Package 'energyRt'

November 3, 2024

Type Package

Title Energy systems modeling toolbox in R, development version

Version 0.50.7.9001

Date 2024-11-03

URL https://energyRt.org, https://github.com/energyRt/energyRt

BugReports https://github.com/energyRt/energyRt/issues

Description Energy systems modeling toolbox in R

Depends R (>= 4.3)

Imports methods, grid, generics, data.table, DBI, RSQLite, tibble, tidyr, dplyr, rlang, stringr, lubridate, purrr, arrow, progressr, tictoc, cli, zoo, registry, options (>= 0.2.0), glue, plyr, grDevices, utils

Suggests pak, gdxtools, knitr, rmarkdown, sf, yaml, ggplot2, testthat (>= 3.0.0), devtools

SystemRequirements Julia (>= 1.6) with JuMP or Python (>= 3.7) with Pyomo or GAMS (>= 24.9) or GLPK (>= 4.0)

License AGPL (>= 3)

RoxygenNote 7.3.2

VignetteBuilder knitr

LazyData true

Encoding UTF-8

Collate 'defaults.R' 'docs.R' 'options.R' 'generics.R' 'class-calendar.R' 'class-commodity.R' 'class-demand.R' 'class-supply.R' 'class-technology.R' 'class-storage.R' 'class-trade.R' 'class-import.R' 'class-export.R' 'class-weather.R' 'class-tax.R' 'class-subsidy.R' 'class-constraint.R' 'class-costs.R' 'class-horizon.R' 'class-config.R' 'class-settings.R' 'class-repository.R' 'class-model.R' 'class-parameter.R' 'class-modInp.R' 'class-modOut.R' 'class-scenario.R' 'add2set.R' 'arrow.R' 'utils.R' 'convert.R' 'data2slots.R' 'draw.R' 'energyRt-package.R' 'get_data.R' 'install.R' 'interpolate.R' 'interpolate2.R' 'interpolate_new.R' 'interpolation_funs.R' 'obj2modInp.R' 'print.R' 'read.R' 'registry.R' 'solve.R' 'start_msg.R' 'timeslices.R' 'write.R' 'write_gams.R' 'write_glpk.R' 'write_jump.R' 'write_pyomo.R'

Config/testthat/edition 3

Config/testthat/parallel false

Roxygen list(markdown = TRUE)

NeedsCompilation no

Author Oleg Lugovoy [aut, cre] (<https://orcid.org/0000-0001-5546-9875>),
Vladimir Potashnikov [aut],
Tarun Sharma [ctb] (<https://orcid.org/0000-0003-3636-2858>)

Maintainer Oleg Lugovoy <olugovoy@optimalsolution.dev>

Contents

energyRt-package *energyRt: Energy systems modeling toolbox in R, development version*

Description

Energy systems modeling toolbox in R

Author(s)

Maintainer: Oleg Lugovoy <olugovoy@optimalsolution.dev> (ORCID)

Authors:

- Vladimir Potashnikov <potashnikov.vu@gmail.com>

Other contributors:

- Tarun Sharma <tarunsharma@ms.iitr.ac.in> (ORCID) [contributor]

See Also

Useful links:

- https://energyRt.org
- https://github.com/energyRt/energyRt
- Report bugs at https://github.com/energyRt/energyRt/issues

add,repository-method *Add an object to the model's repository*

Description

Add an object to the model's repository

Usage

```
## S4 method for signature 'repository'
add(obj, ..., overwrite = FALSE)

## S4 method for signature 'model'
add(obj, ..., overwrite = FALSE, repo_name = NULL)
```

Arguments

obj	model object
...	model elements, allowed classes: ...
overwrite	logical, if TRUE, objects with the same name will be overwritten, error will be reported if FALSE
repo_name	character, optional name of a (sub-)repository to add the object.

Value

model object with added elements to the repository

check_name *Check validity of object's names used in sets*

Description

Check validity of object's names used in sets

Usage

```
check_name(x)
```

Arguments

x	character, name of an object of energyRt

Value

logical, TRUE if the name is valid.

Examples

```
check_name("name")
check_name("1name")
check_name("name1")
check_name("name_1")
check_name("name_1!")
```

class-calendar *An S4 class to represent sub-annual time resolution structure.*

Description

Sub-annual time resolution is represented by nested, named time-frames and time-slices.

Slots

name character. Name of the calendar object. Use to distingush between different structures and subsets of time-slices. The name is used to propose default folder names for the model/scenario scripts to separate solutions of the same scenario with different calendar objects.

desc character. Description of the calendar object, for own references.

timeframes list. Named list of nested sub-annual levels with vectors of individual elements. The top level of the list is the highest level of the calendar, e.g., "ANNUAL". The lowest level is the smallest time-slice, e.g., "MONTH". "ANNUAL" is the default (hardwired) top level of the calendar. All other levels are optional, and create nested sub-annual levels of time-slices. The minimum number of time-slices in a timeframe is two (except for the top level).

year_fraction numeric. The fraction of a year covered by the calendar, e.g. 1 for annual calendar (default), 0.5 for semi-annual, 0.25 for quarterly, etc. Currently must be specified manually for subset calendars to validate the sum of the shares.

`timetable` data.frame. Data frame with levels of timeframes in the named columns, and number of rows equal to the total number of time-slices on the lowest level. Every timeframe is a set of time-slices ("slices") - a named fragment of time with a year-share. Timeframes have nested structure where every slice serves as a parent for the lower level of time-slices (children). The first column is the name of the time-slice, the rest of the columns are the names of the timeframes. The values are the share of the year covered by the time-slice. The sum of the shares in every timeframe should be equal to 1. `weight` is an optional column with the weight of the time-slice in the year, used for sumpled/subset selection of the time-slices.

`slice_share` data.frame. Auto-calculated from the `timetable` two column data.frame with slices from all levels with their individual share in a year. The first column is the name of the time-slice, the second column is the share of the year covered by the time-slice.

`default_timeframe` character. The name of the default level of the time-slices used in the model. If not specified, the lowest level of the timeframes is used as the default timeframe.

`timeframe_rank` character. Auto-calculated from the `timetable` and `timeframes` slots named character vector with ranks of the timeframes. The rank is used to determine the order of the timeframes in the calendar.

`slices_in_frame` integer. Auto-calculated from the `timetable` Number of time-slices in every timeframe.

`slice_family` data.frame. Auto-calculated from the `timetable` data.frame mapping "parent" to "child" slices in two nearest timeframes in the nested hierarchy. The first column is the name of the parent time-slice, the second column is the name of the child time-slice.

`slice_ancestry` data.frame. Auto-calculated from the `timetable` data.frame mapping "child", "grandchild", etc. slices to the "parent" and "grandparent" time-slices in the full hierarchy. The first column is the name of the (grand-) child time-slice, the second column is the name of the (grand-) parent time-slice.

`next_in_timeframe` data.frame. Auto-calculated from the `timetable` data.frame mapping chronological sequence between time-slices in the same timeframe. The first column is the name of the time-slice, the second column is the name of the next time-slice in the same timeframe.

`next_in_year` data.frame. Auto-calculated from the `timetable` data.frame mapping chronological sequence between time-slices in the same timeframe through the whole year. The first column is the name of the time-slice, the second column is the name of the next time-slice in the same timeframe.

`misc` list. Any additional data or information to store in the object.

`class-commodity` *An S4 class to represent a commodity*

Description

A commodity is a good or service that is produced and consumed in the model. The commodity class is used to store information about the commodity. All processes in the model operate on commodities, i.e. they either generate, produce, consume, transform, store, or transport commodities. The creation of a commodity object is done with the `newCommodity` function.

Slots

`name` character. Name of the commodity.

`desc` character. Optional description of the commodity for reference.

`limtype` factor or character. The limit type of the commodity in balance equation, "LO", "UP", or "FX". "LO" by default, meaning that the level of commodity in the model is restricted with the lower bound, excess is allowed. "UP" means that the level of commodity cannot exceed the upper bound. "FX" means that total commudity supply and demand are equal, no excess or deficit is allowed.

`timeframe` character. The default time-frame this commodity operates in the model. The lowest timeframe in the model is used by default.

`unit` character. The main unit of the commodity used in the model.

`emis` data.frame. Emissions factors related to the commodity consumption (if "combustion" parameter of a technology which consumes the commodity is > 0).

comm character. Name of the emitted commodity.

unit character. Unit of the emission factor.

emis numeric. Emission factor, emissions released per unit of the consumed commodity.

`agg` data.frame. Used to define an aggregation of several commodities into the `name` commodity.

comm character. Name of a commodity being aggregated.

unit character. Unit of the commodity being aggregated.

agg numeric. weight of the commodity in the aggregation, must be set for all aggregated commodities.

`misc` list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the commodity data, or other metadata.

`class-config` *An S4 class to represent default model configuration.*

Description

Config class is used to represent the default model configuration. It is stored in the model object and is used to initialize the scenario settings.

Slots

`name` character. Name of the configuration object for own references, also can be used in functions to distinguish between different model or scenario instances.

`desc` character. Description of the configuration object for own references.

`region` character. Coma separated string of all region names in the model. All regions used in the model-objects should be listed here.

`calendar` calendar. Calendar object with the model time parameters.

`horizon` horizon. Horizon object with the model time parameters. The horizon defines the planning period and intervals of the model.

`discount` data.frame. Discount rates, can be assigned by region and year.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

discount numeric. Discount rate. Default is 0.05. The discount rate is used to calculate the present value of future costs and benefits.

`discountFirstYear` logical. If TRUE, the discounting starts from the beginning of the year. If FALSE, the discounting starts from the end of the first year, i.e., the first year is not discounted.

`optimizeRetirement` logical. Incidates if the retirement of capacities of the model objects should be optimized. Also requires the same parameter in the classes with capacitu, such as `technology`, `storage`, `trade` to be set to TRUE to be effective for a specific object.

`defVal` data.frame. Default values of model parameters. The data frame with the default values for every parameter in the model, used to fill the missing values in the model objects. The values are used in the interpolation step. The data is stored in `energyRt:::.defVal` object and can be overwritten by the user and supplied to the model as a parameter.

`interpolation` data.frame. Default interpolation rules for every parameter in the model. The data frame with the default interpolation rules is stored in `energyRt:::.defInt` object and can be overwritten by the user and supplied to the model as a parameter.

`debug` data.frame. Artificial (dummy or sluck) variables to debug model infeasibility. Can be specified by commodities, regions, years, and slices.

`misc` list. Any additional data or information to store in the object.

See Also

Other class config settings scenario model: `class-settings`

`class-constraint` *An S4 class to represent a custom constraint.*

Description

Class `constraint` is used to define custom constraints in the optimization problem. **[Experimental]**

Class `summand` stores information about linear terms (a multiplier and a variable) in the `lhs` of the constraint class. It is auto-created by `newConstraint` function and is not intended to be used directly by the user.

Details

Custom constraints extend the functionality of the model by adding user-defined constraints to the optimization problem. If the predefined constraints are not sufficient to describe the problem, custom constraints can be used to add linear equlity or inequality constraints to define additional relationships between the variables. In many cases this can be done without writing constraints in the GAMS, Julia/JuMP, Python/Pyomo, or GLPK-MathProg languages by using the `constrant` class and the `newConstraint` function. To define a custom constraint with the `newConstraint` function, the user needs to specify the name of the constraint, the type of the relation (equality, less than or equal, greater than or equal), the left-hand side (LHS) terms of the statement, and the right-hand side (RHS) value. The dimension of the constraint is set by the `for.each` parameter. The 'lhs' terms are defined as a list of linear terms (summands). Each summand consists of a variable, a multiplier, and a set of sets for which the summand is defined.

Slots

`name` character. Name of the constraint object, used in sets.

`desc` character. Description of the constraint.

`eq` character. Type of the relation ('==' default, '<=', '>=').

`for.each` list. List with sets for combination of which the constraint is created.

`rhs` data.frame. Named list or data frame with numeric values for each constraint. The dimensions of the data frame should match the dimensions of the sets in the `for.each` slot.

`defVal` numeric. The default value for the rhs. It is recommended to set the default value for the rhs of every constraint to avoid unexpected behavior. If not specified, the default value is 0, and the warning is issued.

`interpolation` character. Interpolation rule for the constraint. Recognized values, any combination of "back", "inter", "forth", e.g., "back.inter" or "forth.inter", indicating the direction of interpolation. The default value is "inter", meaning that the interpolation is done for years between the specified values. The "back" and "forth" values induce backward and forward interpolation of the `rhs` values, respectively.

`lhs` list. List of summands for the left-hand-side of the equation. This slot is created automatically from all named of unnamed lists passed to the `newConstraint` function, except for the named arguments.

`misc` list. Any additional information or parameters to store in the constraint object.

`desc` character. Description of the linear term.

`variable` character. Name of the variable.

`for.sum` list. List of sets for which the summand will be created.

`mult` data.frame. Multiplying coefficients to the variable for each set in the `for.sum` slot.

`defVal` numeric. Default value for the summand.

`misc` list. Additional information.

See Also

Other class constraint policy: `class-costs`, `newConstraint()`, `newCosts()`, `newSubsidy()`, `newTax()`, `subsidy-class`, `tax-class`

Other class constraint: `newConstraint()`

`class-costs` *An S4 class to add costs to objective function*

Description

Costs object is used to define additional costs to add to the model's objective function.

Slots

`name` character. Name of the cost object, used in sets.

`desc` character. Description of the cost object for own references.

`variable` character. Name of the variable included in the costs-constraint.

`subset` data.frame. Named list or data frame with set-values for each dimension of the variable. This slot subsets the variable to the specified set values.

`mult` data.frame. Named list or data frame with numeric values for the variable included in the costs-constraint. A constant or a data frame with the same dimensions as the subseted variable.

`misc` list. Additional information.

See Also

Other class constraint policy: `class-constraint`, `newConstraint()`, `newCosts()`, `newSubsidy()`, `newTax()`, `subsidy-class`, `tax-class`

`class-demand` *An S4 class to declare a demand in the model*

Description

An S4 class to declare a demand in the model

Slots

`name` character. Name of the demand.

`desc` character. Optional description of the demand for reference.

`commodity` character. Name of the commodity for which the demand will be specified.

`unit` character. Optional unit of the commodity.

`dem` data.frame. Specification of the demand.

- **region** character. Name of region for the demand value. NA for every region.
- **year** integer. Year of the demand. NA for every year.
- **slice** character. Name of the slice for the demand value. NA for every slice.
- **dem** numeric. Value of the demand.

`region` character. Optional name of region to narrow the specification of the demand in the case of used NAs. Error will be returned if specified regions in `@dem` are not mensioned in the `@region` slot (if the slot is not empty).

