutput	boolean to write data in mc-ind folder
nbcl	number of cpu cores for parallelization
verbose	logical for printing output

Note

Recommended only with studies spanning on two years.

See Also

computeOtherFromHourlyYear

---

computeOtherFromHourlyYear

Compute daily, weekly, monthly and annual mc-ind from hourly data for one year. (new)

Description

Compute daily, weekly, monthly and annual mc-ind from hourly data for one year. (new)

Usage

```
computeOtherFromHourlyYear(
  mcYear,
  type,
  areas = "all",
  opts = simOptions(),
  timeStep = c("daily", "monthly", "annual", "weekly"),
  writeOutput = F
)
```

Arguments

mcYear	vector of years to compute
type	type of data (areas, links, clusters, clustersRes)
areas	vector of areas. links type will use getLinks() to get data.
opts	study opts
timeStep	timestep of aggregation (daily, monthly and annual, NO weekly)
writeOutput	boolean to write data in mc-ind folder

Note

Recommended only with studies spanning on two years.

See Also

computeOtherFromHourlyMulti

---

computeTimeStampFromHourly

Compute daily, weekly, monthly and annual mc-ind from hourly data.

Description

Antares API: **NO**

Compute daily, weekly, monthly and annual mc-ind from hourly data.

Usage

```
computeTimeStampFromHourly(
  opts,
  mcYears = "all",
  nbcl = 8,
  verbose = 1,
  type = c("areas", "links", "clusters")
)
```

Arguments

opts	opts simulation path.
mcYears	mcYears to compute.
nbcl	number of thread for parallel computing.
verbose	verbose for execution.
type	type of file to compute.

Note

Deprecated on studies v8 or higher.

Examples

```
## Not run:

library(antaresEditObject)
opts <- setSimulationPath("my_study")
computeTimeStampFromHourly(opts)


## End(Not run)
```

---

convertConfigToAdq *Read adequacy patch config.yml into Antares (v8.5+)*

Description

Use this function to load config.yml used in older Antares versions for adequacy patch. Areas in config will be updated to be included in adequacy patch perimeter.

Usage

```
convertConfigToAdq(opts = simOptions(), path = "default")
```

Arguments

opts	List. study options.
path	Character. path to config.yml. Default points to "/user/adequacypatch/" in study

See Also

updateAdequacySettings

copyOutput *Copy of the output files of an Antares study*

Description

Antares API: **NO**

Copy of the output files of an Antares study.

Usage

```
copyOutput(opts, extname, mcYears = "all")
```

Arguments

opts	List of simulation parameters returned by the function antaresRead::setSimulationPath
extname	Extension to be added to the name of the study, to be used as a name for the newly created folder.
mcYears	mcYears to copy. Can be "all".

Examples

```
## Not run:

library(antaresRead)

# Set simulation path
opts = setSimulationPath(path = "PATH/TO/SIMULATION", simulation = "input")

# Create a new area
copyOutput(opts, "_adq")


## End(Not run)
```

copyStudyWeb *Import physical study to Antares Web (managed study)*

Description

Copy study from an existing workspace into a managed study. NOTE : The study must be present in a workspace (DRD, PPSE..) not just locally.

Usage

```
copyStudyWeb(
  opts = antaresRead::simOptions(),
  host,
  token,
  outputs = T,
  groups = NULL,
  suffix = "managedCopy"
)
```

Arguments

opts	List of simulation parameters returned by the function antaresRead::setSimulationPath. If id is not available, antaresRead::searchStudy will be used to find study.
host	Host of AntaREST server API.
token	API personnal access token.
outputs	Logical. Determine if outputs are copied too.
groups	Character. Add study to groups of Antares Web.
suffix	Character. default is "managedCopy" By default the new study will be : study-name_managedCopy

Value

New managed study ID

---

create-binding-constraint

Create a binding constraint

Description

Antares API: **OK**

Create a new binding constraint in an Antares study. createBindingConstraintBulk() allow to create multiple constraints at once.

Usage

```
createBindingConstraint(
  name,
  id = tolower(name),
  values = NULL,
  enabled = TRUE,
  timeStep = c("hourly", "daily", "weekly"),
  operator = c("both", "equal", "greater", "less"),
  filter_year_by_year = "hourly, daily, weekly, monthly, annual",
```

```
  filter_synthesis = "hourly, daily, weekly, monthly, annual",
  coefficients = NULL,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)

createBindingConstraintBulk(constraints, opts = antaresRead::simOptions())
```

Arguments

`name`	The name for the binding constraint.
`id`	An id, default is to use the name.
`values`	Values used by the constraint. It contains one line per time step and three columns "less", "greater" and "equal".
`enabled`	Logical, is the constraint enabled ?
`timeStep`	Time step the constraint applies to : `hourly`, `daily` or `weekly`.
`operator`	Type of constraint: equality, inequality on one side or both sides.
`filter_year_by_year`	Marginal price granularity for year by year
`filter_synthesis`	Marginal price granularity for synthesis
`coefficients`	A named vector containing the coefficients used by the constraint, the coefficients have to be alphabetically ordered.
`overwrite`	If the constraint already exist, overwrite the previous value.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`
`constraints`	A `list` of several named `list` containing data to create binding constraints. **Warning** all arguments for creating a binding constraints must be provided, see examples.

Value

An updated list containing various information about the simulation.

See Also

`editBindingConstraint()` to edit existing binding constraints, `removeBindingConstraint()` to remove binding constraints.

Examples

```
## Not run:
createBindingConstraint(
  name = "myconstraint",
  values = matrix(data = rep(0, 8760 * 3), ncol = 3),
  enabled = FALSE,
  timeStep = "hourly",
  operator = "both",
```

```
  coefficients = c("fr%myarea" = 1)
)

# Create multiple constraints

# Prepare data for constraints
bindings_constraints <- lapply(
  X = seq_len(100),
  FUN = function(i) {
    # use arguments of createBindingConstraint()
    # all arguments must be provided !
    list(
      name = paste0("constraints", i),
      id = paste0("constraints", i),
      values = matrix(data = rep(0, 8760 * 3), ncol = 3),
      enabled = FALSE,
      timeStep = "hourly",
      operator = "both",
      coefficients = c("area1%area2" = 1),
      overwrite = TRUE
    )
  }
)
# create all constraints
createBindingConstraintBulk(bindings_constraints)

## End(Not run)
```

`create-study` *Create an empty Antares study*

Description

Create study on disk or with AntaREST server through the API.

Usage

```
createStudy(path, study_name = "my_study", antares_version = "8.2.0")

createStudyAPI(
  host,
  token = NULL,
  study_name = "my_study",
  antares_version = "8.2.0",
  ...
)
```

Arguments

path	Path where to create study, it should be an empty directory, if it doesn't exist, it'll be created.
study_name	Name of the study.
antares_version	Antares number version.
host	Host of AntaREST server API.
token	API personnal access token.
...	Other query parameters passed to POST request.

Value

Result of `antaresRead::setSimulationPath()` or `setSimulationPathAPI()` accordingly.

Examples

```
## Not run:

createStudy("path/to/simulation")


## End(Not run)
```

---

createArea *Create an area in an Antares study*

Description

Antares API: **OK**

Create a new area in an Antares study.

Usage

```
createArea(
  name,
  color = grDevices::rgb(230, 108, 44, max = 255),
  localization = c(0, 0),
  nodalOptimization = nodalOptimizationOptions(),
  filtering = filteringOptions(),
  adequacy = adequacyOptions(),
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)
```

Arguments

name	Name of the area as a character, without punctuation except - and _.
color	Color of the node
localization	Localization on the map
nodalOptimization	Nodal optimization parameters, see nodalOptimizationOptions().
filtering	Filtering parameters, see filteringOptions().
adequacy	Adequacy parameters, see adequacyOptions().
overwrite	Overwrite the area if already exist.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

See Also

editArea(), removeArea()

Examples

```
## Not run:

library(antaresRead)

# Set simulation path
setSimulationPath(path = "PATH/TO/SIMULATION", simulation = "input")

# Create a new area
createArea("fictive_area")


## End(Not run)
```

---

createCluster *Create a cluster*

Description

Antares API: **OK** (thermal clusters only)

Create a new thermal or RES (renewable energy source) cluster.