`misc` list. Optional list of additional information.

class-export *An S4 class to represent commodity export to the rest of the world.*

Description

Export object represent commodity export to the Rest of the World (RoW).

Details

`export` is a type of process that adds an "external" source to a commodity to the model. The Rest of the World (RoW) is not modeled explicitly, `export` and `import` objects define and control the exchange with the RoW. The operation of the export object is similar to the `demand` objects, the two different classes are used to distinguish domestic and external sources of final consumption. The export is controlled by the `exp` data frame, which specifies bounds and fixed values for the export of the export flow. The `exp.fx` column is used to specify fixed values of the export flow, making the export flow exogenous. The `exp.lo` and `exp.up` columns are used to specify lower and upper bounds of the export flow, making the export flow endogenous. The `price` column is used to specify the exogenous price for the export commodity. The `reserve` slot is used to set limits on the total export over the model horizon.

Slots

`name` character. Name of the export object, used in sets.

`desc` character. Description of the export object.

`commodity` character. Name of the exported commodity.

`unit` character. Unit of the exported commodity.

`reserve` numeric. Total accumulated limit through the model horizon.

`exp` data.frame. Export parameters.

- **region** character. Region name to apply the parameter; use NA to apply to all regions.
- **year** integer. Year to apply the parameter; use NA to apply to all years.
- **slice** character. Time slice to apply the parameter; use NA to apply to all slices.
- **exp.lo** numeric. Export lower bound.
- **exp.up** numeric. Export upper bound.
- **exp.fx** numeric. Fixed export volume, ignored if NA. This parameter overrides `exp.lo` and `exp.up`.

`misc` list. Additional information.

class-horizon *An S4 class to represent model/scenario planning horizon with intervals (year-steps)*

Description

An S4 class to represent model/scenario planning horizon with intervals (year-steps)

Slots

`name` character. Name of the horizon object. Used to distinguish between different horizons in the model or scenario, including the automatic creation of the folder name for the model/scenario scripts.

`desc` character. Description of the horizon object, for own references.

`period` integer. A planning period defined as a sequence of years (arranged, without gaps) of the model planning (e.g. optimization) window. Data with years before or after the planning `period` can present in the model-objects and will be taken into account during interpolation of the model parameters.

`intervals` data.frame. Data frame with three columns, representing start, middle, and the end year of every interval. The first column is the start year of the interval, the second column is the middle year of the interval, the third column is the end year of the interval.

`class-import` *An S4 class to represent commodity import from the rest of the world.*

Description

Use `newImport` to create a new `import` object.

Details

Constructor for import object.

Import object adds an "external" source of commodity to the model. The RoW is not modeled explicitly as a region, `export` and `import` objects define and control the exchange with the RoW. The operation is similar to the `demand` object, but the two ideas distinguishes between internal and external final consumption. This exchange can be exogenously defined (`imp.fx`) or optimized by the model within the given limits (`imp.lo`, `imp.up`). The `price` column is used to define the price of the imported commodity. "Reserve" sets the total amount that can be imported over the model horizon.

Slots

`name` character. Name of the import object, used in sets.

`desc` character. Description of the import object.

`commodity` character. Name of the imported commodity.

`unit` character. Unit of the imported commodity.

`reserve` numeric. Total accumulated limit through the model horizon.

`imp` data.frame. Import parameters.

- **region** character. Region name to apply the parameter; use NA to apply to all regions.
- **year** integer. Year to apply the parameter; use NA to apply to all years.
- **slice** character. Time slice to apply the parameter; use NA to apply to all slices.
- **imp.lo** numeric. Lower bound on the import volume.
- **imp.up** numeric. Upper bound on the import volume.
- **imp.fx** numeric. Fixed import volume, ignored if NA. This parameter overrides `imp.lo` and `imp.up`.

`misc` list. Additional information.

class-model *An S4 class to represent model*

Description

An S4 class to represent model

Slots

`name` character. Name of the model object, for reference, also used in scenario path-functions.

`desc` character. Description of the model object, for own references.

`data` repository or list. A named list of model objects to interpolate and pass to the solver. Use the `repository` class to add objects to the model, or a list of model objects directly.

`config` config. Configuration object with the default model settings.

`misc` list. Any additional data or information to store in the object.

class-modInp *An S4 class to represent model input*

Description

`modInp` class is used to store interpolated date for the model input parameters. It includes all the model sets, mappings, and parameters, interpolated to the scenario's calendar and horizon. The class is automatically created during the interpolation step and is not intended to be created by users.

Slots

`set` list. named list of character vectors with sets used in the model.

`parameters` list. named list of `parameter` objects with three tipes - set: detailed description of the set - parameter: all model parameters, interpolated for every milestone year in the model horizon if applicable. Names of parameters start with 'p'

- mapping sets: auxiliary, automatically created sets used to narrow the dimension of variables and constraints

`gams.equation` list. named list of custom constraints added to the model from the `constraint` class. The name of the slot is outdated and will be changed in the future.

`costs.equation` list. named list of custom costs added to the model's objective from the `costs` class. The name of the slot is outdated and will be changed in the future.

`misc` list. Any additional information or data to store in the object. Also used to store paths to the data files of objects stored on the disk.

class-modOut *An S4 class to store results of a solved scenario*

Description

The class is a part of the scenario object and stores the results of a solved scenario. It is not intended to be used as a standalone object.

Slots

`sets` list. Named list of sets used in the model.

`variables` list. Named list of data frames with variables imported from solved scenario.

`stage` character. Indication of the solution status of the model/scenario.

`solutionLogs` data.frame. Data frame with the model solution logs.

`misc` list. Any additional information or data to store in the object.

class-parameter *An S4 class to specify the model set or parameter*

Description

Class `parameter` is used to represent the model set or parameter. All parameters and sets used in the model are populated with data from the model repository on the interpolation stage and stored as a named list in `model@modInp@data` slot. The class and related methods and functions are not intended for direct use by the user.

Slots

`name` character. Name of the parameter as it appears in GAMS, Julia/JuMP, Python/Pyomo, etc.

`desc` character. Description of the parameter for reference.

`type` factor. Type of the parameter, e.g., "set", "map", "numpar", or "bounds". "set" is a set of elements, "map" is a mapping between sets, "numpar" is a numeric parameter, "bounds" is a parameter with lower and upper bounds.

`dimSets` character. A vector of sets used to define the dimension of the parameter.

`defVal` numeric. Default value of numeric parameters. The default value is used to fill the missing values in the model objects. The values are used in the interpolation step and/or passed to the solver software.

`data` data.frame. Data frame with the parameter values.

`interpolation` character. Interpolation rule for numeric parameters across years. Recognized values are any combination of "back", "inter", "forth", e.g., "back.inter" or "forth.inter", indicating the direction of interpolation.

`inClass` character. The class of the parameter, e.g., "technology", "storage", "trade", "supply", "demand", "export", etc.

`misc` list. Any additional information or data to store in the object.

class-repository *An S4 class to store the model objects.*

Description

Use `newRepository` to create a new repository object.

Slots

`name` character. Name of the repository.

`desc` character. Description of the repository.

`data` list. Model objects ("bricks"), e.g., technologies, constraints, costs, etc., stored in with their names as keys, or gropped in named lists.

`permit` character. Vector with names of classes permitted to store in the repository. There is a default list of permitted classes which can be extended or modified. Used in internal functions, it is not common to modify this slot.

`misc` list. Any additional data or information to store in the object.

See Also

Other repository model data: `newRepository()`

class-scenario *An S4 class to represent scenario, an interpolated and/or solved model.*

Description

An S4 class to represent scenario, an interpolated and/or solved model.

Slots

`name` character. Name of the scenario object, for reference, also used in scenario path-functions.

`desc` character. Description of the scenario object, for own references.

`model` model. Model object with the model data and configuration settings.

`settings` settings. Settings object with the scenario-specific settings. Initialized with the default settings from the model configuration object. Overrule the model config for the scenario-specific parameters.

`modInp` modInp. Model input object with the interpolated model parameters.

`modOut` modOut. Model output object with the model solution and logs.

`status` character. Indication of the solution status of the model/scenario.

`inMemory` logical. Indication if the scenario is stored in memory or on the disk.

`path` character. Path to the scenario folder with the model data and scripts.

`misc` list. Any additional data or information to store in the object, added by the user or the functions.

See Also

`interpolate()`, `solve()`, `register()`, `summary()`, `newScenario()`

class-settings *An S4 class to represent scenario settings*

Description

Class 'settings' inherits all slots from class 'config' and adds the following:

Slots

`subset` list. Named list of subsets used in the model. The names of the list elements are the names of the subsets, the values are the vectors of the subset elements. The subsets are used to narrow the dimension of the model variables and constraints.

`yearFraction` numeric. The fraction of a year covered by the calendar, e.g. 1 for annual calendar (default), 0.5 for semi-annual, 0.25 for quarterly, etc. Currently must be specified manually for subset calendars to validate the sum of the shares.

`solver` list. Named list of solver parameters. The names of the list elements are the names of the solver parameters, the values are the solver parameters themselves. The solver parameters are used to control the optimization process.

`sourceCode` list. Named list of paths to the source code files. The names of the list elements are the names of the source code files, the values are the paths to the source code files. The source code files are used to store the model and scenario scripts.

See Also

Other class config settings scenario model: `class-config`

class-technology *An S4 class to represent technology*

Description

Technology of a technological process in the model is used to convert input commodities into output commodities with consumption or production of auxiliary commodities linked to other parameters or variables of the technology. A broad set of parameters provides flexibility to model various technological processes, including efficiency, availability, costs, and exogenous shocks (weather factors).

Slots

`name` character. Name of the technology, used in sets.

`desc` character. Optional description of the technology for reference.

`input` data.frame. Main commodities input. Main commodities are linked to the process capacity and activity. Their parameters are defined in the `ceff` slot.

comm character. Name of the input commodity.

unit character. Unit of the input commodity.

group character. Name of input-commodities-group.

combustion numeric. combustion factor from 0 to 1 (default 1) to calculate emissions from fuels combustion (commodities intermediate consumption, more broadly)

`output` data.frame. Main commodities output. Main commodities are linked to the process capacity and activity. Their parameters are defined in the `ceff` slot.

comm character. Name of the output commodity.

unit character. Unit of the output commodity.

group character. Name of output-commodities-group.

`aux` data.frame. Auxilary commodities, both input and output, their parameters are defined in the `aeff` slot.

acomm character. Name of the auxilary commodity.

unit character. Unit of the auxilary commodity.

`units` data.frame. Key units of the process activity and capacity (for reference).

capacity character. Unit of capacity

use character. Unit of 'use' (grouped input) if applicable.

activity character. Unit of activity variable of the technology.

costs character. Currency of costs variable of the technology.

`group` data.frame. Details for commodity groups if defined in input and output slots (for reference).

group character. Name of the group. Must match the group names in the input and output slots.

desc character. Description of the group.

unit character. Unit of the group.

`cap2act` numeric. Capacity to activity ratio. Default is 1. Specifies how much product (activity, or output commodity if identical) will be produced per unit of capacity.

`geff` data.frame. Input-commodity-group efficiency parameters.

region character. Name of region to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

group character. Name of group to apply the parameter. Required, must match the group names in the input and output slots.

ginp2use numeric. Group-input-to-use coefficient, default is 1.

`ceff` data.frame. Main commodity and activity efficiency parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

comm character. Name of commodity to apply the parameter, different parameters require specification either input or output commodity.

cinp2use numeric. Commodity-input-to-use coefficient, default is 1.

use2cact numeric. Use-to-commodity-activity coefficient, default is 1.

cact2cout numeric. Commodity-activity-to-commodity-output coefficient, default is 1.

cinp2ginp numeric. Commodity-input-to-group-input coefficient, default is 1.

share.lo numeric. Lower bound on a share of commodity within a group, default is 0.

share.up numeric. Upper bound on a share of commodity within a group, default is 1.

share.fx numeric. Fixed share of commodity within a group, ignored if NA. This parameter overrides `share.lo` and `share.up`.

afc.lo numeric. Lower bound on the physical value of the commodity, ignored if NA.

afc.up numeric. Upper bound on the physical value of the commodity, ignored if NA.

afc.fx numeric. Fixed physical value of the commodity, ignored if NA. This parameter overrides `afc.lo` and `afc.up`.

`aeff` data.frame. Parameters linking main commodities, activities, and capacities to auxiliary commodities.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

acomm character. Name of auxilary commodity to apply the parameter.

cinp2ainp numeric. Main-commodity-input-to-auxilary-commodity-input coefficient, ignored if NA.

cinp2aout numeric. Main-commodity-input-to-auxilary-commodity-output coefficient, ignored if NA.

cout2ainp numeric. Main-commodity-output-to-auxilary-commodity-input coefficient, ignored if NA.

cout2aout numeric. Main-commodity-output-to-auxilary-commodity-output coefficient, ignored if NA.

act2ainp numeric. Technology-activity-to-auxilary-commodity-input coefficient, ignored if NA.

act2aout numeric. Technology-activity-to-auxilary-commodity-output coefficient, ignored if NA.

cap2ainp numeric. Technology-capacity-to-auxilary-commodity-input coefficient, ignored if NA.

cap2aout numeric. Technology-capacity-to-auxilary-commodity-output coefficient, ignored if NA.

ncap2ainp numeric. Technology-new-capacity-to-auxilary-commodity-input-coefficient, ignored if NA.

ncap2aout numeric. Technology-new-capacity-to-auxilary-commodity-output coefficient, ignored if NA.

`af` data.frame. Timeslice-level availability factor parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

af.lo numeric. Lower bound on the availability factor, default is 0.

af.up numeric. Upper bound on the availability factor, default is 1.

af.fx numeric. Fixed availability factor, ignored if NA. This parameter overrides `af.lo` and `af.up`.

rampup numeric. Ramping-up time constraint RHS value, ignored if NA. Depends on the technology timeframe.

rampdown numeric. Ramping-down time constraint RHS value, ignored if NA. Depends on the technology timeframe.

`afs` data.frame. Timeframe-level availability factor constraints.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, required.

afs.lo numeric. Lower bound on the availability factor for the timeframe, default is 0.

afs.up numeric. Upper bound on the availability factor for the timeframe, default is 1.

afs.fx numeric. Fixed availability factor for the timeframe, ignored if NA. This parameter overrides `afs.lo` and `afs.up`.

`weather` data.frame. Parameters linking `weather factors` (external shocks specified by `weather` class) to the availability parameters `af`, `afs`, and `afc`.

weather character. Name of the applied weather factor, required, must match the weather factor names in a `weather` class in the model.

comm character. Name of the commodity with specified `afc.*` to be affected by the weather factor, required if `afc.*` parameters are specified.

wafc.lo numeric. Multiplying coefficient to the lower bound on the commodity availability parameter `afc.lo`, ignored if NA.

wafc.up numeric. Multiplying coefficient to the upper bound on the commodity availability parameter `afc.up`, ignored if NA.

wafc.fx numeric. Multiplying coefficient to the fixed value of the commodity availability parameter `afc.fx`, ignored if NA. This parameter overrides `wafc.lo` and `wafc.up`.

waf.lo numeric. Multiplying coefficient to the lower bound on the availability factor parameter `af.lo`, ignored if NA.

waf.up numeric. Multiplying coefficient to the upper bound on the availability factor parameter `af.up`, ignored if NA.

waf.fx numeric. Multiplying coefficient to the fixed value on the availability factor parameter `af.fx`, ignored if NA. This parameter overrides `waf.lo` and `waf.up`.

wafs.up numeric. Multiplying coefficient to the upper bound on the availability factor parameter `afs.up`, ignored if NA.

wafs.lo numeric. Multiplying coefficient to the lower bound on the availability factor parameter `afs.lo`, ignored if NA.

wafs.fx numeric. Multiplying coefficient to the fixed value on the availability factor parameter `afs.fx`, ignored if NA. This parameter overrides `wafs.lo` and `wafs.up`.

`fixom` data.frame. Fixed operational and maintenance cost (per unit of capacity a year).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

fixom numeric. Fixed operational and maintenance cost, default is 0.

`varom` data.frame. Variable operational and maintenance cost (per unit of activity or commodity).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of the time-slice or (grand-)parent timeframe to apply the parameter, NA for every time-slice of the technology timeframe.

varom numeric. Variable operational and maintenance cost per unit of activity, default is 0.

comm character. Name of the commodity for which the parameter will be applied, required for `cvarom` parameter.

cvarom numeric. Variable operational and maintenance cost per unit of commodity, default is 0.

acomm character. Name of the auxilary commodity for which the `avarom` will be applied, required for `avarom` parameter.

avarom numeric. Variable operational and maintenance cost per unit of auxilary commodity, default is 0.

`invcost` data.frame. Total overnight investment costs of the project (per unit of capacity).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

invcost numeric. Total overnight investment costs of the project (per unit of capacity), default is 0.

wacc numeric. Weighted average cost of capital, (currently ignored).

`start` data.frame. The first year the technology can be installed.

region character. Region name to apply the parameter, NA for every region.

start integer. The first year the technology can be installed, NA means all years of the modeled horizon.

`end` data.frame. The last year the technology can be installed.

region character. Region name to apply the parameter, NA for every region.

end integer. The last year the technology can be installed, default is Inf.

`olife` data.frame. Operational life of the installed technology (in years).

region character. Region name to apply the parameter, NA for every region.

olife integer. Operational life of the technology if installed during optimization, in years, default is 1.

`capacity` data.frame. Capacity of the installed technology (in units of capacity).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, required, values between specified years will be interpolated.

stock numeric. Predefined capacity of the technology in units of capacity, default is 0. This parameter also defines the exogenous capacity retirement (age-based), or exogenous capacity additions, not optimized by the model, and not included in investment costs.

cap.lo numeric. Lower bound on the total capacity (preexisting stock and new installations), ignored if NA.

cap.up numeric. Upper bound on the total capacity (preexisting stock and new installations), ignored if NA.

cap.fx numeric. Fixed total capacity (preexisting stock and new installations), ignored if NA. This parameter overrides `cap.lo` and `cap.up`.

ncap.lo numeric. Lower bound on the new capacity (new installations), ignored if NA.

ncap.up numeric. Upper bound on the new capacity (new installations), ignored if NA.

ncap.fx numeric. Fixed new capacity (new installations), ignored if NA. This parameter overrides `ncap.lo` and `ncap.up`.

ret.lo numeric. Lower bound on the capacity retirement (age-based), ignored if NA.

ret.up numeric. Upper bound on the capacity retirement (age-based), ignored if NA.

ret.fx numeric. Fixed capacity retirement (age-based), ignored if NA. This parameter overrides `ret.lo` and `ret.up`.

`optimizeRetirement` logical. Incidates if the retirement of the technology should be optimized. Also requires the same parameter in the `model` or `scenario` class to be set to TRUE to be effective.

`fullYear` logical. Incidates if the technology is operating on a full-year basis. Used in storages. currently ignored for technologies.

`timeframe` character. Name of timeframe level the technology is operating. By default, the lowest level of timeframe of commodities used in the technology is applied.

`region` character. Vector of regions where the technology exists or can be installed. Optional. If not specified, the technology is applied to all regions. If specified, must include all regions used in other slots.

`misc` list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the technology data, or other metadata.

`class-trade` *An S4 class to represent inter-regional trade*

Description

An S4 class to represent inter-regional trade

Details

Trade objects are used to represent inter-regional exchange in the model. Without trade, every region is isolated and can only use its own resources. The class defines trade routes, efficiency, costs, and other parameters related to the process. Number of routes per trade object is not limited. One trade object can have a part or entire trade network of the model. However, it has a distinct name and all the routs will be optimized together. Create separate trade objects to optimize different parts of the trade network (aka transmission lines).

Slots

`name` character. Name of the trade object, used in sets.

`desc` character. Description of the trade object.

`commodity` character. The traded commodity short name.

`routes` data.frame. Source and destination regions. For bivariate trade define both directions in separate rows.

from character. Source region.

to character. Destination region.

`trade` data.frame. Technical parameters of trade.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

trade numeric. Trade volume.

`aux` data.frame. Auxiliary commodity of trade.

acomm character. Name of the auxiliary commodity (used in sets).

unit character. Unit of the auxiliary commodity.

`aeff` data.frame. Auxiliary commodity efficiency parameters.

acomm character. Name of the auxiliary commodity (used in sets).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

trade2ainp numeric. Trade-to-auxiliary-input-commodity coefficient (multiplier).

trade2aout numeric. Trade-to-auxiliary-output-commodity coefficient (multiplier).

`invcost` data.frame. Investment cost, used when capacityVariable is TRUE.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

invcost numeric. Investment cost.

`fixom` data.frame. (not implemented!) Fixed operation and maintenance costs.

varom data.frame. (not implemented!) Variable operation and maintenance costs.

olife numeric. Operational life of the trade object.

start data.frame. Start year when the trade-type of process is available for investment.

- **region** character. Regions where the trade-type of process is available for investment.
- **start** integer. The first year when the trade-type of process is available for investment.

end data.frame. End year when the trade-type of process is available for investment.

- **region** character. Region name to apply the parameter, NA for every region.
- **end** integer. The last year when the trade-type of process is available for investment.

capacity data.frame. (not implemented!) Capacity parameters of the trade object.

capacityVariable logical. If TRUE, the capacity variable of the trade object is optimized. If FALSE, the capacity is defined by availability parameters (ava.*) in the trade-flow units.

cap2act numeric. Capacity to activity ratio.

optimizeRetirement logical. Incidates if the retirement of the trade object should be optimized. Also requires the same parameter in the model or scenario class to be set to TRUE to be effective.

misc list. Additional information.

---

class-weather *S4 class to represent weather factors*

Description

weather is a data-carrying class with exogenous shocks used to influence operation of processes in the model.

Details

Weather factors are separated from the model parameters and can be added or replaced for different scenarios. !!!Additional details...

Slots

name character. Name of the weather factor, used in sets.

desc character. Description of the weather factor.

unit character. Unit of the weather factor.

region character. Region where the weather factor is applied.

timeframe character. Timeframe of the weather factor.

defVal numeric. Default value of the weather factor, 0 by default.

weather data.frame. Weather factor values.

- **region** character. Region name to apply the parameter, NA for every region.
- **year** integer. Year to apply the parameter, NA for every year.
- **slice** character. Time slice to apply the parameter, NA for every slice.
- **wval** numeric. Weather factor value.

misc list. Additional information.

`convert,character-method` *Convert units*

Description

Convert units

Add units to convert function

Usage

```
## S4 method for signature 'character'
convert(from, to, x = 1, database = "base", ...)

## S4 method for signature 'numeric'
convert(x = 1, from, to, database = "base", ...)

## S4 method for signature 'character,character,numeric'
add_to_convert(
  type,
  unit,
  coef,
  alias = "",
  SI_prefixes = FALSE,
  database = "base",
  update = TRUE
)
```

Arguments

`from`	character of length one with unit name
`to`	character of length one with unit name
`x`	numeric vector with data to convert
`database`	character name of a database with units (`base` by default, other options are not implemented yet).
`...`	currently ignored
`type`	character, type of the unit (one of "Energy", "Power", "Mass", "Time", "Length", "Area", "Pressure", "Density", "Volume", "Flow Rates", "Currency").
`unit`	character, the name of the new unit to add to the `database`.
`coef`	numeric, convert factor to the base unit of this type (see the first column of `convert_data[[database]][[type]]`).
`alias`	character vector, alternative name(s) for the same unit.
`SI_prefixes`	logical, can be used with `SI` prefixes, FALSE by default.

Value

numeric vector with converted values

updated `convert_data` in the `.GlobalEnv`, the values will not update the package data.

Examples

```
convert("MWh", "kWh")
convert("kWh", "MJ")
convert("kWh/kg", "MJ/t", 1e-3)
convert("cents/kWh", "USD/MWh")
convert(1000, "kWh", "MWh")
convert("kWh", "MJ")
convert(1, "kWh/kg", "MJ/t")
convert(5, "cents/kWh", "USD/MWh")
## Not run:
convert_data$base$Currency
add_to_convert("Currency", unit = "JPY", coef = 140)
convert_data$base$Currency
## End(Not run)
```

---

convert_data *Basic units conversion database for* convert *methods*

Description

Basic units conversion database for convert methods

Usage

```
convert_data
```

Format

An object of class list of length 1.

---

draw *Draw a schematic representation of a process*

Description

A generic method for drawing a schematic representation of processes-type classes.

Usage

```
draw(obj, ...)

## S4 method for signature 'technology'
draw(obj, ...)

## S4 method for signature 'storage'
draw(obj, ...)

## S4 method for signature 'supply'
draw(obj, ...)
```

```
## S4 method for signature 'demand'
draw(obj, ...)

## S4 method for signature 'export'
draw(obj, ...)

## S4 method for signature 'import'
draw(obj, ...)

## S4 method for signature 'trade'
draw(obj, ...)
```

Arguments

...	Additional arguments passed to the specific method.
object	The object to draw: technology, storage, trade, demand, supply, export, or import.
region	A node to draw the trade process for. node is an alias for region. Default is the first node in the trade object.

Value

displays a schematic representation of the process, returns NULL.

A figure with a schematic representation of the export process.

A figure with a schematic representation of the import process.