Usage

```
createCluster(
  area,
  cluster_name,
  group = "Other",
  ...,
  list_pollutants = NULL,
  time_series = NULL,
  prepro_data = NULL,
  prepro_modulation = NULL,
  add_prefix = TRUE,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)

createClusterRES(
  area,
  cluster_name,
  group = "Other RES 1",
  ...,
  time_series = NULL,
  add_prefix = TRUE,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where to create the cluster.
cluster_name	Name for the cluster, it will prefixed by area name, unless you set add_prefix = FALSE.
group	Group of the cluster, depends on cluster type: • thermal cluster, one of: Gas, Hard coal, Lignite, Mixed fuel, Nuclear, Oil, Other, Other 2, Other 3, Other 4. • renewable cluster, one of: Wind Onshore, Wind Offshore, Solar Thermal, Solar PV, Solar Rooftop, Other RES 1, Other RES 2, Other RES 3, Other RES 4.
...	Parameters to write in the Ini file. Careful! Some parameters must be set as integers to avoid warnings in Antares, for example, to set unitcount, you'll have to use unitcount = 1L.
list_pollutants	list named with specific pollutants (only for Antares version >= 860)
time_series	the "ready-made" 8760-hour time-series available for simulation purposes.
prepro_data	Pre-process data, a data.frame or matrix, default is a matrix with 365 rows and 6 columns.

prepro_modulation
: Pre-process modulation, a `data.frame` or `matrix`, if specified, must have 8760 rows and 1 or 4 columns.

add_prefix
: If `TRUE` (the default), `cluster_name` will be prefixed by area name.

overwrite
: Logical, overwrite the cluster or not.

opts
: List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

An updated list containing various information about the simulation.

Note

Parameter `list_pollutants` is only available for Antares studies >= v8.6.0.

You must provide named `list` (numerical values or NULL) :

`list( "nh3"= 0.25, "nox"= 0.45, "pm2_5"= 0.25, "pm5"= 0.25, "pm10"= 0.25, "nmvoc"= 0.25, "so2"= 0.25, "op1"= 0.25, "op2"= 0.25, "op3"= 0.25, "op4"= 0.25, "op5"= NULL, "co2"= NULL)`

See Also

`editCluster()` or `editClusterRES()` to edit existing clusters, `removeCluster()` or `removeClusterRES()` to remove clusters.

Examples

```
## Not run:

library(antaresRead)
library(antaresEditObject)

# Create a cluster :
createCluster(
  area = "fr",
  cluster_name = "my_cluster",
  group = "other",
  unitcount = 1L, # or as.integer(1)
  marginal_cost = 50
)
# by default, cluster name is prefixed
# by the area name
levels(readClusterDesc()$cluster)
# > "fr_my_cluster"

# To prevent this, use `add_prefix`
createCluster(
  area = "fr",
  cluster_name = "my_cluster",
  add_prefix = FALSE,
  group = "other",
  marginal_cost = 50
```

```
)
levels(readClusterDesc()$cluster)
# > "my_cluster"


# Create a RES cluster :
createClusterRES(
  area = "fr",
  cluster_name = "my_cluster_res",
  group = "other",
  unitcount = 1L, # or as.integer(1)
  nominalcapacity = 50,
  ts_interpretation = "power-generation"
)

# You can also specify that the Time-Series of the RES cluster are
# production factors :
createClusterRES(
  area = "fr",
  cluster_name = "my_cluster_res",
  group = "other",
  unitcount = 1L, # or as.integer(1)
  nominalcapacity = 50,
  ts_interpretation = "production-factor"
)


# Pre-process data :

# this is the default data :
createCluster(
  area = "fr",
  cluster_name = "my_cluster",
  prepro_data = matrix(
    data = c(rep(1, times = 365 * 2),
             rep(0, times = 365 * 4)),
    ncol = 6
  )
)

# with a data.frame
createCluster(
  area = "fr",
  cluster_name = "my_cluster",
  prepro_data = data.frame(
    v1 = rep(7, 365), # column name does not matter
    v2 = rep(27, 365),
    v3 = rep(0.05, 365),
    v4 = rep(0.12, 365),
    v5 = rep(0, 365),
    v6 = rep(1, 365)
  )
)
```

```
# Pre-process modulation :
# this is the default data
createCluster(
  area = "fr",
  cluster_name = "my_cluster",
  prepro_modulation = matrix(
    data = c(rep(1, times = 365 * 24 * 3),
             rep(0, times = 365 * 24 * 1)),
    ncol = 4
  )
)

# with a data.frame
createCluster(
  area = "fr",
  cluster_name = "my_cluster",
  prepro_modulation = data.frame(
    var1 = rep(0, 8760), # column name does not matter
    var2 = rep(1, 8760),
    var3 = rep(0, 8760),
    var4 = rep(1, 8760)
  )
)


## End(Not run)
```

`createClusterBulk` *Create serial thermal cluster*

Description

For each area, the thermal cluster data are generated :

- Writing `.ini` files
- Writing time_series files
- Writing prepro_data files
- Writing prepro_modulation files

Usage

```
createClusterBulk(
  cluster_object,
  area_zone,
  add_prefix = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

cluster_object	list mutiple list containing the parameters for writing each cluster
area_zone	character name of area to create cluster
add_prefix	logical prefix cluster name with area name
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Details

see the example to write a cluster object, see the original function createCluster()

Structure of cluster_object :

The list must be structured with named items

- parameter : list of paramaters to write in .ini file
- overwrite : logical to choose to overwrite an existing cluster (if not present, set to FALSE)
- time_series : matrix or data.frame the "ready-made" 8760-hour time-series
- prepro_data : matrix or data.frame Pre-process data
- prepro_modulation : matrix or data.frame Pre-process modulation

Details for sublist cluster_object[["parameter"]] :

- name : Name for the cluster, it will prefixed by area name, unless you set add_prefix = FALSE
- group : Group of the cluster, depends on cluster type
- ... : Parameters to write in the Ini file

Value

An updated list containing various information about the simulation.

list containing meta information about the simulation

Examples

```
## Not run:

# /!\/!\/!\ use or create a study /!\/!\/!\

# data preparation for sutructures
ts <- matrix(rep(c(0, 8000), each = 24*364),
             ncol = 2)

df_pd <- matrix(rep(c(1, 1, 1, 0), each = 24*365),
                ncol = 4)

df_pm <- matrix(data = c(rep(1, times = 365 * 24 * 3), rep(0, times = 365 * 24 * 1)),
                ncol = 4)


# Example cluster object
```

```
zone_test_1 <- list(
  `CCGT old 1`= list(
  parameter= list(
  name= "CCGT old 1",
  group = "Other",
  unitcount= 10L,
  nominalcapacity= 100,
   enabled= "true",
   `min-stable-power`= 80L,
   `min-up-time`= 20L,
   `min-down_time`= 30L),
   overwrite= TRUE,
   time_series = ts_8760,
   prepro_data = df_pd,
   prepro_modulation = df_pm))

 # overwrite existing cluster
zone_test_2 <- list(
 `PEAK`= list(parameter= list(
   name= "PEAK",
   group = "Other"),
   overwrite= TRUE,
   time_series = ts,
   prepro_data = df_pd,
   prepro_modulation = df_pm))

# Create multiple areas with multiple clusters
list_areas <- antaresRead::getAreas()[1:5]

lapply(list_areas, createClusterBulk,
cluster_object = c(zone_test_1, zone_test_2),
add_prefix = TRUE)


## End(Not run)
```

`createClusterST` *Create a short-term storage cluster*

Description

Antares API: **OK**

Create a new ST-storage cluster for >= v8.6.0 Antares studies.

Usage

```
createClusterST(
  area,
  cluster_name,
```

```
  group = "Other1",
  storage_parameters = storage_values_default(),
  PMAX_injection = NULL,
  PMAX_withdrawal = NULL,
  inflows = NULL,
  lower_rule_curve = NULL,
  upper_rule_curve = NULL,
  add_prefix = TRUE,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where to create the cluster.
cluster_name	Name for the cluster, it will prefixed by area name, unless you set add_prefix = FALSE.
group	Group of the cluster, one of : "PSP_open", "PSP_closed", "Pondage", "Battery", "Other". It corresponds to the type of stockage.
storage_parameters	list Parameters to write in the Ini file (see Note).
PMAX_injection	modulation of charging capacity on an 8760-hour basis. The values are float between 0 and 1.
PMAX_withdrawal	modulation of discharging capacity on an 8760-hour basis. The values are float between 0 and 1.
inflows	imposed withdrawals from the stock for other uses, The values are integer.
lower_rule_curve	This is the lower limit for filling the stock imposed each hour. The values are float between 0 and 1.
upper_rule_curve	This is the upper limit for filling the stock imposed each hour. The values are float between 0 and 1.
add_prefix	If TRUE (the default), cluster_name will be prefixed by area name.
overwrite	Logical, overwrite the cluster or not.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Note

To write parameters to the list.ini file. You have function storage_values_default() who is called by default. This function return list containing six parameters for cluster st-storage. See example section.

To write data (.txt file), you have parameter for each output file :

- PMAX-injection.txt
- PMAX-withdrawal.txt
- inflows.txt
- lower-rule-curve.txt
- upper-rule-curve.txt

See Also

editClusterST() to edit existing clusters, readClusterSTDesc() to read cluster, removeClusterST() to remove clusters.

Examples

```
## Not run:

# list for cluster parameters :
storage_values_default()

# create a cluster by default (with default parameters values + default data values):
createClusterST(area = "my_area",
                "my_cluster")

# Read cluster in study
 # by default, cluster name is prefixed
 # by the area name
levels(readClusterSTDesc()$cluster)
# > "my_area_my_cluster"

# create cluster with custom parameter and data
my_parameters <- storage_values_default()
my_parameters$efficiency <- 0.5
my_parameters$reservoircapacity <- 10000


inflow_data <- matrix(3, 8760)
ratio_data <- matrix(0.7, 8760)
createClusterST(area = "my_area",
                 "my_cluster",
                 storage_parameters = my_parameters,
                 PMAX_withdrawal = ratio_data,
                 inflows = inflow_data,
                 PMAX_injection = ratio_data,
                 lower_rule_curve = ratio_data,
                 upper_rule_curve = ratio_data)

## End(Not run)
```

createDistrict *Create a district*

Description

Allows selecting a set of areas so as to bundle them together in a "district".

Usage