Examples

```
TECH01 <- newTechnology(
  "TECH01",
  desc = "Technology Description",
  input = data.frame(
    comm = c("COM1", "COM2", "COM5", "COM7", "COM8", "COM9"),
    group = c("1", "1", NA, "2", "2", "2"),
    unit = c("unit1", "unit2", "unit5", "unit7", "unit8", "unit9")
  ),
  output = data.frame(
    comm = c("COM3", "COM4", "COM6"),
    group = c("3", NA, "3"),
    unit = c("unit3", "unit4", "unit6")
  ),
  group = data.frame(
    group = c("1", "2", "3"),
    desc = c("Group1", "Group2", "Group3"),
    unit = "unit"
  ),
  aux = data.frame(
    acomm = c("AUX1", "AUX2", "AUX3", "AUX4"),
    unit = c("unit1", "unit2", "unit3", "unit4")
  ),
  region = c("R1", "R2", "R3"),
  geff = data.frame(
    group = c("1", "2"),
    ginp2use = c(0.12, 0.789)
```

```
  ),
  ceff = data.frame(
    comm = c("COM1", "COM2", "COM5", "COM7", "COM8", "COM9", "COM3", "COM4", "COM6"),
    cinp2ginp = c(.1, .2, NA, .7, .8, .9, rep(NA, 3)),
    cinp2use = c(NA, NA, .5, NA, NA, NA, rep(NA, 3)),
    use2cact = c(rep(NA, 6), .36, .4, .36),
    cact2cout = c(rep(NA, 6), .3, NA, .6),
    share.lo = c(.01, .02, NA, .07, .08, .0, .03, NA, .06),
    share.up = c(.91, .92, NA, .97, .98, 1, .83, NA, .96)
  ),
  aeff = data.frame(
    acomm = c("AUX1", "AUX2", "AUX3", "AUX4"),
    comm = c(NA, "COM1", NA, "COM3"),
    act2ainp = c(1, NA, NA, NA),
    cinp2aout = c(NA, 2, NA, NA),
    cap2aout = c(NA, NA, 3, NA),
    cout2aout = c(NA, NA, NA, 4)
  ),
  weather = data.frame(
    weather = "WEATHER_CF1",
    waf.up = .99
  )
)
draw(TECH01)
STG_ELC <- newStorage(
  name = "STG_ELC", # used in sets
  desc = "Electricity storage (battery)", # for own reference
  commodity = "ELECTRICITY", # must match the commodity name in the model
  aux = data.frame(
    acomm = "LITHIUM", # auxiliary commodity for battery production
    unit = "ton" # unit of the auxiliary commodity
  ),
  start = data.frame(
    start = 2020 # the first year of the process is available for installation
  ),
  end = data.frame(
    end = 2030 # last year of the process is available for installation
  ),
  olife = data.frame(
    olife = 20 # operational life of the storage in years
  ),
  seff = data.frame(
    stgeff = 0.999, # storage efficiency
    inpeff = 0.9, # charging efficiency
    outeff = 0.9 # discharging efficiency
  ),
  aeff = data.frame(
    acomm = "LITHIUM", # track lithium use for battery production
    ncap2ainp = convert(4 * 250, "Wh/kg", "GWh/kt") # lithium per energy capacity
  ),
  af = data.frame(
    # af.lo = 0., # lower bound for the capacity factor
    af.up = 1. # upper bound for the capacity factor
  ),
  fixom = data.frame(
    # region = "R1",
    # year = 2020,
```

```
    fixom = 0.9 # fixed operation and maintenance cost
  ),
  cap2stg = 4, # four-hours of storage
  invcost = data.frame(
    region = c("R1", NA), # region R1 and all other regions
    invcost = c(1e3, 1.1e3) # investment cost in MUSD/GWh of 4-hour storage
  ),
  fullYear = TRUE, # full year storage cycle
  weather = data.frame(
    weather = "AMBIENT_TEMP", # weather factor for capacity factor
    waf.up = 1 # affects upper boundary of capacity factor
    # waf.lo = 0.9 # affects lower boundary of capacity factor
  )
  # region = c("R1", "R2", "R3"),
)
draw(STG_ELC)

SUP_COA <- newSupply(
  name = "SUP_COA",
  desc = "Coal supply",
  commodity = "COA",
  unit = "PJ",
  reserve = data.frame(
    region = c("R1", "R2", "R3"),
    res.up = c(2e5, 1e4, 3e6) # total reserves/deposits
  ),
  availability = data.frame(
    region = c("R1", "R2", "R3"),
    year = NA_integer_,
    slice = "ANNUAL",
    ava.up = c(1e3, 1e2, 2e2), # annual availability
    cost = c(10, 20, 30) # cost of the resource (currency per unit)
  ),
  region = c("R1", "R2", "R3")
)
draw(SUP_COA)
DSTEEL <- newDemand(
  name = "DSTEEL",
  desc = "Steel demand",
  commodity = "STEEL",
  unit = "Mt",
  dem = data.frame(
    region = "UTOPIA", # NA for every region
    year = c(2020, 2030, 2050),
    slice = "ANNUAL",
    dem = c(100, 200, 300)
  ),
  region = "UTOPIA", # optional, to narrow the specification of the demand
)
draw(DSTEEL)
EXPOIL <- newExport(
  name = "EXPOIL", # used in sets
  desc = "Oil export from the model to RoW", # for own reference
  commodity = "OIL", # must match the commodity name in the model
  unit = "Mtoe", # for own reference
  exp = data.frame(
    region = rep(c("R1", "R2"), each = 2), # export region(s)
```

```
    year = rep(c(2020, 2050)), # export years
    price = 500, # export price in MUSD/Mtoe (USD/t),
    exp.up = rep(c(1e3, 1e4), each = 2), # upper bound for export in each year
    exp.lo = rep(c(5e2, 0), each = 2) # lower bound for export in each year
  )
)
draw(EXPOIL)
IMPOIL <- newImport(
  name = "IMPOIL", # used in sets
  desc = "Oil import to the model to RoW", # for own reference
  commodity = "OIL", # must match the commodity name in the model
  unit = "Mtoe", # for own reference
  imp = data.frame(
    region = rep(c("R1", "R2"), each = 2), # import region(s)
    year = rep(c(2020, 2050)), # import years
    price = 600, # import price in MUSD/Mtoe (USD/t),
    imp.up = rep(c(1e4, 1e6), each = 2), # upper bound for import in each year
    imp.lo = rep(c(1e4, 1e5), each = 2) # lower bound for import in each year
  )
)
draw(IMPOIL)
PIPELINE2 <- newTrade(
  name = "PIPELINE2",
  desc = "Some transport pipeline",
  commodity = "OIL",
  routes = data.frame(
    src = c("R1", "R1", "R2", "R3"),
    dst = c("R2", "R3", "R3", "R2")
  ),
  trade = data.frame(
    src = c("R1", "R1", "R2", "R3"),
    dst = c("R2", "R3", "R3", "R2"),
    teff = c(0.912, 0.913, 0.923, 0.932)
  ),
  aux = data.frame(
    acomm = c("ELC", "CH4"),
    unit = c("MWh", "kt")
  ),
  aeff = data.frame(
    acomm = c("ELC", "CH4", "ELC", "CH4"),
    src = c("R1", "R1", "R2", "R3"),
    dst = c("R2", "R2", "R3", "R2"),
    csrc2ainp = c(.5, NA, .3, NA),
    cdst2ainp = c(.4, NA, .6, NA),
    csrc2aout = c(NA, .1, NA, .2)
  ),
  olife = list(olife = 60)
)
draw(PIPELINE2, node = "R1")
draw(PIPELINE2, node = "R2")
draw(PIPELINE2, node = "R3")
```

`drop_na_cols` *Drop columns in a data.frame with all NA values*

Description

A wrapper with dplyr functions to drop columns with no information (all NA values)

Usage

```
drop_na_cols(x, unique = TRUE)
```

Arguments

x	data.frame
unique	logical, if TRUE (default), unique() function will be applied to the result.

Value

data.frame with dropped columns

Examples

```
x <- data.frame(a = c(1, 2, NA), b = c(NA, NA, NA), c = c(NA, 2, 3))
drop_na_cols(x)
```

---

en_install_julia_pkgs *Install Julia packages*

Description

Install Julia packages

Usage

```
en_install_julia_pkgs(pkgs = NULL, update = FALSE)
```

Arguments

pkgs	A character vector of Julia packages to install. The default is c("JuMP", "HiGHS", "Cbc", "Clp", "RData", "RCall", "CodecBzip2", "Gadfly", "DataFrames", "CSV", "SQLite", "Dates"). If you have pre-installed CPLEX or Gurobi, you can add them to the list.

Value

NULL if the completion is successful. The verification of the installation is done by the user or by the function en_check_julia().

Examples

```
## Not run:
en_install_julia_pkgs()

## End(Not run)
```

`findData` *Performs search for available data in* scenario *object.*

Description

Performs search for available data in *scenario* object.

Usage

```
findData(
  scen,
  dataType = c("parameters", "variables"),
  setsNames_ = NULL,
  valueColumn = TRUE,
  allSets = TRUE,
  ignore.case = FALSE,
  add_weights = "auto",
  dropEmpty = TRUE,
  dfDim = TRUE,
  dfNames = TRUE,
  asMatrix = FALSE
)
```

Arguments

`scen`	object *scenario* with model solution.
`dataType`	type of data to lok for (currently only "parameters" and "variables").
`setsNames_`	regular expression pattern for names of sets which will be included in search.
`valueColumn`	logical, if TRUE will return variables and parameters with 'value' column (to filter sets and mappings).
`allSets`	logical, if TRUE *and* operator should be used in search the sets, *or* will be used if FALSE.
`ignore.case`	grepl parameter for matching names.
`dropEmpty`	logical, if TRUE drops parameters and variables with zero length.
`dfDim`	logical, if TRUE returns dimension *dim*.
`dfNames`	logical, when TRUE returns names of the data frame column.
`asMatrix`	return results as a matrix (not implemented).

Value

list with variables and parameters name, each includes *dim* and *names* character vectors.

findDuplicates *Function to find duplicated values in interpolated scenario.*

Description

Function to find duplicated values in interpolated scenario.

Usage

```
findDuplicates(x)
```

Arguments

x	scenario or data.frame with data to check.

Value

data.frame with duplicated values.

Examples

```
## Not run:
findDuplicates(scen_BASE)

## End(Not run)
```

getData *Extracts information from scenario objects, based on filters.*

Description

Extracts information from scenario objects, based on filters.

Usage

```
getData(
  scen,
  name = NULL,
  ...,
  merge = FALSE,
  process = FALSE,
  parameters = TRUE,
  variables = TRUE,
  ignore.case = TRUE,
  newNames = NULL,
  newValues = NULL,
  na.rm = FALSE,
  digits = NULL,
  drop.zeros = FALSE,
  asTibble = TRUE,
```

```
  stringsAsFactors = FALSE,
  yearsAsFactors = FALSE,
  drop_duplicated_scenarios = TRUE,
  scenNameInList = as.logical(length(scen) - 1),
  verbose = FALSE
)

get_data(
  scen,
  name = NULL,
  ...,
  merge = FALSE,
  process = FALSE,
  parameters = TRUE,
  variables = TRUE,
  ignore.case = TRUE,
  newNames = NULL,
  newValues = NULL,
  na.rm = FALSE,
  digits = NULL,
  drop.zeros = FALSE,
  asTibble = TRUE,
  stringsAsFactors = FALSE,
  yearsAsFactors = FALSE,
  drop_duplicated_scenarios = TRUE,
  scenNameInList = as.logical(length(scen) - 1),
  verbose = FALSE
)
```

Arguments

scen	Object scenario or list of scenarios.
name	character vector with names of parameters and/or variables.
...	filters for various sets (setname = c(val1, val2) or setname_ = "matching pattern"), see details.
merge	if TRUE, the search results will be merged in one dataframe; the named list will be returned if FALSE.
process	if TRUE, dimensions "tech", "stg", "trade", "imp", "expp", "dem", and "sup" will be renamed with "process".
parameters	if TRUE, parameters will be included in the search and returned if found.
variables	if TRUE, variables will be included in the search and returned if found.
ignore.case	grepl parameter if regular expressions are used in '...' or 'name_'.
newNames	renaming sets, named character vector or list with new names as values, and old names as names - the input parameter to renameSets function. The operation is performed before merging the data (merge parameter).
newValues	revalue sets, named character vector or list with new values as values, and old values as names - the input parameter to revalueSets function. The operation is performed after merging the data (merge parameter).
na.rm	if TRUE, NA values will be dropped.

digits	if integer, indicates the number of decimal places for rounding, if NULL - no actions.
drop.zeros	logical, should rows containing zero values be filtered out.
asTibble	logical, if the data.frames should be converted into tibbles.
stringsAsFactors	logical, should the sets values be converted to factors?
yearsAsFactors	logical, should year be converted to factors? Set 'year' is integer by default.
scenNameInList	logical, should the name of the scenarios be used if not provided in the list with several scenarios?
verbose	

Examples

```
## Not run:
data("utopia_scen_BAU.RData")
getData(scen, name = "pDemand", year = 2015, merge = TRUE)
getData(scen, name = "vTechOut", comm = "ELC", merge = TRUE, year = 2015)
elc2050 <- getData(scen, parameters = FALSE, comm = "ELC", year = 2050)
names(elc2050)
elc2050$vBalance

## End(Not run)
```

---

get_registry *Returns the current registry object.*

Description

Returns the current registry object.

Usage

```
get_registry()
```

Value

The current registry object.

---

get_slot_info *Retrieve slot details in rd-format*

Description

Retrieve slot details in rd-format

Usage

```
get_slot_info(class_name = "technology", slot_name = "ceff", col_names = TRUE)
```

Arguments

class_name	character, name of class.
slot_name	character, name of slot to retrieve.
col_names	logical, if columns information should be returned for data.frame slots.

Value

character, roxygen2 formatted string with slot details.

Examples

```
slotNames("technology")
get_slot_info("technology", "input") |> cat()
get_slot_info("technology", "capacity") |> cat()
get_slot_info("demand", "dem") |> cat()
get_slot_info("commodity", "agg") |> cat()
```

---

hour2HOUR *Convert hours (integer) values to HOUR set 'hNN'*

Description

Convert hours (integer) values to HOUR set 'hNN'

Usage

```
hour2HOUR(x, width = 2, prefix = "h", flag = "0")
```

Arguments

x	integer vector, hours (for example, 0-23 for daily data, 0-167 for weekly data, etc.)
width	integer, width of the output string
prefix	character, prefix to add to the name, default is 'h'
flag	character, flag to add to the name, default is '0'

Value

character vector of the same length as x with formatted hours to be used in the HOUR set.

Examples

```
hour2HOUR(0:23)
```

interpolate,model-method *Interpolate model*

Description

Interpolate model

Usage

```
## S4 method for signature 'model'
interpolate(object, ...)
```

Arguments

object	model or scenario type of object.

Value

scenario object with enclosed model (slot @model) and interpolated parameters (slot @modInp).

Examples

```
## Not run:
scen <- interpolate(mod)

## End(Not run)
```

isInMemory *Is object stored in memory?*

Description

Is object stored in memory?

Usage

```
isInMemory(obj)
```

Arguments

obj	Object, checks

Value

Logical value, TRUE if object is stored in memory, FALSE if on disk.

Examples

```
## Not run:
isInMemory(scen_BASE)

## End(Not run)
```

load_scenario *Load scenario (in progress)*

Description

Load scenario (in progress)

Usage

```
load_scenario(
  path,
  name = NULL,
  env = .scen,
  overwrite = FALSE,
  ignore_errors = FALSE,
  verbose = TRUE
)
```

Arguments

path	character. Path to saved with function save_scenario scenario directory.
name	character. Name to assign to the loaded scenario object. By default, the name is taken from the loaded scenario object.
env	environment. Environment to assign the loaded scenario object.
overwrite	logical. Overwrite existing scenario object in the environment.
ignore_errors	logical. Ignore load errors and continue execution. This option is useful when some data is missing or corrupted.
verbose	logical. Print messages.

Value

TRUE if scenario is loaded, FALSE if not.

Examples

```
## Not run:
load_scenario("scenarios/base")

## End(Not run)
```

`make_scenario_dirname` *Make a name for a scenario directory*

Description

A function to automate the creation of a scenario directory name. Used internally in `solve*()` and `interpolate*()` functions. Also can be used to amend the name of the scenario directory and explicitly assign the directory name to save the scenario object.

Usage

```
make_scenario_dirname(
  scen,
  name = scen@name,
  model_name = scen@model@name,
  calendar_name = scen@settings@calendar@name,
  horizon_name = scen@settings@horizon@name,
  prefix = NULL,
  suffix = NULL,
  sep = "_"
)
```

Arguments

`scen`	scenario object
`name`	character, name of the scenario, default is `scen@name`
`model_name`	character, name of the model, default is `scen@model@name`
`calendar_name`	character, name of the calendar, default is `scen@settings@calendar@name`
`horizon_name`	character, name of the horizon, default is `scen@settings@horizon@name`
`prefix`	character, prefix to add to the name
`suffix`	character, suffix to add to the name
`sep`	character, separator, default is _

Value

character, name of the scenario directory

Examples

```
## Not run:
make_scenario_dirname(scen_BASE)
make_scenario_dirname(scen_BASE, prefix = "prefix", suffix = "suffix")

## End(Not run)
```

make_timetable *Create timetable of time-slices from given structure as a list*

Description

Create timetable of time-slices from given structure as a list

Usage

```
make_timetable(
  struct = list(ANNUAL = "ANNUAL"),
  year_fraction = 1,
  warn = FALSE
)
```

Arguments

struct	named list of timeframes with sets of timeslices and optional shares of every slice or frame in the nest
warn	logical, if TRUE, warning will be issued if ANNUAL level does not exists in the given structure. The level will be auto-created to complete the time-structure.

Value

an data.frame with the specified structure.

Examples

```
make_timetable()
make_timetable(list("SEASON" = c("WINTER", "SUMMER")))
make_timetable(list("SEASON" = c("WINTER" = .6, "SUMMER" = .4)))
make_timetable(list(
  "SEASON" = list(
    "WINTER" = list(.3, DAY = c("MORNING", "EVENING")),
    "SUMMER" = list(.7, DAY = c("MORNING", "EVENING"))
  )
))

make_timetable(list(
  "SEASON" = list("WINTER" = .3, "SUMMER" = .7),
  "DAY" = c("MORNING", "EVENING")
))
```

`newCalendar` *Generate a new calendar object from*

Description

Calendars are defined by the structure of timeframes and time-slices with shares of time in a year. The structure is represented by a `timetable` data.frame with levels of timeframes in the named columns, and names of individual time-slices in every timeframe. The number of rows in `timetable` is equal to the total number of time-slices on the lowest level. Every timeframe is a set of timeslices ("slices") - a named fragment of time with a year-share. Timeframes have nested structure. Currently, every "parent"-timeframe must have the same number of elements as the "child"-timeframe. (This may change in the future.)

ANNUAL character, annual, the top level of timeframes

TIMEFRAME2 character, (optional) first subannual level of timeframes

TIMEFRAME3 character, (optional) second subannual level of timeframes

... character, (optional) further subannual levels of timeframes

slice character, name of the time-slices used in sets to refer to the lowest level of timeframes. If not specified, will be auto-created with the formula: `{SLICE2}_{SLICE3}...`

Usage

```
newCalendar(
  name = "",
  desc = "",
  timetable = NULL,
  year_fraction = 1,
  default_timeframe = NULL,
  misc = list(),
  ...
)
```

Arguments

`name`	character. Name of the calendar object. Use to distingush between different structures and subsets of time-slices. The name is used to propose default folder names for the model/scenario scripts to separate solutions of the same scenario with different calendar objects.
`desc`	character. Description of the calendar object, for own references.
`timetable`	data.frame. Data frame with levels of timeframes in the named columns, and number of rows equal to the total number of time-slices on the lowest level. Every timeframe is a set of time-slices ("slices") - a named fragment of time with a year-share. Timeframes have nested structure where every slice serves as a parent for the lower level of time-slices (children). The first column is the name of the time-slice, the rest of the columns are the names of the timeframes. The values are the share of the year covered by the time-slice. The sum of the shares in every timeframe should be equal to 1. `weight` is an optional column with the weight of the time-slice in the year, used for sumpled/subset selection of the time-slices.

year_fraction	numeric. The fraction of a year covered by the calendar, e.g. 1 for annual calendar (default), 0.5 for semi-annual, 0.25 for quarterly, etc. Currently must be specified manually for subset calendars to validate the sum of the shares.
default_timeframe	character. The name of the default level of the time-slices used in the model. If not specified, the lowest level of the timeframes is used as the default timeframe.
misc	list. Any additional data or information to store in the object.
...	ignored

Value

an object of class calendar with the specified structure.

Examples

```
newCalendar()
```

---

newCommodity *Create new commodity object*

Description

Create new commodity object

Usage

```
newCommodity(
  name = "",
  desc = "",
  limtype = "LO",
  timeframe = character(),
  unit = character(),
  agg = data.frame(),
  emis = data.frame(),
  misc = list()
)
```

Arguments

name	character. Name of the commodity.
desc	character. Optional description of the commodity for reference.
limtype	factor or character. The limit type of the commodity in balance equation, "LO", "UP", or "FX". "LO" by default, meaning that the level of commodity in the model is restricted with the lower bound, excess is allowed. "UP" means that the level of commodity cannot exceed the upper bound. "FX" means that total commudity supply and demand are equal, no excess or deficit is allowed.
timeframe	character. The default time-frame this commodity operates in the model. The lowest timeframe in the model is used by default.
unit	character. The main unit of the commodity used in the model.

agg — data.frame. Used to define an aggregation of several commodities into the name commodity.

- **comm** character. Name of a commodity being aggregated.
- **unit** character. Unit of the commodity being aggregated.
- **agg** numeric. weight of the commodity in the aggregation, must be set for all aggregated commodities.

emis — data.frame. Emissions factors related to the commodity consumption (if "combustion" parameter of a technology which consumes the commodity is > 0).

- **comm** character. Name of the emitted commodity.
- **unit** character. Unit of the emission factor.
- **emis** numeric. Emission factor, emissions released per unit of the consumed commodity.

misc — list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the commodity data, or other metadata.

Value

commodity object

Examples

```
newCommodity(name = "ELC", desc = "Electricity")
```

---

newConstraint *Create constraint object to add custom constraints to the model.*

Description

The function creates a new constraint object that can be used to add custom constraints to the model.

Usage

```
newConstraint(
  name,
  desc = "",
  ...,
  eq = "==",
  for.each = NULL,
  rhs = data.frame(),
  defVal = NULL,
  interpolation = "inter",
  replace_zerros = 1e-20
)

isConstraint(object)

newConstraintS(
  name,
```

```
  type,
  eq = "==",
  rhs = 0,
  for.sum = list(),
  for.each = list(),
  defVal = 0,
  rule = NULL,
  comm = NULL,
  cout = TRUE,
  cinp = TRUE,
  aout = TRUE,
  ainp = TRUE
)
```

Arguments

name	character. Name of the constraint object, used in sets.
desc	character. Description of the constraint.
...	named or unnamed list(s) of left-hand side (LHS) linear terms (summands) to define the constraint. Every summand is defined as a list with the following elements: • variable - name of the variable in the summand. • mult - multiplier for the variable in the summand. • for.sum - list of sets for which the summand is defined. The summands can be passed as named or unnamed lists. They will be added to the lhs slot of the constraint object as linear terms of mulipliers and variables.
eq	Type of the relation ('==' default, '<=', '>=')
for.each	list or data.frame with sets that define the dimension of the constraint.
rhs	a numeric value, list or data frame with sets and numeric values for each constraint. Note: zero values will be replaced with replace_zerros to avoid dropping them by the interpolation algorithms.
defVal	numeric. The default value for the rhs. It is recommended to set the default value for the rhs of every constraint to avoid unexpected behavior. If not specified, the default value is 0, and the warning is issued.
interpolation	character. Interpolation rule for the constraint. Recognized values, any combination of "back", "inter", "forth", e.g., "back.inter" or "forth.inter", indicating the direction of interpolation. The default value is "inter", meaning that the interpolation is done for years between the specified values. The "back" and "forth" values induce backward and forward interpolation of the rhs values, respectively.
replace_zerros	numeric value to replace zero values in rhs and defVal. Default is 1e-20.
object	any R object

Details

Custom constraints extend the functionality of the model by adding user-defined constraints to the optimization problem. If the predefined constraints are not sufficient to describe the problem, custom constraints can be used to add linear equlity or inequality constraints to define additional relationships between the variables. In many cases this can be done without writing constraints in the GAMS, Julia/JuMP, Python/Pyomo, or GLPK-MathProg languages by using the constrant

class and the newConstraint function. To define a custom constraint with the newConstraint function, the user needs to specify the name of the constraint, the type of the relation (equality, less than or equal, greater than or equal), the left-hand side (LHS) terms of the statement, and the right-hand side (RHS) value. The dimension of the constraint is set by the for.each parameter. The 'lhs' terms are defined as a list of linear terms (summands). Each summand consists of a variable, a multiplier, and a set of sets for which the summand is defined.

Value

Object of class constraint.

TRUE if the object inherits class constraint, FALSE otherwise.

Functions

- isConstraint(): Check if an object is a constraint.

See Also

Other class constraint policy: class-constraint, class-costs, newCosts(), newSubsidy(), newTax(), subsidy-class, tax-class

Other class constraint: class-constraint

Examples

```
isConstraint(1)
isConstraint(newConstraint(""))
```

---

newCosts *Create new costs object*

Description

Costs object is used to define additional costs to add to the model's objective function.

Usage

```
newCosts(name, variable, desc = "", mult = NULL, subset = NULL, misc = NULL)
```

Arguments

name	get_slot_info("costs", "name")
variable	get_slot_info("costs", "variable")
desc	get_slot_info("costs", "desc")
mult	get_slot_info("costs", "mult")
subset	get_slot_info("costs", "subset")

Value

costs object with given specifications.

See Also

Other class constraint policy: class-constraint, class-costs, newConstraint(), newSubsidy(), newTax(), subsidy-class, tax-class

---

newDemand *Create new demand object*

Description

Create new demand object

Update data in a demand object

Usage

```
newDemand(
  name = "",
  desc = character(),
  commodity = character(),
  unit = character(),
  dem = data.frame(),
  region = character(),
  misc = list(),
  ...
)

## S4 method for signature 'demand'
update(object, ...)
```

Arguments

name	character. Name of the demand.
desc	character. Optional description of the demand for reference.
commodity	character. Name of the commodity for which the demand will be specified.
unit	character. Optional unit of the commodity.
dem	data.frame. Specification of the demand. **region** character. Name of region for the demand value. NA for every region. **year** integer. Year of the demand. NA for every year. **slice** character. Name of the slice for the demand value. NA for every slice. **dem** numeric. Value of the demand.
region	character. Optional name of region to narrow the specification of the demand in the case of used NAs. Error will be returned if specified regions in @dem are not mensioned in the @region slot (if the slot is not empty).
misc	list. Optional list of additional information.
object	demand object

Value

demand object with given specifications.

Examples

```
DSTEEL <- newDemand(
 name = "DSTEEL",
 desc = "Steel demand",
 commodity = "STEEL",
 unit = "Mt",
 dem = data.frame(
    region = "UTOPIA", # NA for every region
    year = c(2020, 2030, 2050),
    slice = "ANNUAL",
    dem = c(100, 200, 300)
 ),
 region = "UTOPIA", # optional, to narrow the specification of the demand
 )
 class(DSTEEL)
 draw(DSTEEL)
```

---

newExport *Create new export object*

Description

Export object represent commodity export to the Rest of the World (RoW).

Usage

```
newExport(
  name,
  desc = "",
  commodity = "",
  unit = NULL,
  reserve = Inf,
  exp = data.frame(),
  misc = list(),
  ...
)
```

Arguments

name	character. Name of the export object, used in sets.
desc	character. Description of the export object.
commodity	character. Name of the exported commodity.
unit	character. Unit of the exported commodity.
reserve	numeric. Total accumulated limit through the model horizon.
exp	data.frame. Export parameters.
	region character. Region name to apply the parameter; use NA to apply to all regions.
	year integer. Year to apply the parameter; use NA to apply to all years.
	slice character. Time slice to apply the parameter; use NA to apply to all slices.

exp.lo numeric. Export lower bound.

exp.up numeric. Export upper bound.

exp.fx numeric. Fixed export volume, ignored if NA. This parameter overrides `exp.lo` and `exp.up`.

`misc` list. Additional information.

Details

`export` is a type of process that adds an "external" source to a commodity to the model. The Rest of the World (RoW) is not modeled explicitly, `export` and `import` objects define and control the exchange with the RoW. The operation of the export object is similar to the `demand` objects, the two different classes are used to distinguish domestic and external sources of final consumption. The export is controlled by the `exp` data frame, which specifies bounds and fixed values for the export of the export flow. The `exp.fx` column is used to specify fixed values of the export flow, making the export flow exogenous. The `exp.lo` and `exp.up` columns are used to specify lower and upper bounds of the export flow, making the export flow endogenous. The `price` column is used to specify the exogenous price for the export commodity. The `reserve` slot is used to set limits on the total export over the model horizon.

Value

export object with given specifications.

Examples

```
EXPOIL <- newExport(
  name = "EXPOIL", # used in sets
  desc = "Oil export from the model to RoW", # for own reference
  commodity = "OIL", # must match the commodity name in the model
  unit = "Mtoe", # for own reference
  exp = data.frame(
    region = rep(c("R1", "R2"), each = 2), # export region(s)
    year = rep(c(2020, 2050)), # export years
    price = 500, # export price in MUSD/Mtoe (USD/t),
    exp.up = rep(c(1e3, 1e4), each = 2), # upper bound for export in each year
    exp.lo = rep(c(5e2, 0), each = 2) # lower bound for export in each year
  )
)
draw(EXPOIL)
```

`newHorizon` *Create a new object of class 'horizon'*

Description

The function creates a new object of class 'horizon' that represents the planning horizon of the model/scenario.

Usage

```
newHorizon(
  period = NULL,
  intervals = NULL,
  mid_is_end = FALSE,
  mid_is_start = FALSE,
  force_BY_interval_to_1_year = TRUE,
  desc = NULL,
  name = NULL
)

## S4 method for signature 'horizon'
update(object, ..., warn_nodata = TRUE)

## S4 method for signature 'config'
setHorizon(obj, period, ...)

## S4 method for signature 'config'
update(object, ..., warn_nodata = TRUE)
```

Arguments

period	(optional) integer vector with a range or a sequence of years to define the full period of the model/scenario. If not provided, the range of 'intervals' will be used.
intervals	(optional) either data.frame or integer vector. The data.frame must have start, mid, and end columns with modeled interval. The vector will be considered as lengths of each modeled interval in period.
mid_is_end	logical, if TRUE, the mid-year will be set to the end of the interval.
mid_is_start	logical, if TRUE, the mid-year will be set to the start of the interval.
force_BY_interval_to_1_year	logical, if TRUE (default), the base-year (first) interval will be forced to one year.
desc	character. Description of the horizon object, for own references.
name	character. Name of the horizon object. Used to distinguish between different horizons in the model or scenario, including the automatic creation of the folder name for the model/scenario scripts.
horizon	a new horizon object to be set.

Value

An object of class 'horizon'

Examples

```
newHorizon(2020:2050)
newHorizon(2020:2030, desc = "One-year intervals")
newHorizon(2020:2030, c(1, 2, 5, 10), desc = "Different length intervals")
newHorizon(2020:2035, c(1, 2, 5, 5, 5))
newHorizon(2020:2050, c(1, 2, 5, 7, 1))
```

```
newHorizon(intervals = data.frame(
  start = c(2030, 2031, 2034),
  mid =   c(2030, 2032, 2037),
  end =   c(2030, 2033, 2040)),
  desc = "Explicit assignment of intervals via data.frame"
  )

newHorizon(period = 2020:2050,
           intervals = data.frame(
             start = c(2030, 2031, 2034),
             mid =   c(2030, 2032, 2037),
             end =   c(2030, 2033, 2040)),
             desc = "The period will be trimmed to the scope of intervals")

newHorizon(2020:2050, c(3, 2, 5, 10),
           desc = "Pay attention to the length of the first interval")

newHorizon(period = 2020:2040,
           intervals = data.frame(
             start = c(2030, 2032, 2035),
             mid =   c(2031, 2033, 2037),
             end =   c(2032, 2034, 2040)))
```

`newImport` *Create new export object*

Description

Import object to represent commodity import from the Rest of the World (RoW).