```
createDistrict(
  name,
  caption = NULL,
  comments = NULL,
  apply_filter = c("none", "add-all", "remove-all"),
  add_area = NULL,
  remove_area = NULL,
  output = FALSE,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)
```

Arguments

name	District's name.
caption	Caption for the district.
comments	Comments for the district.
apply_filter	Possible values are add-all to add all areas to the district, remove-all to clear the district, or none (default) to don't apply a filter.
add_area	Character vector of area(s) to add to the district.
remove_area	Character vector of area(s) to remove from the district.
output	Logical, compute the results for the district or not?
overwrite	Logical, should the district be overwritten if already exist?
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:
createDistrict(
  name = "mydistrict",
  apply_filter = "add-all",
  remove_area = c("fr", "be")
```

```
)

## End(Not run)
```

---

createDSR *Create a Demand Side Response (DSR)*

Description

Antares API: **OK**

Create a Demand Side Response (DSR)

Usage

```
createDSR(
  areasAndDSRParam = NULL,
  spinning = 2,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)

getCapacityDSR(area = NULL, opts = antaresRead::simOptions())

editDSR(
  area = NULL,
  unit = NULL,
  nominalCapacity = NULL,
  marginalCost = NULL,
  spinning = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

areasAndDSRParam	A data.frame with 4 columns area, unit, nominalCapacity, marginalCost and hour. Hour represent the number of activation hours for the DSR per day.
spinning	DSR spinning
overwrite	Overwrite the DSR plant if already exist. This will overwrite the previous area and links.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()
area	an area where to edit the DSR
unit	DSR unit number
nominalCapacity	DSR nominalCapacity
marginalCost	DSR marginalCost

Value

An updated list containing various information about the simulation.

`getCapacityDSR()` returns DSR capacity (unit * nominalCapacity of virtual cluster) of the area

Examples

```
## Not run:

library(antaresEditObject)
path <- pathToYourStudy
opts <- setSimulationPath(path, simulation = "input")

# area, unit, nominalCapacity and marginalCost
dsrData <- data.frame(area = c("a", "b"), unit = c(10,20),
                  nominalCapacity = c(100, 120), marginalCost = c(52, 65), hour = c(3, 7))

createDSR(dsrData)

createDSR(dsrData, spinning = 3, overwrite = TRUE)
getAreas()


## End(Not run)
## Not run:

getCapacityDSR("a")
editDSR("a", unit = 50, nominalCapacity = 8000)
getCapacityDSR("a")


## End(Not run)
## Not run:

getCapacityDSR("a")
editDSR("a", unit = 50, nominalCapacity = 8000, marginalCost = 45, hour = 9)
getCapacityDSR("a")


## End(Not run)
```

`createLink` *Create a link between two areas*

Description

Antares API: **OK**

Create a new link between two areas in an Antares study.

Usage

```
createLink(
  from,
  to,
  propertiesLink = propertiesLinkOptions(),
  dataLink = NULL,
  tsLink = NULL,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)
```

Arguments

from, to	The two areas linked together.
propertiesLink	a named list containing the link properties, e.g. hurdles-cost or transmission-capacities for example. See propertiesLinkOptions().
dataLink	See Details section below.
tsLink	Transmission capacities time series. First N columns are direct TS, following N are indirect ones.
overwrite	Logical, overwrite the previous between the two areas if exist
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Details

The eight potential times-series are:

- **NTC direct** : the upstream-to-downstream capacity, in MW. Default to 1.
- **NTC indirect** : the downstream-to-upstream capacity, in MW. Default to 1.
- **Hurdle cost direct** : an upstream-to-downstream transmission fee, in euro/MWh. Default to 0.
- **Hurdle cost indirect** : a downstream-to-upstream transmission fee, in euro/MWh. Default to 0.
- **Impedances** : virtual impedances that are used in economy simulations to give a physical meaning to raw outputs, when no binding constraints have been defined to enforce Kirchhoff's laws. Default to 0.
- **Loop flow** : amount of power flowing circularly though the grid when all "nodes" are perfectly balanced (no import and no export). Default to 0.
- **PST min** : lower bound of phase-shifting that can be reached by a PST installed on the link, if any. Default to 0.
- **PST max** : upper bound of phase-shifting that can be reached by a PST installed on the link, if any. Default to 0.

According to Antares version, usage may vary :

< v7.0.0 : 5 first columns are used in the following order: NTC direct, NTC indirect, Impedances, Hurdle cost direct, Hurdle cost indirect.

>= v7.0.0 : 8 columns in order above are expected.

>= v8.2.0 : there's 2 cases :

- 8 columns are provided: 2 first are used in `tsLink`, other 6 are used for link data
- 6 columns are provided: you must provide NTC data in `tsLink` argument.

Value

An updated list containing various information about the simulation.

Note

In Antares, areas are sorted in alphabetical order to establish links between. For example, link between "fr" and "be" will appear under "be". So the areas are sorted before creating the link between them, and `dataLink` is rearranged to match the new order.

See Also

`editLink()`, `removeLink()`

Examples

```
## Not run:

library(antaresRead)

# Set simulation path
setSimulationPath(path = "PATH/TO/SIMULATION", simulation = "input")

# Create a link between two areas
createLink(from = "first_area", to  = "second_area")


## End(Not run)
```

`createPSP` *Create a Pumped Storage Power plant (PSP)*

Description

Antares API: **OK**

Create a Pumped Storage Power plant (PSP)

Usage

```
createPSP(
  areasAndCapacities = NULL,
  namePumping = "Psp_In",
  nameTurbining = "Psp_Out",
  hurdleCost = 5e-04,
  timeStepBindConstraint = "weekly",
  efficiency = NULL,
  overwrite = FALSE,
  opts = antaresRead::simOptions()
)

getCapacityPSP(
  area = NULL,
  nameTurbining = "Psp_Out",
  timeStepBindConstraint = "weekly",
  opts = antaresRead::simOptions()
)

editPSP(
  area = NULL,
  capacity = NULL,
  namePumping = "Psp_In",
  nameTurbining = "Psp_Out",
  timeStepBindConstraint = "weekly",
  hurdleCost = 5e-04,
  opts = antaresRead::simOptions()
)
```

Arguments

areasAndCapacities	A data.frame with 2 columns `installedCapacity` and `area`.
namePumping	The name of the pumping area
nameTurbining	The name of the turbining area
hurdleCost	The cost of the PSP
timeStepBindConstraint	Time step for the binding constraint : `daily` or `weekly`
efficiency	The efficiency of the PSP
overwrite	Overwrite the Pumped Storage Power plant if already exist. This will overwrite the previous area and links.
opts	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`
area	an area name
capacity	PSP capacity for the area

Value

An updated list containing various information about the simulation.

`getCapacityPSP()` returns PSP capacity of the area

Examples

```
## Not run:

library(antaresEditObject)
path<-pathToYourStudy
opts<-setSimulationPath(path, simulation = "input")
pspData<-data.frame(area=c("a", "b"), installedCapacity=c(800,900))

createPSP(pspData, efficiency = 0.8)

createPSP(pspData, efficiency = 0.66, overwrite = TRUE)
createPSP(pspData, efficiency = 0.98, timeStepBindConstraint = "daily")
getAreas()


## End(Not run)

## Not run:

getCapacityPSP("a")
editPSP("a", capacity = 8000, hurdleCost = 0.1)
getCapacityPSP("a")

areaName<-"suisse"
createArea(areaName, overwrite = TRUE)
pspData<-data.frame(area=c(areaName), installedCapacity=c(9856))
createPSP(pspData, efficiency = 0.5, overwrite = TRUE, timeStepBindConstraint = "daily")

getCapacityPSP(areaName, timeStepBindConstraint = "daily")



## End(Not run)
```

`create_referential_series_type` *Create the correspondence data frame between the symbol and the type in scenario builder*

Description

Create the correspondence data frame between the symbol and the type in scenario builder

Usage

```
create_referential_series_type()
```

Value

a data.frame.

---

deleteStudy *Delete a study or a simulation*

Description

Delete a study or a simulation

Usage

```
deleteStudy(opts = simOptions(), prompt_validation = FALSE, simulation = NULL)
```

Arguments

opts	List. study options
prompt_validation	logical to put validation message to delete study (default FALSE)
simulation	simulation to be deleted (default NULL)

---

dicoGeneralSettings *Correspondence between arguments of* updateGeneralSettings *and actual Antares parameters.*

Description

Correspondence between arguments of updateGeneralSettings and actual Antares parameters.

Usage

```
dicoGeneralSettings(arg)
```

Arguments

arg	An argument from function updateGeneralSettings.

Value

The corresponding Antares general parameter.

Examples

```
dicoGeneralSettings("year.by.year") # "year-by-year"
```

dicoOptimizationSettings *Correspondence between arguments of* updateOptimizationSettings *and actual Antares parameters.*

Description

Correspondence between arguments of updateOptimizationSettings and actual Antares parameters.

Usage

```
dicoOptimizationSettings(arg)
```

Arguments

arg	An argument from function updateOptimizationSettings.

Value

The corresponding Antares general parameter.

Examples

```
dicoGeneralSettings("year.by.year") # "year-by-year"
```

editArea *Edit an area in an Antares study*

Description

Antares API: **OK**

Edit an existing area in an Antares study.

Usage

```
editArea(
  name,
  color = NULL,
  localization = NULL,
  nodalOptimization = NULL,
  filtering = NULL,
  adequacy = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

name	Name of the area as a character, without punctuation except - and _.
color	Color of the node
localization	Localization on the map
nodalOptimization	Nodal optimization parameters, see nodalOptimizationOptions().
filtering	Filtering parameters, see filteringOptions().
adequacy	Adequacy parameters, see adequacyOptions().
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

See Also

createArea(), removeArea()

Examples

```
## Not run:

library(antaresRead)

# Set simulation path
setSimulationPath(path = "PATH/TO/SIMULATION", simulation = "input")

# Edit an existing area
editArea("area", color = grDevices::rgb(230, 108, 44, max = 255),
  localization = c(1, 1),
  opts = antaresRead::simOptions())

editArea("de",  nodalOptimization = list("spilledenergycost" = list(fr = 30)),
opts = antaresRead::simOptions())

editArea("de",  nodalOptimization = nodalOptimizationOptions(),
opts = antaresRead::simOptions())

editArea(
  "de",
  filtering = list("filter_synthesis"=paste(c("hourly","daily"),collapse = ", "))
)


## End(Not run)
```

`editBindingConstraint` *Update an existing binding constraint*

Description

Antares API: **OK**

Update an existing binding constraint in an Antares study.