Usage

```
newImport(
  name,
  desc = "",
  commodity = "",
  unit = NULL,
  reserve = Inf,
  imp = data.frame(),
  misc = list(),
  ...
)
```

Arguments

`name`	character. Name of the import object, used in sets.
`desc`	character. Description of the import object.
`commodity`	character. Name of the imported commodity.
`unit`	character. Unit of the imported commodity.
`reserve`	numeric. Total accumulated limit through the model horizon.
`imp`	data.frame. Import parameters.

region character. Region name to apply the parameter; use NA to apply to all regions.

year integer. Year to apply the parameter; use NA to apply to all years.

slice character. Time slice to apply the parameter; use NA to apply to all slices.

imp.lo numeric. Lower bound on the import volume.

imp.up numeric. Upper bound on the import volume.

imp.fx numeric. Fixed import volume, ignored if NA. This parameter overrides `imp.lo` and `imp.up`.

`misc` list. Additional information.

Details

Constructor for import object.

Import object adds an "external" source of commodity to the model. The RoW is not modeled explicitly as a region, `export` and `import` objects define and control the exchange with the RoW. The operation is similar to the `demand` object, but the two ideas distinguishes between internal and external final consumption. This exchange can be exogenously defined (`imp.fx`) or optimized by the model within the given limits (`imp.lo`, `imp.up`). The `price` column is used to define the price of the imported commodity. "Reserve" sets the total amount that can be imported over the model horizon.

Value

import object with given specifications.

Examples

```
IMPOIL <- newImport(
  name = "IMPOIL", # used in sets
  desc = "Oil import to the model to the RoW", # for own reference
  commodity = "OIL", # must match the commodity name in the model
  unit = "Mtoe", # for own reference
  imp = data.frame(
    region = rep(c("R1", "R2"), each = 2), # import region(s)
    year = rep(c(2020, 2050)), # import years
    price = 600, # import price in MUSD/Mtoe (USD/t),
    imp.up = rep(c(1e4, 1e6), each = 2), # upper bound for import in each year
    imp.lo = rep(c(1e4, 1e5), each = 2) # lower bound for import in each year
  )
)
draw(IMPOIL)
```

`newModel` *Create new model object*

Description

Create new model object

Usage

```
newModel(name = "", desc = "", ...)

## S4 method for signature 'model'
setHorizon(obj, ...)

## S4 method for signature 'model'
getHorizon(obj)
```

Arguments

name	name of the model
...	configuration parameters (see class config) and model elements (classes commodity, technology, etc.)

Value

model object containing model elements (@data) and configuration (@config)

Examples

```
## Not run:
mod <- newModel(
  name = "MyModel",
  desc = "My first model",
  data = model_repository,
  discount = 0.05,
  horizon = newHorizon(period = 2020:2050,
                       intervals = rep(5, 10)),
  calendar = calendars$d365h24
  )

## End(Not run)
```

---

newRegistry *Create a new registry object.*

Description

Create a new registry object to store records of scenarios, models, and repositories. **[Experimental]**

Usage

```
newRegistry(
  class = c("scenario", "model", "repository"),
  name = NULL,
  registry_env = ".GlobalEnv",
  store_env = ".scen"
)
```

Arguments

class	character, type of the classes to be stored in the registry.
name	character, name of the registry object.
registry_env	character, environment to store the registry object.
store_env	character, environment to store the objects.

Examples

```
# The `registry` methods are in development.
```

---

newRepository *A constructor for the repository class*

Description

Repository class is used to store the model 'bricks' such as commodity, technology, supply, demand, trade, import, export, trade, storage, etc. Calendars, settings, and configurations cannot be stored in the repository, they have separate slots in model or scenario objects.

Usage

```
newRepository(
  name = "base_repository",
  ...,
  desc = NA_character_,
  misc = list()
)
```

Arguments

name	character. Name of the repository.
...	list. Model objects ("bricks"), e.g., technologies, constraints, costs, etc., stored in with their names as keys, or gropped in named lists.
desc	character. Description of the repository.
misc	list. Any additional data or information to store in the object.

See Also

Other repository model data: class-repository

newScenario *Generate a new scenario object*

Description

Generate a new scenario object

Usage

```
newScenario(
  name,
  model = NULL,
  path = fp(get_scenarios_path(), name),
  ...,
  env_name = ".scen",
  registry = get_registry(),
  replace = FALSE
)
```

Arguments

name	character. Name of the scenario object, for reference, also used in scenario path-functions.
path	character. Path to the scenario folder with the model data and scripts.
...	any model objects or settings to be assigned to the scenario.
env_name	character. Name of the environment to register the scenario. Default is ".scen". Used only if registry is provided. (in development)
registry	optional registry object to register the scenario. (in development)
replace	logical. If TRUE, replace the entry of the scenario in the registry if the entry already exists. (in development)

Value

A new scenario object.

Examples

```
# It is suggested to use `interpolate(model)` or `solve(model)` to create a new scenario.
```

newStorage *Create new storage object*

Description

Storage type of technological processes with accumulating capacity of a commodity.

Usage

```
newStorage(
  name = "",
  desc = "",
  commodity = character(),
  aux = data.frame(),
  region = character(),
  start = data.frame(),
  end = data.frame(),
  olife = data.frame(),
  charge = data.frame(),
  seff = data.frame(),
  aeff = data.frame(),
  af = data.frame(),
  fixom = data.frame(),
  varom = data.frame(),
  invcost = data.frame(),
  capacity = data.frame(),
  cap2stg = 1,
  fullYear = TRUE,
  weather = data.frame(),
  optimizeRetirement = FALSE,
  misc = list(),
  ...
)
```

Arguments

name	character. Name of the storage (used in sets).
desc	character. Description of the storage.
commodity	character. Name of the stored commodity.
aux	data.frame. Auxiliary commodities.
	acomm character. Name of the auxiliary commodity (used in sets).
	unit character. Unit of the auxiliary commodity.
region	character. Region where the storage technology exists or can be installed.
start	data.frame. Start year when the storage is available for installation.
	region character. Regions where the storage is available for investment.
	start integer. The first year when the storage is available for investment.
end	data.frame. Last year when the storage is available for investment.
	region character. Region name to apply the parameter, NA for every region.
	end integer. The last year when the storage is available for investment.
olife	data.frame. Operational life of the storage technology, applicable to the new investment only, the operational life (retirement) of preexiting capacity is described in the stock slot.
	region character. Region name to apply the parameter, NA for every region.
	olife integer. Operational life of the storage technology in years.
charge	data.frame. Pre-charged level at the beginning of the operational cycle.
	region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice for which the charged level will be specified.

charge numeric. Pre-charged or targeted level at the specified slice.

`seff` data.frame. Storage efficiency parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

stgeff numeric. Storage decay annual rate.

inpeff numeric. Input efficiency rate.

outeff numeric. Output efficiency rate.

`aeff` data.frame. Auxiliary commodities efficiency parameters.

acomm character. Name of the auxiliary commodity (used in sets).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

stg2ainp numeric. Storaage-level-to-auxiliary-input-commodity coefficient (multiplier).

cinp2ainp numeric. Input-commodity-to-auxiliary-input-commodity coefficient (multiplier).

cout2ainp numeric. Output-commodity-to-auxiliary-input-commodity coefficient (multiplier).

stg2aout numeric. Storage-level-to-auxiliary-output-commodity coefficient (multiplier).

cinp2aout numeric. Input-commodity-to-auxiliary-output-commodity coefficient (multiplier).

cout2aout numeric. Output-commodity-to-auxiliary-output-commodity coefficient (multiplier).

cap2ainp numeric. Capacity-to-auxiliary-input-commodity coefficient (multiplier).

cap2aout numeric. Capacity-to-auxiliary-output-commodity coefficient (multiplier).

ncap2ainp numeric. New-capacity-to-auxiliary-input-commodity coefficient (multiplier).

ncap2aout numeric. New-capacity-to-auxiliary-output-commodity coefficient (multiplier).

ncap2stg numeric. New-capacity-to-storage-level coefficient (multiplier).

`af` data.frame. Availability factor parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

af.lo numeric. Lower bound of the availability factor.

af.up numeric. Upper bound of the availability factor.

af.fx numeric. Fixed value of the availability factor. This parameter overrides `af.lo` and `af.up`.

cinp.lo numeric. Lower bound of the input commodity availability factor.

cinp.up numeric. Upper bound of the input commodity availability factor.

cinp.fx numeric. Fixed value of the input commodity availability factor. This parameter overrides `cinp.lo` and `cinp.up`.

cout.lo numeric. Lower bound of the output commodity availability factor.

cout.up numeric. Upper bound of the output commodity availability factor.

cout.fx numeric. Fixed value of the output commodity availability factor. This parameter overrides `cout.lo` and `cout.up`.

`fixom` data.frame. Fixed operation and maintenance cost.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

fixom numeric. Fixed operation and maintenance cost for the specified sets.

`varom` data.frame. Variable operation and maintenance cost.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

inpcost numeric. Costs associated with the input commodity.

outcost numeric. Costs associated with the output commodity.

stgcost numeric. Costs associated with the storage level.

`invcost` data.frame. Investment cost.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

invcost numeric. Overnight investment cost for the specified region and year.

wacc numeric. Weighted average cost of capital. If not supplied, the discount from the model or scenario is used. (currently ignored)

`capacity` data.frame. Capacity parameters of the storage technology.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

cap numeric. Capacity of the storage technology.

cap.lo numeric. Lower bound of the storage capacity.

cap.up numeric. Upper bound of the storage capacity.

cap.fx numeric. Fixed value of the storage capacity. This parameter overrides `cap.lo` and `cap.up`.

ncap.lo numeric. Lower bound of the new storage capacity.

ncap.up numeric. Upper bound of the new storage capacity.

ncap.fx numeric. Fixed value of the new storage capacity. This parameter overrides `ncap.lo` and `ncap.up`.

ret.lo numeric. Lower bound of the storage capacity retirement.

ret.up numeric. Upper bound of the storage capacity retirement.

ret.fx numeric. Fixed value of the storage capacity retirement. This parameter overrides `ret.lo` and `ret.up`.

`cap2stg` numeric. Charging and discharging capacity to the storing capacity inverse ratio. Can be used to define the storage duration.

`fullYear` logical. If TRUE (default), the storage technology operates between parent timeframes through the year. The last time-slice in the timeframe is used as a preciding time-slice for the first time-slice in the the same group of time-slices within the parent timeframe. if FALSE, the storage charge and discchare cycle is limited to the parent timeframe. The last time-slice in the timeframe is used as a preciding time-slice for the first time-slice in the the same group of time-slices within the parent timeframe.

`weather` data.frame. Weather factors multipliers.

weather character. Name of the weather factor to apply.

waf.lo numeric. Coefficient that links the weather factor with the lower bound of the availability factor.

waf.up numeric. Coefficient that links the weather factor with the upper bound of the availability factor.

waf.fx numeric. Coefficient that links the weather factor with the fixed value of the availability factor. This parameter overrides `waf.lo` and `waf.up`.

wcinp.lo numeric. Coefficient that links the weather factor with the lower bound of the input commodity availability factor.

wcinp.up numeric. Coefficient that links the weather factor with the upper bound of the input commodity availability factor.

wcinp.fx numeric. Coefficient that links the weather factor with the fixed value of the input commodity availability factor. This parameter overrides `wcinp.lo` and `wcinp.up`.

wcout.lo numeric. Coefficient that links the weather factor with the lower bound of the output commodity availability factor.

wcout.up numeric. Coefficient that links the weather factor with the upper bound of the output commodity availability factor.

wcout.fx numeric. Coefficient that links the weather factor with the fixed value of the output commodity availability factor. This parameter overrides `wcout.lo` and `wcout.up`.

`optimizeRetirement` logical. Incidates if the retirement of the storage should be optimized. Also requires the same parameter in the `model` or `scenario` class to be set to TRUE to be effective.

`misc` list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the storage data, or other metadata.

Details

Storage can be used in combination with other processes, such as technologies, supply, or demand to represent complex technological chains, demand or supply technologies with time-shift. Operation of storage includes accumulation, storing, and release of the stored commodity. The storing cycle operates on the ordered time-slices of the commodity timeframe. The cycle is looped either on an annual basis (last time-slice of a year follows the first time slice of the same year) or within the parent time-frame (for example, when commodity time-frame is "HOUR" and the parent time-frame is "DAY" then the storage cycle will be a calendar day).

Value

storage object

Examples

```
STG1 <- newStorage(
  name = "STG1",
  desc = "Storage description",
  commodity = "electricity",
  region = "R1",
  start = data.frame(region = "R1", start = 0),
```

```
  end = data.frame(region = "R1", end = 1),
  olife = data.frame(region = "R1", olife = 20),
  charge = data.frame(
    # region = "R1",
    year = 2020,
    # slice = "HOUR",
    charge = 0.1
  ),
  seff = data.frame(
    # region = "R1",
    # year = 2020,
    # slice = "HOUR",
    stgeff = 0.999,
    inpeff = 0.9,
    outeff = 0.9
  ),
  aeff = data.frame(
    acomm = "electricity",
    region = "R1",
    year = 2020,
    # slice = "HOUR",
    stg2ainp = 0.9,
    cinp2ainp = 0.1,
    cout2ainp = 0.2,
    stg2aout = 0.9,
    cinp2aout = 0.9,
    cout2aout = 0.9,
    cap2ainp = 0.9,
    cap2aout = 0.9,
    ncap2ainp = 0.9,
    ncap2aout = 0.9,
    ncap2stg = 0.9
  ),
  af = data.frame(
    region = "R1", year = 2020, slice = "HOUR",
    af.lo = 0.9, af.up = 0.9, af.fx = 0.9, cinp.up = 0.9,
    cinp.fx = 0.9, cinp.lo = 0.9, cout.up = 0.9,
    cout.fx = 0.9, cout.lo = 0.9
  ),
  fixom = data.frame(region = "R1", year = 2020, fixom = 0.9),
  varom = data.frame(
    region = "R1", year = 2020, slice = "HOUR",
    inpcost = 0.9, outcost = 0.9, stgcost = 0.9
  ),
  invcost = data.frame(
    region = "R1", year = 2020, invcost = 0.9,
    wacc = 0.9, retcost = 0.9
  ),
  capacity = data.frame(
    region = "R1", year = 2020, stock = 0.9,
    cap.lo = 0.9, cap.up = 0.9, cap.fx = 0.9, ncap.lo = 0.9,
    ncap.up = 0.9, ncap.fx = 0.9, ret.lo = 0.9, ret.up = 0.9,
    ret.fx = 0.9
  ),
  cap2stg = 1,
  fullYear = TRUE,
  weather = data.frame(
```

```
  weather = "sunny",
  waf.lo = 0.9,
  waf.up = 0.9,
  waf.fx = 0.9, wcinp.lo = 0.9,
  wcinp.fx = 0.9, wcinp.up = 0.9, wcout.lo = 0.9, wcout.fx = 0.9,
  wcout.up = 0.9
),
optimizeRetirement = FALSE,
misc = list()
)
```

`newSubsidy` *Create a new subsidy object*

Description

Subsidies are used to represent the financial support provided to production, consumption, or balance of a commodity.

Usage

```
newSub(
  name,
  desc = "",
  comm = "",
  region = character(),
  defVal = 0,
  sub = data.frame(),
  misc = list(),
  ...
)
```

Arguments

`name`	character. Name of the subsidy object, used in sets.
`desc`	character. Description of the subsidy object.
`comm`	character. Name of the subsidized commodity.
`region`	character. Region where the subsidy is applied.
`defVal`	numeric. Default value of the subsidy.
`sub`	data.frame. Subsidy values. **region** character. Region name to apply the parameter, NA for every region. **year** integer. Year to apply the parameter, NA for every year. **slice** character. Time slice to apply the parameter, NA for every slice. **inp** numeric. Input subsidy, e.g., per unit of commodity consumed by all processes. **out** numeric. Output subsidy, e.g., per unit of commodity produced by all processes. **bal** numeric. Balance subsidy, e.g., per unit of commodity balance (production - consumption).
`misc`	list. Any additional information or data to store in the subsidy object.

Value

An object of class sub

See Also

Other class constraint policy: class-constraint, class-costs, newConstraint(), newCosts(), newTax(), subsidy-class, tax-class

Examples

```
SUB_BIO <- newSub(
 name = "SUB_BIO", # used in sets
 desc = "Biofuel consumption subsidy", # for own reference
 comm = "BIO", # must match the commodity name in the model
 region = "R1", # region where the subsidy is applied
 defVal = 0, # default value
 sub = data.frame(
    # region = "R1",
    year = 2025:2030,
    inp = 0.9 # subsidy rate
   )
 )
```

---

newSupply *Constructor for supply object.*

Description

Creates an instance of the supply class and initializes it with the given data and parameters.

Usage

```
newSupply(
  name = "",
  desc = "",
  commodity = character(),
  unit = character(),
  weather = data.frame(),
  reserve = data.frame(),
  availability = data.frame(),
  region = character(),
  misc = list(),
  ...
)
```

Arguments

name	character. Name of the supply object, used in sets.
desc	character. Description of the supply object.
commodity	character. The supplied commodity short name.
unit	character. The main unit of the commodity used in the model.

`weather` data.frame. Weather factors to apply to the supply.

weather character. Name of the weather factor to apply. Must match the weather factor names in a `weather` class in the model.

wava.lo numeric. Coefficient that links the weather factor with the lower bound of the availability factor `ava.lo`.

wava.up numeric. Coefficient that links the weather factor with the upper bound of the availability factor `ava.up`.

wava.fx numeric. Coefficient that links the weather factor with the fixed value of the availability factor `ava.fx`. This parameter overrides `wava.lo` and `wava.up`.

`reserve` data.frame. Total available resource. Applicable to exhaustible resources. Set for each region. If not set, the resource is considered infinite.

region character. Region name to apply the parameter. Use NA to apply to all regions.

res.lo numeric. Lower bound of the total available resource.

res.up numeric. Upper bound of the total available resource.

res.fx numeric. Fixed value of the total available resource. This parameter overrides `res.lo` and `res.up`.

`availability` data.frame. Availability of the resource in physical units.

region character. Region name to apply the parameter. Use NA to apply to all regions.

year integer. Year to apply the parameter. Use NA to apply to all years.

slice character. Time slice to apply the parameter. Use NA to apply to all slices.

ava.lo numeric. Lower bound of the availability factor.

ava.up numeric. Upper bound of the availability factor.

ava.fx numeric. Fixed value of the availability factor. This parameter overrides `ava.lo` and `ava.up`.

cost numeric. Cost of the resource extraction, if not set, the resource is considered free.

`region` character. Regions where the supply process exists. Must include all regions used in other slots. `availability` and `reserve` slots also limit possible regions.

`misc` list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the supply data, or other metadata.

Details

The `supply` class is used to add a domestic source of a commodity to the model, with given reserves, availability, and costs.

Value

supply object with given specifications.

Examples

```
SUP_COA <- newSupply(
   name = "SUP_COA",
   desc = "Coal supply",
```

```
   commodity = "COA",
   unit = "PJ",
   reserve = data.frame(
      region = c("R1", "R2", "R3"),
      res.up = c(2e5, 1e4, 3e6) # total reserves/deposits
   ),
   availability = data.frame(
      region = c("R1", "R2", "R3"),
      year = NA_integer_,
      slice = "ANNUAL",
      ava.up = c(1e3, 1e2, 2e2), # annual availability
      cost = c(10, 20, 30) # cost of the resource (currency per unit)
   ),
   region = c("R1", "R2", "R3")
 )
class(SUP_COA)
# draw(SUP_COA)
```

---

newTax *Create a new tax object*

Description

Taxes are used to represent the financial levy imposed on production, consumption, or balance of a commodity.

Usage

```
newTax(
  name,
  desc = "",
  comm = "",
  region = character(),
  defVal = 0,
  tax = data.frame(),
  misc = list(),
  ...
)
```

Arguments

name	character. Name of the tax object, used in sets.
desc	character. Description of the tax object.
comm	character. Name of the taxed commodity.
region	character. Region where the tax is applied.
defVal	numeric. Default value of the tax for not specified sets, 0 if not specified.
tax	data.frame. Tax values. **region** character. Region name to apply the parameter, NA for every region. **year** integer. Year to apply the parameter, NA for every year. **slice** character. Time slice to apply the parameter, NA for every slice.

inp numeric. Input tax, e.g., per unit of commodity consumed by all processes.

out numeric. Output tax, e.g., per unit of commodity produced by all processes.

bal numeric. Balance tax, e.g., per unit of commodity balance (production - consumption).

`misc` list. Additional information.

Value

An object of class `tax`

See Also

Other class constraint policy: `class-constraint`, `class-costs`, `newConstraint()`, `newCosts()`, `newSubsidy()`, `subsidy-class`, `tax-class`

Examples

```
CO2TAX <- newTax(
 name = "CO2TAX",
 desc = "Tax on net CO2 emissions",
 comm = "CO2",
 region = "R1",
 defVal = 0,
 tax = data.frame(
 # region = "R1", # not required when @region is set
 year = c(2030, 2040, 2050),
 bal =  c(10, 50, 200) # $10, $50, $200 per ton, will be interpolated
 # out = ... use to tax output commodity
 # inp = ... use to tax input commodity
   ),
 misc = list(
  source = "https://www.example.com/tax"
  )
 )
```

`newTechnology` *Create a new "technology" object.*

Description

This function initializes and returns an S4 object of class `technology`, representing a specific technology with given attributes. The function has the same arguments as slot-names in the `technology` class. Every argument has a specific format as described below and in the class documentation.

Usage

```
newTechnology(
  name = "",
  desc = "",
  input = data.frame(),
  output = data.frame(),
  group = data.frame(),
```

```
  aux = data.frame(),
  units = data.frame(),
  cap2act = as.numeric(1),
  geff = data.frame(),
  ceff = data.frame(),
  aeff = data.frame(),
  af = data.frame(),
  afs = data.frame(),
  weather = data.frame(),
  capacity = data.frame(),
  invcost = data.frame(),
  fixom = data.frame(),
  varom = data.frame(),
  olife = data.frame(),
  region = character(),
  start = data.frame(),
  end = data.frame(),
  timeframe = character(),
  fullYear = TRUE,
  optimizeRetirement = FALSE,
  misc = list(),
  ...
)

## S4 method for signature 'technology'
update(object, ...)
```

Arguments

name	character. Name of the technology, used in sets.
desc	character. Optional description of the technology for reference.
input	data.frame. Main commodities input. Main commodities are linked to the process capacity and activity. Their parameters are defined in the ceff slot.
	comm character. Name of the input commodity.
	unit character. Unit of the input commodity.
	group character. Name of input-commodities-group.
	combustion numeric. combustion factor from 0 to 1 (default 1) to calculate emissions from fuels combustion (commodities intermediate consumption, more broadly)
output	data.frame. Main commodities output. Main commodities are linked to the process capacity and activity. Their parameters are defined in the ceff slot.
	comm character. Name of the output commodity.
	unit character. Unit of the output commodity.
	group character. Name of output-commodities-group.
group	data.frame. Details for commodity groups if defined in input and output slots (for reference).
	group character. Name of the group. Must match the group names in the input and output slots.
	desc character. Description of the group.
	unit character. Unit of the group.

`aux` data.frame. Auxilary commodities, both input and output, their parameters are defined in the `aeff` slot.

acomm character. Name of the auxilary commodity.

unit character. Unit of the auxilary commodity.

`units` data.frame. Key units of the process activity and capacity (for reference).

capacity character. Unit of capacity

use character. Unit of 'use' (grouped input) if applicable.

activity character. Unit of activity variable of the technology.

costs character. Currency of costs variable of the technology.

`cap2act` numeric. Capacity to activity ratio. Default is 1. Specifies how much product (activity, or output commodity if identical) will be produced per unit of capacity.

`geff` data.frame. Input-commodity-group efficiency parameters.

region character. Name of region to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

group character. Name of group to apply the parameter. Required, must match the group names in the input and output slots.

ginp2use numeric. Group-input-to-use coefficient, default is 1.

`ceff` data.frame. Main commodity and activity efficiency parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

comm character. Name of commodity to apply the parameter, different parameters require specification either input or output commodity.

cinp2use numeric. Commodity-input-to-use coefficient, default is 1.

use2cact numeric. Use-to-commodity-activity coefficient, default is 1.

cact2cout numeric. Commodity-activity-to-commodity-output coefficient, default is 1.

cinp2ginp numeric. Commodity-input-to-group-input coefficient, default is 1.

share.lo numeric. Lower bound on a share of commodity within a group, default is 0.

share.up numeric. Upper bound on a share of commodity within a group, default is 1.

share.fx numeric. Fixed share of commodity within a group, ignored if NA. This parameter overrides `share.lo` and `share.up`.

afc.lo numeric. Lower bound on the physical value of the commodity, ignored if NA.

afc.up numeric. Upper bound on the physical value of the commodity, ignored if NA.

afc.fx numeric. Fixed physical value of the commodity, ignored if NA. This parameter overrides `afc.lo` and `afc.up`.

`aeff` data.frame. Parameters linking main commodities, activities, and capacities to auxiliary commodities.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

acomm character. Name of auxilary commodity to apply the parameter.

cinp2ainp numeric. Main-commodity-input-to-auxilary-commodity-input coefficient, ignored if NA.

cinp2aout numeric. Main-commodity-input-to-auxilary-commodity-output coefficient, ignored if NA.

cout2ainp numeric. Main-commodity-output-to-auxilary-commodity-input coefficient, ignored if NA.

cout2aout numeric. Main-commodity-output-to-auxilary-commodity-output coefficient, ignored if NA.

act2ainp numeric. Technology-activity-to-auxilary-commodity-input coefficient, ignored if NA.

act2aout numeric. Technology-activity-to-auxilary-commodity-output coefficient, ignored if NA.

cap2ainp numeric. Technology-capacity-to-auxilary-commodity-input coefficient, ignored if NA.

cap2aout numeric. Technology-capacity-to-auxilary-commodity-output coefficient, ignored if NA.

ncap2ainp numeric. Technology-new-capacity-to-auxilary-commodity-input-coefficient, ignored if NA.

ncap2aout numeric. Technology-new-capacity-to-auxilary-commodity-output coefficient, ignored if NA.

`af` data.frame. Timeslice-level availability factor parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, NA for every slice.

af.lo numeric. Lower bound on the availability factor, default is 0.

af.up numeric. Upper bound on the availability factor, default is 1.

af.fx numeric. Fixed availability factor, ignored if NA. This parameter overrides `af.lo` and `af.up`.

rampup numeric. Ramping-up time constraint RHS value, ignored if NA. Depends on the technology timeframe.

rampdown numeric. Ramping-down time constraint RHS value, ignored if NA. Depends on the technology timeframe.

`afs` data.frame. Timeframe-level availability factor constraints.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of slice to apply the parameter, required.

afs.lo numeric. Lower bound on the availability factor for the timeframe, default is 0.

afs.up numeric. Upper bound on the availability factor for the timeframe, default is 1.

afs.fx numeric. Fixed availability factor for the timeframe, ignored if NA. This parameter overrides `afs.lo` and `afs.up`.

`weather` data.frame. Parameters linking `weather factors` (external shocks specified by `weather` class) to the availability parameters `af`, `afs`, and `afc`.

weather character. Name of the applied weather factor, required, must match the weather factor names in a `weather` class in the model.

comm character. Name of the commodity with specified `afc.*` to be affected by the weather factor, required if `afc.*` parameters are specified.

wafc.lo numeric. Multiplying coefficient to the lower bound on the commodity availability parameter `afc.lo`, ignored if NA.

wafc.up numeric. Multiplying coefficient to the upper bound on the commodity availability parameter `afc.up`, ignored if NA.

wafc.fx numeric. Multiplying coefficient to the fixed value of the commodity availability parameter `afc.fx`, ignored if NA. This parameter overrides `wafc.lo` and `wafc.up`.

waf.lo numeric. Multiplying coefficient to the lower bound on the availability factor parameter `af.lo`, ignored if NA.

waf.up numeric. Multiplying coefficient to the upper bound on the availability factor parameter `af.up`, ignored if NA.

waf.fx numeric. Multiplying coefficient to the fixed value on the availability factor parameter `af.fx`, ignored if NA. This parameter overrides `waf.lo` and `waf.up`.

wafs.up numeric. Multiplying coefficient to the upper bound on the availability factor parameter `afs.up`, ignored if NA.

wafs.lo numeric. Multiplying coefficient to the lower bound on the availability factor parameter `afs.lo`, ignored if NA.

wafs.fx numeric. Multiplying coefficient to the fixed value on the availability factor parameter `afs.fx`, ignored if NA. This parameter overrides `wafs.lo` and `wafs.up`.