Usage

```
editBindingConstraint(
  name,
  id = tolower(name),
  values = NULL,
  enabled = NULL,
  timeStep = NULL,
  operator = NULL,
  filter_year_by_year = NULL,
  filter_synthesis = NULL,
  coefficients = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

`name`	The name for the binding constraint.
`id`	An id, default is to use the name.
`values`	Values used by the constraint. It contains one line per time step and three columns "less", "greater" and "equal".
`enabled`	Logical, is the constraint enabled ?
`timeStep`	Time step the constraint applies to : `hourly`, `daily` or `weekly`.
`operator`	Type of constraint: equality, inequality on one side or both sides.
`filter_year_by_year`	Marginal price granularity for year by year
`filter_synthesis`	Marginal price granularity for synthesis
`coefficients`	A named vector containing the coefficients used by the constraint, the coefficients have to be alphabetically ordered.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

An updated list containing various information about the simulation.

See Also

createBindingConstraint() to create new binding constraints, removeBindingConstraint() to remove binding constraints.

Examples

```
## Not run:
editBindingConstraint(
  name = "myconstraint",
  values = matrix(data = rep(0, 8760 * 3), ncol = 3),
  enabled = FALSE,
  timeStep = "hourly",
  operator = "both",
  coefficients = c("fr%de" = 1)
)

## End(Not run)
```

---

editCluster *Edit an existing cluster*

Description

Antares API: **OK** (thermal clusters only)

Edit parameters, pre-process data and time series of an existing cluster, thermal or RES (renewable energy source).

Usage

```
editCluster(
  area,
  cluster_name,
  ...,
  list_pollutants = NULL,
  time_series = NULL,
  prepro_data = NULL,
  prepro_modulation = NULL,
  add_prefix = TRUE,
  opts = antaresRead::simOptions()
)

editClusterRES(
  area,
  cluster_name,
  ...,
  time_series = NULL,
  add_prefix = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where to create the cluster.
cluster_name	Name for the cluster, it will prefixed by area name, unless you set add_prefix = FALSE.
...	Parameters to write in the Ini file. Careful! Some parameters must be set as integers to avoid warnings in Antares, for example, to set unitcount, you'll have to use unitcount = 1L.
list_pollutants	list named with specific pollutants (only for Antares version >= 860)
time_series	the "ready-made" 8760-hour time-series available for simulation purposes.
prepro_data	Pre-process data, a data.frame or matrix, default is a matrix with 365 rows and 6 columns.
prepro_modulation	Pre-process modulation, a data.frame or matrix, if specified, must have 8760 rows and 1 or 4 columns.
add_prefix	If TRUE (the default), cluster_name will be prefixed by area name.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Note

Parameter list_pollutants is only available for Antares studies >= v8.6.0.

You must provide named list (numerical values or NULL) :

```
list( "nh3"= 0.25, "nox"= 0.45, "pm2_5"= 0.25, "pm5"= 0.25, "pm10"= 0.25, "nmvoc"= 0.25,
"so2"= 0.25, "op1"= 0.25, "op2"= 0.25, "op3"= 0.25, "op4"= 0.25, "op5"= NULL, "co2"= NULL)
```

See Also

createCluster() or createClusterRES() to create new clusters, removeCluster() or removeClusterRES() to remove clusters.

Examples

```
## Not run:

# Update only nominalCapacity for an existing cluster
editCluster(
  area = "myarea",
  cluster_name = "mycluster",
  nominalcapacity = 10600.000
)


## End(Not run)
```

editClusterST *Edit a short-term storage cluster*

Description

Antares API: **OK**

Edit parameters and time series of an existing `st-storage` cluster (Antares studies >= v8.6.0).

Usage

```
editClusterST(
  area,
  cluster_name,
  group = NULL,
  storage_parameters = NULL,
  PMAX_injection = NULL,
  PMAX_withdrawal = NULL,
  inflows = NULL,
  lower_rule_curve = NULL,
  upper_rule_curve = NULL,
  add_prefix = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

`area`	The area where to create the cluster.
`cluster_name`	Name for the cluster, it will prefixed by area name, unless you set `add_prefix = FALSE`.
`group`	Group of the cluster, one of : "PSP_open", "PSP_closed", "Pondage", "Battery", "Other". It corresponds to the type of stockage.
`storage_parameters`	Parameters to write in the Ini file.
`PMAX_injection`	modulation of charging capacity on an 8760-hour basis. The values are float between 0 and 1.
`PMAX_withdrawal`	modulation of discharging capacity on an 8760-hour basis. The values are float between 0 and 1.
`inflows`	imposed withdrawals from the stock for other uses, The values are integer.
`lower_rule_curve`	This is the lower limit for filling the stock imposed each hour. The values are float between 0 and 1.
`upper_rule_curve`	This is the upper limit for filling the stock imposed each hour. The values are float between 0 and 1.

add_prefix	If TRUE (the default), cluster_name will be prefixed by area name.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

See Also

createClusterST() to edit existing clusters, removeClusterST() to remove clusters.

---

editLink *Edit a link between two areas*

Description

Antares API: **OK**

Edit a link between two areas in an Antares study.

Usage

```
editLink(
  from,
  to,
  hurdles_cost = NULL,
  transmission_capacities = NULL,
  asset_type = NULL,
  display_comments = NULL,
  filter_synthesis = NULL,
  filter_year_by_year = NULL,
  dataLink = NULL,
  tsLink = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

from, to	The two areas linked together.
hurdles_cost	Logical, which is used to state whether (linear) transmission fees should be taken into account or not in economy and adequacy simulations
transmission_capacities	Character, one of enabled, ignore or infinite, which is used to state whether the capacities to consider are those indicated in 8760-hour arrays or if zero or infinite values should be used instead (actual values / set to zero / set to infinite)
asset_type	Character, one of ac, dc, gas, virt or other. Used to state whether the link is either an AC component (subject to Kirchhoff's laws), a DC component, or another type of asset.

display_comments
: Logical, display comments or not.

filter_synthesis
: Character, vector of time steps used in the output synthesis, among `hourly`, `daily`, `weekly`, `monthly`, and `annual`

filter_year_by_year
: Character, vector of time steps used in the output year-by-year, among `hourly`, `daily`, `weekly`, `monthly`, and `annual`

dataLink
: See Details section below.

tsLink
: Transmission capacities time series. First N columns are direct TS, following N are indirect ones.

opts
: List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

An updated list containing various information about the simulation.

Note

See `createLink()` for more documentation

See Also

`createLink()`, `removeLink()`

Examples

```
## Not run:
editLink(
  from = "area1",
  to = "area2",
  transmission_capacities = "infinite",
  filter_synthesis = c("hourly","daily"),
  filter_year_by_year = c("weekly","monthly")
)

## End(Not run)
```

---

fill_empty_hydro_ini_file

Write default values in hydro.ini file if the section is empty

---

Description

For a given area, if the data is empty, pick value from default values for `use heuristic`, `follow load` and `reservoir` sections.

Usage

```
fill_empty_hydro_ini_file(area, opts = antaresRead::simOptions())
```

Arguments

area	The area where to write the value, i.e. lhs in the section.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

---

fill_empty_hydro_ts_file

Write default input time series if mingen.txt *or/and* mod.txt *is empty*

Description

Write default input time series if mingen.txt or/and mod.txt is empty

Usage

```
fill_empty_hydro_ts_file(area, opts = antaresRead::simOptions())
```

Arguments

area	The area where to write the input time series.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

---

filteringOptions *Output profile options for creating an area*

Description

Output profile options for creating an area

Usage

```
filteringOptions(
  filter_synthesis = c("hourly", "daily", "weekly", "monthly", "annual"),
  filter_year_by_year = c("hourly", "daily", "weekly", "monthly", "annual")
)
```

Arguments

filter_synthesis
: Character, vector of time steps used in the output synthesis, among `hourly`, `daily`, `weekly`, `monthly`, and `annual`

filter_year_by_year
: Character, vector of time steps used in the output year-by-year, among `hourly`, `daily`, `weekly`, `monthly`, and `annual`

Value

a named list

Examples

```
filteringOptions(
  filter_synthesis=c("hourly","daily"),
  filter_year_by_year=c("weekly","monthly")
)
```

---

getJobLogs *Retrieve job log from API*

Description

Retrieve job log from API

Usage

```
getJobLogs(job_id, opts = antaresRead::simOptions())
```

Arguments

job_id	The job identifier.
opts	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

Logs as character string.

Examples

```
## Not run:

antaresRead::setSimulationPathAPI(
  host = "http://localhost:8080",
  study_id = "39c604fc-687f-46c4-9fa6-59b57ff9c8d1",
  token = NULL,
  simulation = "input"
```

```
)
job <- runSimulation()
getJobLogs(job)


## End(Not run)
```

`getJobs` *Retrieve API jobs*

Description

Retrieve API jobs

Usage

```
getJobs(job_id = NULL, opts = antaresRead::simOptions())
```

Arguments

`job_id`	The job identifier, if `NULL` (default), retrieve all jobs.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

A `data.table` with information about jobs.

Examples

```
## Not run:

getJobs()


## End(Not run)
```

`get_default_hydro_ini_values` *Get default hydro.ini values*

Description

Get default hydro.ini values

Usage

```
get_default_hydro_ini_values()
```

get_type_check_mingen_vs_hydrostorage

Get the type of control to execute using the 3 necessary booleans

Description

Get the type of control to execute using the 3 necessary booleans

Usage

```
get_type_check_mingen_vs_hydrostorage(hydro_params)
```

Arguments

hydro_params	a list of 3 booleans to compute the type of control to make.

Value

a character containing the type of control to execute.

get_type_check_mingen_vs_hydrostorage_to_trigger

Get the type of control to execute between mingen data and hydro storage data

Description

Compute the type of control to make between :

- input/hydro/series/<area>/mingen.txt
- input/hydro/series/<area>/mod.txt

This control is implemented in Antares too.

Usage

```
get_type_check_mingen_vs_hydrostorage_to_trigger(
  area,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where the type of control must be computed.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

Value

a character containing the type of control to execute.

Note

Function called only for an **Antares version >= 860**.

`get_type_check_mingen_vs_maxpower_to_trigger`
Get the type of control to execute between mingen data and maxpower data

---

Description

Compute the type of control to make between :

- `input/hydro/series/<area>/mingen.txt`
- `input/hydro/common/capacity/maxpower_<area>.txt`

This control is implemented in Antares too. No control to execute if `reservoir` section in hydro.ini for the area is set to TRUE.

Usage

```
get_type_check_mingen_vs_maxpower_to_trigger(
  area,
  opts = antaresRead::simOptions()
)
```

Arguments

`area`	The area where the type of control must be computed.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`.

Value

a character containing the type of control to execute.

Note

Function called only for an **Antares version >= 860**.

importZipStudyWeb *Import a local study to Antares Web*

Description

Import a local study to Antares Web

Usage

```
importZipStudyWeb(host, token, zipfile_name, opts = antaresRead::simOptions())
```

Arguments

host	Host of AntaREST server API.
token	API personnal access token.
zipfile_name	Name of the zipfile of the study.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

list_pollutants_values *Output pollutants list for thermal clusters*

Description

Output pollutants list for thermal clusters

Usage

```
list_pollutants_values(multi_values = NULL)
```

Arguments

multi_values	put values to init list values, default as NULL

Value

a named list

Examples

```
list_pollutants_values()
```

mockSimulationAPI *Mock API usage*

Description

Use this to generate command without an active API connection, it allow to use function to edit a study to later on get API commands.

Usage

```
mockSimulationAPI(force = FALSE, antares_version = "8.2.0")
```

Arguments

force	Logical, force mocking simulation even if antaresRead::setSimulationPathAPI has already been called.
antares_version	Antares version number.

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:
# Mock simulation API
mockSimulationAPI()
# Create an area
createArea("new area")
# Get commands
getVariantCommands()

## End(Not run)
```

nodalOptimizationOptions *Nodal optimization parameters for creating an area*

Description

Nodal optimization parameters for creating an area

Usage

```
nodalOptimizationOptions(
  non_dispatchable_power = TRUE,
  dispatchable_hydro_power = TRUE,
  other_dispatchable_power = TRUE,
  spread_unsupplied_energy_cost = 0,
  spread_spilled_energy_cost = 0,
  average_unsupplied_energy_cost = 0,
  average_spilled_energy_cost = 0
)
```

Arguments

non_dispatchable_power	logical, default to FALSE
dispatchable_hydro_power	logical, default to FALSE
other_dispatchable_power	logical, default to FALSE
spread_unsupplied_energy_cost	numeric, default to 0
spread_spilled_energy_cost	numeric, default to 0
average_unsupplied_energy_cost	numeric, default to 0
average_spilled_energy_cost	numeric, default to 0

Value

a named list

Examples

```
nodalOptimizationOptions()
```

---

playlist *Get the playlist of an Antares study Antares API:* **OK**

Description

`getPlaylist` gives the identifier of the MC years which will be simulated in the Antares study, taking into account the potential use of a playlist which can skip some MC years

`setPlaylist` is a function which modifies the input file of an ANTARES study and set the playlist in order to simulate only the MC years given in input

Usage

```
getPlaylist(opts = antaresRead::simOptions())

setPlaylist(playlist, weights = NULL, opts = antaresRead::simOptions())
```

Arguments

opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()
playlist	vector of MC years identifier to be simulated can be a list (V8 compatibility) but not recommended
weights	data.table, 2 columns : mcYears and weights. Only with after antares V8

Value

- getPlaylist returns a vector of the identifier of the simulated MC year.
- setPlaylist does not return anything. It is used to modify the input of an Antares study.

Examples

```
## Not run:
setSimulationPath("PATH/TO/STUDY/")
# or
setSimulationPathAPI(
  host = "http://localhost:8080",
  study_id = "6f98a393-155d-450f-a581-8668dc355235",
  token = NULL,
  simulation = "input"
)

# augment number of MC years
updateGeneralSettings(nbyears = 10)

# Get the actual playlist
getPlaylist()
# [1] 2 4 6

# set a new playlist
setPlaylist(c(3, 5, 7))

## End(Not run)
```

---

propertiesLinkOptions *Properties for creating a link*

Description

Properties for creating a link

Usage

```
propertiesLinkOptions(
  hurdles_cost = FALSE,
  transmission_capacities = "enabled",
  asset_type = "ac",
  display_comments = TRUE,
  filter_synthesis = c("hourly", "daily", "weekly", "monthly", "annual"),
  filter_year_by_year = c("hourly", "daily", "weekly", "monthly", "annual")
)
```

Arguments

`hurdles_cost`	Logical, which is used to state whether (linear) transmission fees should be taken into account or not in economy and adequacy simulations
`transmission_capacities`	Character, one of `enabled`, `ignore` or `infinite`, which is used to state whether the capacities to consider are those indicated in 8760-hour arrays or if zero or infinite values should be used instead (actual values / set to zero / set to infinite)
`asset_type`	Character, one of `ac`, `dc`, `gas`, `virt` or `other`. Used to state whether the link is either an AC component (subject to Kirchhoff's laws), a DC component, or another type of asset.
`display_comments`	Logical, display comments or not.
`filter_synthesis`	Character, vector of time steps used in the output synthesis, among `hourly`, `daily`, `weekly`, `monthly`, and `annual`
`filter_year_by_year`	Character, vector of time steps used in the output year-by-year, among `hourly`, `daily`, `weekly`, `monthly`, and `annual`

Value

A named list that can be used in `createLink()`.

Examples

```
## Not run:
propertiesLinkOptions(
  hurdles_cost = TRUE,
  filter_synthesis=c("hourly","daily"),
  filter_year_by_year=c("weekly","monthly")
)

## End(Not run)
```

removeArea *Remove an area from an Antares study*

Description

Antares API: **OK**

Remove an area in an Antares study.

Usage

```
removeArea(name, opts = antaresRead::simOptions())
```

Arguments

name	An area name.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

See Also

createArea(), editArea()

Examples

```
## Not run:
removeArea("fictive_area")

## End(Not run)
```

removeBindingConstraint *Remove a Binding Constraint*

Description

Antares API: **OK**

Remove a binding constraint in an Antares study.

Usage

```
removeBindingConstraint(name, opts = antaresRead::simOptions())
```

Arguments

name	Name(s) of the binding constraint(s) to remove.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

See Also

createBindingConstraint() to create new binding constraints, editBindingConstraint() to edit existing binding constraints.

Examples

```
## Not run:
removeBindingConstraint("mybindingconstraint")

## End(Not run)
```

---

removeCluster *Remove a cluster*

Description

Antares API: **OK** (thermal clusters only)

Remove a cluster, thermal RES (renewable energy source) or short-term storage, and all its data.

Usage

```
removeCluster(
  area,
  cluster_name,
  add_prefix = TRUE,
  opts = antaresRead::simOptions()
)

removeClusterRES(
  area,
  cluster_name,
  add_prefix = TRUE,
  opts = antaresRead::simOptions()
)

removeClusterST(
  area,
  cluster_name,
```

```
  add_prefix = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where to create the cluster.
cluster_name	Name for the cluster, it will prefixed by area name, unless you set add_prefix = FALSE.
add_prefix	If TRUE (the default), cluster_name will be prefixed by area name.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

See Also

createCluster(), createClusterRES() or createClusterST() to create new clusters, editCluster() or editClusterRES() or editClusterST() to edit existing clusters.

Examples

```
## Not run:
createCluster(
  area = "fr",
  cluster_name = "fr_gas",
  group = "other",
  `marginal-cost` = 50
)

removeCluster(area = "fr", cluster_name = "fr_gas")

## End(Not run)
```

---

removeLink *Remove a link between two areas*

Description

Antares API: **OK**

Remove a link between two areas in an Antares study.

Usage

```
removeLink(from, to, opts = antaresRead::simOptions())
```

Arguments

from, to	The two areas linked together.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:
createLink(from = "myarea", to  = "myarea2")
removeLink(from = "myarea", to  = "myarea2")

## End(Not run)
```

---

replicate_missing_ts *Replicate a data.table as many times as needed to get the same number of time series between 2 data.tables*

Description

Replicate a data.table as many times as needed to get the same number of time series between 2 data.tables

Usage

```
replicate_missing_ts(xts, yts)
```

Arguments

xts	a data.table of time series type to replicate if necessary.
yts	a data.table of time series type to use as reference to match its number of time series.

Value

the data.table x replicated to match the number of time series of y.

rollback_to_previous_data *Rollback to previous hydro data if the data is not consistent*

Description

Rollback the data to previous one if the check is KO. For a given area, check if the data is consistent and rollback to previous data if the check is KO.

Usage

```
rollback_to_previous_data(
  area,
  prev_data,
  rollback_type,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where to execute the control and rollback the data.
prev_data	The original data to restore if necessary.
rollback_type	The file to restore and the control(s) to execute.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

Note

Function called only for an **Antares version >= 860**.

runSimulation *Run an Antares Simulation*

Description

Antares API: **OK**

Run an ANTARES study

Usage

```
runSimulation(
  name,
  mode = "economy",
  path_solver = getOption("antares.solver"),
  wait = TRUE,
  show_output_on_console = FALSE,
  parallel = TRUE,
  ...,
  opts = antaresRead::simOptions()
)
```

Arguments

`name`	Name of the simulation. In API mode, `name` will be used as `output_suffix` argument.
`mode`	Simulation mode, can take value "economy", "adequacy" or "draft".
`path_solver`	Character containing the Antares Solver path
`wait`	Logical, indicating whether the R interpreter should wait for the simulation to finish, or run it asynchronously.
`show_output_on_console`	Logical, indicating whether to capture the ANTARES log and show it on the R console.
`parallel`	Logical. If `TRUE` the ANTARES simulation will be run in parallel mode (Work only with ANTARES v6.0.0 or more). In that case, the number of cores used by the simulation is the one set in advanced_settings/simulation_cores (see ANTARES interface).
`...`	Additional arguments (API only), such as `nb_cpu`, `time_limit`, ... See API documentation for all available options.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

In API mode it return a `list` with either the job id in case of success of the command or details about the error produce. In non-API mode the function does not return anything, it is used to launch an ANTARES simulation.

`runTsGenerator` *Run Time-Series Generator*

Description

Antares API: **NO**

Usage

```
runTsGenerator(
  path_solver = getOption("antares.solver"),
  wait = TRUE,
  show_output_on_console = FALSE,
  opts = antaresRead::simOptions()
)
```

Arguments

path_solver	Character containing the Antares Solver path.
wait	Logical, indicating whether the R interpreter should wait for the simulation to finish, or run it asynchronously.
show_output_on_console	Logical, indicating whether to capture the ANTARES log and show it on the R console.
opts	List of simulation parameters returned by the function `antaresRead::setSimulationPath`.

Examples

```
## Not run:
library(antaresRead)
setSimulationPath(path = "path/to/study")

library(antaresEditObject)
runTsGenerator(
  path_solver = "path/to/antares-6.0-solver.exe",
  show_output_on_console = TRUE
)

## End(Not run)
```

---

scenario-builder *Read, create, update & deduplicate scenario builder*

Description

Antares API: **OK**

Read, create, update & deduplicate scenario builder.

Usage

```
scenarioBuilder(
  n_scenario,
  n_mc = NULL,
  areas = NULL,
  areas_rand = NULL,
```

```
  coef_hydro_levels = NULL,
  opts = antaresRead::simOptions()
)

readScenarioBuilder(
  ruleset = "Default Ruleset",
  as_matrix = TRUE,
  opts = antaresRead::simOptions()
)

updateScenarioBuilder(
  ldata,
  ruleset = "Default Ruleset",
  series = NULL,
  clusters_areas = NULL,
  links = NULL,
  opts = antaresRead::simOptions()
)

clearScenarioBuilder(
  ruleset = "Default Ruleset",
  opts = antaresRead::simOptions()
)

deduplicateScenarioBuilder(
  ruleset = "Default Ruleset",
  opts = antaresRead::simOptions()
)
```

Arguments

`n_scenario`	Number of scenario.
`n_mc`	Number of Monte-Carlo years.
`areas`	Areas to use in scenario builder, if `NULL` (default) all areas in Antares study are used.
`areas_rand`	Areas for which to use `"rand"`.
`coef_hydro_levels`	Hydro levels coefficients.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`
`ruleset`	Ruleset to read.
`as_matrix`	If `TRUE` (default) return a `matrix`, else a `list`.
`ldata`	A `matrix` obtained with `scenarioBuilder`, or a named list of matrices obtained with `scenarioBuilder`, names must be 'l', 'h', 'w', 's', 't', 'r', 'ntc' or 'hl', depending on the series to update.
`series`	Name(s) of the serie(s) to update if `ldata` is a single `matrix`.

clusters_areas A data.table with two columns area and cluster to identify area/cluster couple to update for thermal or renewable series. Default is to read clusters description and update all couples area/cluster.

links Links to use if series is "ntc". Either a simple vector with links described as "area01%area02 or a data.table with two columns from and to. Default is to read existing links and update them all.

Value

scenarioBuilder : a matrix

readScenarioBuilder : a list of matrix or list according to as_matrix parameters.

Note

series = "ntc" is only available with Antares >= 8.2.0. series = "hl" each value must be between 0 and 1.

Examples

```
## Not run:

library(antaresRead)
library(antaresEditObject)

# simulation path
setSimulationPath(
  path = "pat/to/simulation",
  simulation = "input"
)

# Create a scenario builder matrix
sbuilder <- scenarioBuilder(
  n_scenario = 51,
  n_mc = 2040,
  areas_rand = c("fr", "be")
)
sbuilder[, 1:6]
dim(sbuilder)

# Create a scenario builder matrix for hydro levels (use case 1)
sbuilder <- scenarioBuilder(
  n_mc = opts$parameters$general$nbyears,
  areas = c("fr", "be"),
  coef_hydro_levels = c(0.1, 0.9)
)

# Create a scenario builder matrix for hydro levels (use case 2)
sbuilder <- scenarioBuilder(
  n_mc = opts$parameters$general$nbyears,
  areas = c("fr", "be"),
  coef_hydro_levels = c(runif(opts$parameters$general$nbyears)
```

```
  , runif(opts$parameters$general$nbyears)
  )
)

# Read previous scenario builder
# in a matrix format
prev_sb <- readScenarioBuilder()


# Update scenario builder

# for load serie
updateScenarioBuilder(ldata = sbuilder, series = "load")

# equivalent as
updateScenarioBuilder(ldata = list(l = sbuilder))


# update several series

# same input
sbuilder
updateScenarioBuilder(
  ldata = sbuilder,
  series = c("load", "hydro", "solar")
)

# different input
updateScenarioBuilder(ldata = list(
  l = load_sb,
  h = hydro_sb,
  s = solar_sb
))

# Deduplicate scenario builder

deduplicateScenarioBuilder()

## End(Not run)
```

---

searchStudy *Search study in AntaREST*

Description

Search study in AntaREST

Usage

```
searchStudy(
```

```
  workspace = NULL,
  folder = NULL,
  name = NULL,
  ...,
  host = NULL,
  token = NULL
)
```

Arguments

workspace	Space in which to search for a study.
folder	Folder in which to search for a study.
name	Name for the study.
...	Other query parameters.
host	Host of AntaREST server API.
token	API personnal access token.

Value

a data.table with informations about studies on the server.

Examples

```
## Not run:

searchStudies(host = "http://localhost:8080")


## End(Not run)
```

---

setAPImode *Set API mode*

Description

Two modes are available when using the API:

- **async**: record all API calls, but nothing is sent to the server
- **sync**: send query to the API each time a function is used

Usage

```
setAPImode(mode = c("sync", "async"), opts = antaresRead::simOptions())
```

Arguments

mode	The mode you want to use.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:
# See vignette for complete documentation
vignette("api-variant-management")

# Usage :
setAPImode("sync")


## End(Not run)
```

`setSolverPath` *Set path to Antares Solver*

Description

Set path to Antares Solver

Usage

```
setSolverPath(path)
```

Arguments

`path`	(optional) Path to the solver (e.g. `antares-6.0-solver.exe` in \bin directory where Antares is installed). If missing, a window opens and lets the user choose the directory of the simulation interactively.

Examples

```
## Not run:

setSolverPath(path = "C:/antares/bin/antares-6.0-solver.exe")


## End(Not run)
```

storage_values_default

Short Term Storage Property List

Description

Short Term Storage Property List

Usage

```
storage_values_default()
```

Value

a named list

Examples

```
storage_values_default()
```

updateAdequacySettings

Update adequacy parameters of an Antares study

Description

Antares API: **OK**

Update adequacy parameters of an Antares study

Usage

```
updateAdequacySettings(
  include_adq_patch = NULL,
  set_to_null_ntc_from_physical_out_to_physical_in_for_first_step = NULL,
  set_to_null_ntc_between_physical_out_for_first_step = NULL,
  include_hurdle_cost_csr = NULL,
  check_csr_cost_function = NULL,
  enable_first_step = NULL,
  price_taking_order = NULL,
  threshold_initiate_curtailment_sharing_rule = NULL,
  threshold_display_local_matching_rule_violations = NULL,
  threshold_csr_variable_bounds_relaxation = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

include_adq_patch
: Logical. If TRUE, will run Adequacy Patch

set_to_null_ntc_from_physical_out_to_physical_in_for_first_step
: Logical. default to TRUE

set_to_null_ntc_between_physical_out_for_first_step
: Logical. default to TRUE

include_hurdle_cost_csr
: Logical. default to FALSE

check_csr_cost_function
: Logical. default to FALSE

enable_first_step
: Logical. default to TRUE

price_taking_order
: Character. can take values DENS (default value) and Load.

threshold_initiate_curtailment_sharing_rule
: Double. default to 0.0

threshold_display_local_matching_rule_violations
: Double. default to 0.0

threshold_csr_variable_bounds_relaxation
: Integer. default to 3

opts
: List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

updateAdequacySettings(
  include_adq_patch = TRUE,
  set_to_null_ntc_from_physical_out_to_physical_in_for_first_step = TRUE,
  set_to_null_ntc_between_physical_out_for_first_step = TRUE
)


## End(Not run)
```

`updateGeneralSettings` *Update general parameters of an Antares study*

Description

Antares API: **OK**

Update general parameters of an Antares study

Usage

```
updateGeneralSettings(
  mode = NULL,
  horizon = NULL,
  nbyears = NULL,
  simulation.start = NULL,
  simulation.end = NULL,
  january.1st = NULL,
  first.month.in.year = NULL,
  first.weekday = NULL,
  leapyear = NULL,
  year.by.year = NULL,
  derated = NULL,
  custom.ts.numbers = NULL,
  user.playlist = NULL,
  filtering = NULL,
  active.rules.scenario = NULL,
  generate = NULL,
  nbtimeseriesload = NULL,
  nbtimeserieshydro = NULL,
  nbtimeserieswind = NULL,
  nbtimeseriesthermal = NULL,
  nbtimeseriessolar = NULL,
  refreshtimeseries = NULL,
  intra.modal = NULL,
  inter.modal = NULL,
  refreshintervalload = NULL,
  refreshintervalhydro = NULL,
  refreshintervalwind = NULL,
  refreshintervalthermal = NULL,
  refreshintervalsolar = NULL,
  readonly = NULL,
  geographic.trimming = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

mode	Economy, Adequacy, Draft.
horizon	Reference year (static tag, not used in the calculations)
nbyears	Number of Monte-Carlo years that should be prepared for the simulation (not always the same as the Number of MC years actually simulated, see 'selection mode' below).
simulation.start	First day of the simulation (e.g. 8 for a simulation beginning on the second week of the first month of the year)
simulation.end	Last day of the simulation (e.g. 28 for a simulation ending on the fourth week of the first month of the year)
january.1st	First day of the year (Mon, Tue, etc.).
first.month.in.year	Actual month by which the Time-series begin (Jan to Dec, Oct to Sep, etc.)
first.weekday	In economy or adequacy simulations, indicates the frame (Mon- Sun, Sat-Fri, etc.) to use for the edition of weekly results.
leapyear	(TRUE/FALSE) indicates whether February has 28 or 29 days.
year.by.year	(False) No individual results will be printed out, (True) For each simulated year, detailed results will be printed out in an individual directory7 : Study_name/OUTPUT/simu_tag/Economy /mc-i-number
derated	See Antares General Reference Guide.
custom.ts.numbers	See Antares General Reference Guide.
user.playlist	See Antares General Reference Guide.
filtering	See Antares General Reference Guide.
active.rules.scenario	See Antares General Reference Guide.
generate	See Antares General Reference Guide.
nbtimeseriesload	See Antares General Reference Guide.
nbtimeserieshydro	See Antares General Reference Guide.
nbtimeserieswind	See Antares General Reference Guide.
nbtimeseriesthermal	See Antares General Reference Guide.
nbtimeseriessolar	See Antares General Reference Guide.
refreshtimeseries	See Antares General Reference Guide.
intra.modal	See Antares General Reference Guide.
inter.modal	See Antares General Reference Guide.

refreshintervalload
: See Antares General Reference Guide.

refreshintervalhydro
: See Antares General Reference Guide.

refreshintervalwind
: See Antares General Reference Guide.

refreshintervalthermal
: See Antares General Reference Guide.

refreshintervalsolar
: See Antares General Reference Guide.

readonly
: See Antares General Reference Guide.

geographic.trimming
: logical indicates whether to store the results for all time spans (FALSE) or for custom time spans (TRUE)

opts
: List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

# Update number of Monte-Carlo years
updateGeneralSettings(nbyears = 42)

# Use a vector to update a parameter that
# can take multiple values
updateGeneralSettings(generate = c("thermal", "hydro"))


## End(Not run)
```

---

updateInputSettings *Update input parameters of an Antares study*

Description

Antares API: **OK**

Update input parameters of an Antares study

Usage

```
updateInputSettings(import, opts = antaresRead::simOptions())
```

Arguments

import	Series to import.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

updateInputSettings(import = c("thermal"))
updateInputSettings(import = c("hydro", "thermal"))


## End(Not run)
```

---

updateOptimizationSettings

Update optimization parameters of an Antares study

Description

Antares API: **OK**

Update optimization parameters and other preferences of an Antares study

Usage

```
updateOptimizationSettings(
  simplex.range = NULL,
  transmission.capacities = NULL,
  include.constraints = NULL,
  include.hurdlecosts = NULL,
  include.tc.min.stable.power = NULL,
  include.tc.min.up.down.time = NULL,
  include.dayahead = NULL,
  include.strategicreserve = NULL,
  include.spinningreserve = NULL,
  include.primaryreserve = NULL,
  include.exportmps = NULL,
  power.fluctuations = NULL,
  shedding.strategy = NULL,
  shedding.policy = NULL,
  unit.commitment.mode = NULL,
  number.of.cores.mode = NULL,
  renewable.generation.modelling = NULL,
```

```
  day.ahead.reserve.management = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

`simplex.range`	week or day
`transmission.capacities`	true, false or infinite (since v8.4 can also take : local-values, null-for-all-links, infinite-for-all-links, null-for-physical-links, infinite-for-physical-links)
`include.constraints`	true or false
`include.hurdlecosts`	true or false
`include.tc.min.stable.power`	true or false
`include.tc.min.up.down.time`	true or false
`include.dayahead`	true or false
`include.strategicreserve`	true or false
`include.spinningreserve`	true or false
`include.primaryreserve`	true or false
`include.exportmps`	true or false (since v8.3.2 can take also : none, optim-1, optim-2, both-optims)
`power.fluctuations`	free modulations, minimize excursions or minimize ramping
`shedding.strategy`	share margins
`shedding.policy`	shave peaks or minimize duration
`unit.commitment.mode`	fast or accurate
`number.of.cores.mode`	minimum, low, medium, high or maximum
`renewable.generation.modelling`	aggregated or clusters
`day.ahead.reserve.management`	global
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

updateOptimizationSettings(
  simplex.range = "week",
  power.fluctuations = "minimize ramping"
)


## End(Not run)
```

`updateOutputSettings` *Update output parameters of an Antares study*

Description

Antares API: **OK**

Update output parameters of an Antares study

Usage

```
updateOutputSettings(
  synthesis = NULL,
  storenewset = NULL,
  archives = NULL,
  result.format = NULL,
  opts = antaresRead::simOptions()
)
```

Arguments

`synthesis`	Logical. If TRUE, synthetic results will be stored in a directory Study_name/OUTPUT/simu_tag/Econom all. If FALSE, No general synthesis will be printed out.
`storenewset`	Logical. See Antares General Reference Guide.
`archives`	Character vector. Series to archive.
`result.format`	Character. Output format (txt-files or zip).
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

updateOutputSettings(
  synthesis = TRUE,
  storenewset = FALSE,
  archives = c("load", "wind"),
  result.format = "zip"
)


## End(Not run)
```

`variant` *Create a study's variant*

Description

API: create a new variant for a given study or use a pre-existing one.

Usage

```
createVariant(name, opts = antaresRead::simOptions())

useVariant(name, variant_id = NULL, opts = antaresRead::simOptions())
```

Arguments

`name`	Name for the variant to create or the name of an existent variant.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`
`variant_id`	ID of the variant to use, if specified `name` is ignored.

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:
# See vignette for complete documentation
vignette("api-variant-management")

## End(Not run)
```

variant-commands *Get API commands generated*

Description

Get API commands generated

Usage

```
getVariantCommands(
  last = NULL,
  actions = NULL,
  opts = antaresRead::simOptions()
)

writeVariantCommands(
  path,
  last = NULL,
  actions = NULL,
  ...,
  opts = antaresRead::simOptions()
)
```

Arguments

last	Return the last command generated if TRUE, or a numeric for returning a specified number of commands. Default is to return all commands.
actions	A character vector of actions to return.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()
path	Path to the JSON file to write on disk.
...	Additional arguments passed to jsonlite::write_json()

Value

a list of commands to edit a variant

write-ini *Write configuration options in file or API*

Description

Write configuration options in file or API

Usage

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

writeIniFile(listData, pathIni, overwrite = FALSE)

writeIniAPI(listData, pathIni, opts)
```

Arguments

listData	list, modified list obtained by antaresRead:::readIniFile.
pathIni	Character, Path to ini file.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()
...	Additional arguments.
default_ext	Default extension used for config files.
overwrite	logical, should file be overwritten if already exist?

Examples

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

## End(Not run)
```

---

writeEconomicOptions *Write Economic Options*

Description

Antares API: **OK**

This function allows to create or edit economic options. Areas/options present in the input dataframe are edited, while all other values are left unchanged.

Usage

```
writeEconomicOptions(x, opts = antaresRead::simOptions())
```

Arguments

x	A dataframe. Must contain an area column listing some (but not necessarily all) areas of the study. Can contain up to 7 other columns among: average_unsupplied_energy_cost, spread_unsupplied_energy_cost, average_spilled_energy_cost, spread_spilled_energy_cost (numeric columns), non_dispatchable_power, dispatchable_hydro_power and other_dispatchable_power (logical columns).
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath

Examples

```
## Not run:

library(antaresRead)

# Set simulation path
setSimulationPath(path = "PATH/TO/SIMULATION", simulation = "input")

# Write some economic options for areas a, b and c
writeEconomicOptions(data.frame(
  area = c("a", "b", "c"),
  dispatchable_hydro_power = c(TRUE, FALSE, FALSE),
  spread_unsupplied_energy_cost = c(0.03, 0.024, 0.01),
  average_spilled_energy_cost = c(10, 8, 8),
  stringsAsFactors = FALSE
))


## End(Not run)
```

---

writeHydroValues *Write Hydro Values*

Description

Antares API: **OK**

Write waterValues, reservoirLevels, maxpower, inflowPattern and creditModulations data for a given area.

Usage

```
writeHydroValues(
  area,
  type,
  data,
  overwrite = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where to add the values.
type	Type of hydro file, it can be "waterValues", "reservoir", "maxpower", "inflowPattern" or "creditmodulations".
data	The data must have specific dimension depending on the type of file : • waterValues: a 365x101 numeric matrix: marginal values for the stored energy based on date (365 days) and on the reservoir level (101 round percentage values ranging from 0% to 100%). OR a 3-column matrix with 365x101 rows. In this latter case the 3 columns must be 'date', 'level' and 'value' (in this order), and the rows must be sorted by: ascending day, ascending level. • reservoir: a 365x3 numeric matrix. The columns contains respectively the levels min, avg and max. • maxpower: a 365x4 numeric matrix. • inflowPattern: a 365x1 numeric matrix. • creditmodulations: a 2x101 numeric matrix.
overwrite	Logical. Overwrite the values if a file already exists.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

Warning

For an **Antares version >= 860**, control of data consistency between mingen.txt and maxpower_<area>.txt can be executed.

This control depends on the values you find in hydro.ini file.

Examples

```
## Not run:

writeHydroValues("fictive_area", type = "inflowPattern", data = matrix(rep(0, 365*1), nrow = 365))


## End(Not run)
```

---

writeIniHydro *Edit hydro.ini values*

Description

Antares API: **OK**

For a given area, write its data in the hydro.ini file.

Usage

```
writeIniHydro(area, params, mode = "other", opts = antaresRead::simOptions())
```

Arguments

area	The area where to edit the values.
params	The list data must have specific names and specific types : • follow load : logical or NULL • use heuristic : logical or NULL • use water : logical or NULL • hard bounds : logical or NULL • use leeway : logical or NULL • power to level : logical or NULL • reservoir : logical or NULL • inter-daily-breakdown : numeric, integer or NULL • intra-daily-modulation : numeric, integer or NULL • inter-monthly-breakdown : numeric, integer or NULL • leeway low : numeric, integer or NULL • leeway up : numeric, integer or NULL • pumping efficiency : numeric, integer or NULL • initialize reservoir date : numeric, integer or NULL • reservoir capacity : numeric, integer or NULL
mode	Execution mode. Useful when you create a new area or remove an existing area to avoid control on hydro data.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

Warning

For an **Antares version >= 860**, control of data consistency between mingen.txt and mod.txt can be executed.

For an **Antares version >= 860**, control of data consistency between mingen.txt and maxpower_<area>.txt can be executed.

These controls depend on the values you find in hydro.ini file.

Examples

```
## Not run:
opts <- setSimulationPath(studypath, simulation = "input")
createArea("fictive_area")
writeIniHydro(area = "fictive_area"
, params = list("leeway low" = 2.5, "leeway up" = 25))


## End(Not run)
```

writeInputTS *Write input time series*

Description

Antares API: **OK**

This function writes input time series in an Antares project.

Usage

```
writeInputTS(
  data,
  type = c("load", "hydroROR", "hydroSTOR", "mingen", "wind", "solar", "tsLink"),
  area = NULL,
  link = NULL,
  overwrite = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

data	A 8760*N matrix of hourly time series, except when type is "hydroSTOR". In this latter case "hydroSTOR" data must have either be 365 rows (Antares v7) or 12 rows (v6 and earlier).
type	Serie to write: "load", "hydroROR", "hydroSTOR", "wind", "solar", "tsLink" or "mingen". If type == "mingen", "antaresVersion" should be >= 860. Refers to note section below.
area	The area where to write the input time series.
link	Link for which writing transmission capacities time series, must like "area01%area02" or "area01 - area02" or c("area01", "area02").
overwrite	Logical. Overwrite the values if a file already exists.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Value

An updated list containing various information about the simulation.

Warning

You cannot use area and link arguments at the same time.

For an **Antares version >= 860**, control of data consistency between mingen.txt and mod.txt can be executed.

These controls depend on the values you find in hydro.ini file.

Note

For an **Antares version >= 860**, the `mingen.txt` file is created.

The `mingen.txt` file can be created under two conditions:

- The number of columns must be equal to either `1` or the number in `mod.txt`
- If the `mod.txt` file is empty or has one column, then there is no dimension constraint

Examples

```
## Not run:

# Write solar time series
writeInputTS(
  area = "fictive_area",
  type = "solar",
  data = matrix(rep(4, 8760*2), nrow = 8760)
)


## End(Not run)
```

`writeMiscGen` *Write Misc Gen data*

Description

Antares API: **OK**

Usage

```
writeMiscGen(data, area, opts = antaresRead::simOptions())
```

Arguments

`data`	Data to write.
`area`	Name of the area for which to write data.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`

Value

An updated list containing various information about the simulation.

Examples

```
## Not run:

writeMiscGen(matrix(data = c(rep(0, 8760 * 7), rep(-100000, 8760)), ncol = 8), "area1")


## End(Not run)
```

writeOutputValues *Write output value for Antares*

Description

Antares API: **NO**

This function write all output values for an Antares study.

Usage

```
writeOutputValues(data, opts = NULL)
```

Arguments

data	obtain with readAntares
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath()

Examples

```
## Not run:

library(antaresRead)
library(data.table)
opts <- setSimulationPath("my_study")
data <- readAntares(links = "all", areas = "all", clusters = "all")
writeOutputValues(data)


## End(Not run)
```

writeSeriesPrepro *Write prepro data*

Description

Antares API: **NO**

This function allows to write load, wind and solar prepro data. Using `character(0)` allows to erase data (cf Examples).

Usage

```
writeSeriesPrepro(
  area,
  type = c("load", "wind", "solar"),
  coefficients = NULL,
  daily_profile = NULL,
  translation = NULL,
  conversion = NULL,
  overwrite = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

area	The area where to write prepro data.
type	Type of data to write : "load", "wind" or "solar".
coefficients	A 12*6 matrix of monthly values for the primary parameters alpha, beta, gamma, delta, theta and mu.
daily_profile	A 24*12 matrix of hourly / monthly coefficients K(hm) that are used to modulate the values of the stationary stochastic process by which the actual process is approximated.
translation	A vector of length 8760 (or 8760*1 matrix) to add to the time-series generated, prior or after scaling.
conversion	A 2*N matrix (with 1<=N<=50) that is used to turn the initial time-series produced by the generators into final data. See Antares General Reference Guide.
overwrite	Logical. Overwrite the values if a file already exists.
opts	List of simulation parameters returned by the function antaresRead::setSimulationPath().

Examples

```
## Not run:

writeSeriesPrepro("fictive_area", type = "solar", daily_profile = matrix(rep(1, 24*12), nrow = 24))

# Erase daily profile data:
writeSeriesPrepro("fictive_area", type = "solar", daily_profile = character(0))


## End(Not run)
```

`writeWaterValues` *Write water values*

Description

Antares API: **OK**

Write water values for a given area.

Usage

```
writeWaterValues(
  area,
  data = NULL,
  overwrite = TRUE,
  opts = antaresRead::simOptions()
)
```

Arguments

`area`	The area where to add the water values.
`data`	A 365x101 numeric matrix: table of marginal values for the stored energy, which depends on the date (365 days) and on the reservoir level (101 round percentage values ranging from 0% to 100%). OR a 3-column matrix with 365x101 rows. In this latter case the 3 columns must be 'date', 'level' and 'value' (in this order), and the rows must be sorted by: ascending day, ascending level.
`overwrite`	Logical. Overwrite the values if a file already exists.
`opts`	List of simulation parameters returned by the function `antaresRead::setSimulationPath()`.

Examples

```
## Not run:

writeWaterValues("fictive_area", data = matrix(rep(0, 365*101), nrow = 365))


## End(Not run)
```

Index