`capacity` data.frame. Capacity of the installed technology (in units of capacity).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, required, values between specified years will be interpolated.

stock numeric. Predefined capacity of the technology in units of capacity, default is 0. This parameter also defines the exogenous capacity retirement (age-based), or exogenous capacity additions, not optimized by the model, and not included in investment costs.

cap.lo numeric. Lower bound on the total capacity (preexisting stock and new installations), ignored if NA.

cap.up numeric. Upper bound on the total capacity (preexisting stock and new installations), ignored if NA.

cap.fx numeric. Fixed total capacity (preexisting stock and new installations), ignored if NA. This parameter overrides `cap.lo` and `cap.up`.

ncap.lo numeric. Lower bound on the new capacity (new installations), ignored if NA.

ncap.up numeric. Upper bound on the new capacity (new installations), ignored if NA.

ncap.fx numeric. Fixed new capacity (new installations), ignored if NA. This parameter overrides `ncap.lo` and `ncap.up`.

ret.lo numeric. Lower bound on the capacity retirement (age-based), ignored if NA.

ret.up numeric. Upper bound on the capacity retirement (age-based), ignored if NA.

ret.fx numeric. Fixed capacity retirement (age-based), ignored if NA. This parameter overrides `ret.lo` and `ret.up`.

`invcost` data.frame. Total overnight investment costs of the project (per unit of capacity).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

invcost numeric. Total overnight investment costs of the project (per unit of capacity), default is 0.

wacc numeric. Weighted average cost of capital, (currently ignored).

`fixom` data.frame. Fixed operational and maintenance cost (per unit of capacity a year).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

fixom numeric. Fixed operational and maintenance cost, default is 0.

`varom` data.frame. Variable operational and maintenance cost (per unit of activity or commodity).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Name of the time-slice or (grand-)parent timeframe to apply the parameter, NA for every time-slice of the technology timeframe.

varom numeric. Variable operational and maintenance cost per unit of activity, default is 0.

comm character. Name of the commodity for which the parameter will be applied, required for `cvarom` parameter.

cvarom numeric. Variable operational and maintenance cost per unit of commodity, default is 0.

acomm character. Name of the auxilary commodity for which the `avarom` will be applied, required for `avarom` parameter.

avarom numeric. Variable operational and maintenance cost per unit of auxilary commodity, default is 0.

`olife` data.frame. Operational life of the installed technology (in years).

region character. Region name to apply the parameter, NA for every region.

olife integer. Operational life of the technology if installed during optimization, in years, default is 1.

`region` character. Vector of regions where the technology exists or can be installed. Optional. If not specified, the technology is applied to all regions. If specified, must include all regions used in other slots.

`start` data.frame. The first year the technology can be installed.

region character. Region name to apply the parameter, NA for every region.

start integer. The first year the technology can be installed, NA means all years of the modeled horizon.

`end` data.frame. The last year the technology can be installed.

region character. Region name to apply the parameter, NA for every region.

end integer. The last year the technology can be installed, default is Inf.

`timeframe` character. Name of timeframe level the technology is operating. By default, the lowest level of timeframe of commodities used in the technology is applied.

`fullYear` logical. Incidates if the technology is operating on a full-year basis. Used in storages. currently ignored for technologies.

optimizeRetirement
: logical. Incidates if the retirement of the technology should be optimized. Also requires the same parameter in the model or scenario class to be set to TRUE to be effective.

misc
: list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the technology data, or other metadata.

...
: slot-names with data to update (see newTechnology)

object
: object of class technology

Value

An object of class technology.

Examples

```
ECOAL <- newTechnology(
  name = "ECOAL", # name, used in sets, no white spaces or special characters
  desc = "Generic coal power plant", # any description of the technology
  input = data.frame(
    comm = "COAL", # name of input commodity
    unit = "MMBtu", # unit of the input commodity
    # combustion factor from 0 to 1 (default 1) to calculate emissions
    # from fuels combustion (commodities intermediate consumption, more broadly)
    combustion = 1
  ),
  output = data.frame(
    comm = "ELC", # name of output commodity
    unit = "MWh" # unit of the output commodity
  ),
  aux = data.frame(
    acomm = c("NOx", "SO2", "Hg"), # names of auxilary commodities
    unit = c("kg", "kg", "g") # units
  ),
  # Capacity to activity ration: 8760 MWh output a year per MW of capacity
  cap2act = 8760,
  ceff = data.frame( # efficiency parameters for the main commodities
    comm = "COAL",
    # efficiency, 1/10 MWh per MMBtu, inverse heat rate
    # check: 1 / convert(10, "MMBtu", "MWh") ~= 34% efficiency
    cinp2use = 1 / 10
  ),
  aeff = data.frame( # paramaters for the auxilary commodities
    acomm = c("NOx", "SO2", "Hg"),
    act2aout = c(0.1, 0.2, 0.3) # emission factors, linked to activity
  ),
  af = data.frame( # availability (capacity) factor by time slices
    af.up = 0.95 # maximum 95% per hour
  ),
  afs = data.frame( # availability factor by timeframes
    slice = "ANNUAL", # annual availability factor
    afs.lo = 0.40, # at least 40% per year
    afs.up = 0.85 # maximum 85% per year
  ),
  fixom = data.frame( # fixed operational and maintenance cost
    region = c("R1", "R2", NA), # regions, NA - all other regions
```

```
    fixom = c(100, 200, 150) # MW a year
  ),
  varom = data.frame( # variable operational and maintenance cost
    region = c("R1", "R2"), # regions
    varom = c(1, 2) # $1 and $2 per MWh
  ),
  invcost = data.frame( # investment cost
    year = c(2020, 2030, 2040), # to differentiate by years
    invcost = c(1000, 900, 800) # $1000, $900, $800 per MW
  ),
  start = data.frame( # start year
    start = 2020 # can be installed from 2020
  ),
  end = data.frame( # end year
    end = 2040 # can be installed until 2040
  ),
  olife = data.frame( # operational life
    olife = 30 # years
  ),
  capacity = data.frame( # existing capacity
    year = c(2020, 2030, 2040), # to differentiate by years
    region = c("R1"), # exists only in R1
    stock = c(300, 200, 100) # age-based exogenous retirement
  ),
  # regions where the technology can be installed
  region = c("R1", "R2", "R5", "R7"),
)
draw(ECOAL)
```

---

newTrade *Create new trade object*

Description

Constructor for trade object.

Usage

```
newTrade(
  name = "",
  desc = "",
  commodity = character(),
  routes = data.frame(),
  trade = data.frame(),
  fixom = data.frame(),
  varom = data.frame(),
  invcost = data.frame(),
  olife = data.frame(),
  start = data.frame(start = -Inf, stringsAsFactors = FALSE),
  end = data.frame(end = Inf, stringsAsFactors = FALSE),
  capacity = data.frame(),
  capacityVariable = TRUE,
```

```
  aux = data.frame(),
  aeff = data.frame(),
  cap2act = 1,
  optimizeRetirement = FALSE,
  misc = list(),
  ...
)
```

Arguments

name	character. Name of the trade object, used in sets.
desc	character. Description of the trade object.
commodity	character. The traded commodity short name.
routes	data.frame. Source and destination regions. For bivariate trade define both directions in separate rows. **from** character. Source region. **to** character. Destination region.
trade	data.frame. Technical parameters of trade. **region** character. Region name to apply the parameter, NA for every region. **year** integer. Year to apply the parameter, NA for every year. **slice** character. Time slice to apply the parameter, NA for every slice. **trade** numeric. Trade volume.
fixom	data.frame. (not implemented!) Fixed operation and maintenance costs.
varom	data.frame. (not implemented!) Variable operation and maintenance costs.
invcost	data.frame. Investment cost, used when capacityVariable is TRUE. **region** character. Region name to apply the parameter, NA for every region. **year** integer. Year to apply the parameter, NA for every year. **invcost** numeric. Investment cost.
olife	numeric. Operational life of the trade object.
start	data.frame. Start year when the trade-type of process is available for investment. **region** character. Regions where the trade-type of process is available for investment. **start** integer. The first year when the trade-type of process is available for investment.
end	data.frame. End year when the trade-type of process is available for investment. **region** character. Region name to apply the parameter, NA for every region. **end** integer. The last year when the trade-type of process is available for investment.
capacity	data.frame. (not implemented!) Capacity parameters of the trade object.
capacityVariable	logical. If TRUE, the capacity variable of the trade object is optimized. If FALSE, the capacity is defined by availability parameters (`ava.*`) in the tradeflow units.
aux	data.frame. Auxiliary commodity of trade. **acomm** character. Name of the auxiliary commodity (used in sets). **unit** character. Unit of the auxiliary commodity.

`aeff` data.frame. Auxiliary commodity efficiency parameters.

acomm character. Name of the auxiliary commodity (used in sets).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

trade2ainp numeric. Trade-to-auxiliary-input-commodity coefficient (multiplier).

trade2aout numeric. Trade-to-auxiliary-output-commodity coefficient (multiplier).

`cap2act` numeric. Capacity to activity ratio.

`optimizeRetirement` logical. Incidates if the retirement of the trade object should be optimized. Also requires the same parameter in the `model` or `scenario` class to be set to TRUE to be effective.

`misc` list. Additional information.

Details

Trade objects are used to represent inter-regional exchange in the model. Without trade, every region is isolated and can only use its own resources. The class defines trade routes, efficiency, costs, and other parameters related to the process. Number of routes per trade object is not limited. One trade object can have a part or entire trade network of the model. However, it has a distinct name and all the routs will be optimized together. Create separate trade objects to optimize different parts of the trade network (aka transmission lines).

Value

trade object with given specifications.

Examples

```
PIPELINE1 <- newTrade(
  name = "PIPELINE1",
  desc = "Some transport pipeline",
  commodity = "OIL",
  routes = data.frame(
    src = c("R1", "R2"),
    dst = c("R2", "R3")
  ),
  trade = data.frame(
    src = c("R1", "R2"),
    dst = c("R2", "R3"),
    teff = c(0.99, 0.98)
  ),
  olife = list(olife = 60)
)
draw(PIPELINE1)

PIPELINE2 <- newTrade(
  name = "PIPELINE2",
  desc = "Some transport pipeline",
  commodity = "OIL",
  routes = data.frame(
```

```
    src = c("R1", "R1", "R2", "R3"),
    dst = c("R2", "R3", "R3", "R2")
  ),
  trade = data.frame(
    src = c("R1", "R1", "R2", "R3"),
    dst = c("R2", "R3", "R3", "R2"),
    teff = c(0.912, 0.913, 0.923, 0.932)
  ),
  aux = data.frame(
    acomm = c("ELC", "CH4"),
    unit = c("MWh", "kt")
  ),
  aeff = data.frame(
    acomm = c("ELC", "CH4", "ELC", "CH4"),
    src = c("R1", "R1", "R2", "R3"),
    dst = c("R2", "R2", "R3", "R2"),
    csrc2ainp = c(.5, NA, .3, NA),
    cdst2ainp = c(.4, NA, .6, NA),
    csrc2aout = c(NA, .1, NA, .2)
  ),
  olife = list(olife = 60)
)
draw(PIPELINE2, node = "R1")
draw(PIPELINE2, node = "R2")
draw(PIPELINE2, node = "R3")
```

`newWeather` *Create new weather object*

Description

`weather` is a data-carrying class with exogenous shocks used to influence operation of processes in the model.

Usage

```
newWeather(
  name = "",
  desc = "",
  unit = as.character(NA),
  region = character(),
  timeframe = character(),
  defVal = 0,
  weather = data.frame(),
  misc = list(),
  ...
)
```

Arguments

`name`	character. Name of the weather factor, used in sets.
`desc`	character. Description of the weather factor.
`unit`	character. Unit of the weather factor.

region	character. Region where the weather factor is applied.
timeframe	character. Timeframe of the weather factor.
defVal	numeric. Default value of the weather factor, 0 by default.
weather	data.frame. Weather factor values.
	region character. Region name to apply the parameter, NA for every region.
	year integer. Year to apply the parameter, NA for every year.
	slice character. Time slice to apply the parameter, NA for every slice.
	wval numeric. Weather factor value.

Details

Weather factors are separated from the model parameters and can be added or replaced for different scenarios. !!!Additional details...

Value

weather object with given specifications.

Examples

```
## Not run:

# use/make time resolution of the model: timetalbe
ttbl <- make_timetable(tsl_levels$d365_h24)
ttbl

WSOL <- newWeather(
  name = "WSOL",
  desc = "Horiontal solar PV capacity factor",
  timeframe = "HOUR",
  defVal = 0.,
  weather = data.frame(
    region = "R1",
    year = 2015, #
    slice = ttbl$slice,
    wval = runif(length(ttbl$slice), 0., 1) # use your data
  )
)

## End(Not run)
```

---

obj2mem *Loads objects from disk to memory*

Description

Loads objects from disk to memory

Usage

```
obj2mem(obj, verbose = TRUE)
```

Arguments

obj	Object of S4 class, saved on disk (scenario, model, etc.)
verbose	If TRUE, prints messages

Value

Object of the same S4 class as input object, with all of the slots loaded in memory.

Examples

```
## Not run:
obj2mem(scen_ondisk)

## End(Not run)
```

---

print,parameter-method *Print methods for the energyRt classes*

Description

Print methods for the energyRt classes

Usage

```
## S4 method for signature 'parameter'
print(x, ...)

print(x, ...)
```

---

read_solution *Read solution*

Description

The function and method read outputs of solved model/scenario and return the scenario object populated with variables data.

Usage

```
read_solution(obj, ...)

## S4 method for signature 'scenario'
read(obj, ...)
```

Arguments

obj	scenario object
...	optional tmp.dir (if missing in the scenario object or to replace the saved path)

Value

The function returns the scenario object with populated modOut slot from the solved model directory.

See Also

`solve()` to run the script, solve the scenario. `write_sc()` to read

Examples

```
## Not run:
scen <- read(scen)

## End(Not run)
```

`register` *Register an object in the registry.*

Description

Register an repository, model, or scenario object in the registry. **[Experimental]**

Usage

```
register(
  obj,
  registry,
  name = obj@name,
  project = "",
  path = "",
  memo = "",
  datetime = lubridate::now(tzone = "UTC"),
  user = Sys.info()["user"],
  system = Sys.info()["sysname"],
  ...,
  env = obj@misc$env,
  replace = FALSE
)
```

Arguments

`obj`	object to be registered.
`registry`	registry object to add the entry.
`name`	character, name of the object.
`project`	character, optional, the name of the project.
`path`	character, optional path to the object's 'onDisk' directory.
`memo`	character, optional short note about the object.
`datetime`	timestamp, optional, date and time of the registration.
`user`	character, optional, user who registered the object.

system	character, optional, system where the object is registered.
...	(reserved for future use).
env	character, environment where the object is stored.
replace	logical, if TRUE, replace the existing entry.

Examples

```
# `registry` methods are in development.
```

`renameSets` *Rename data.frame columns of list of data.frames.*

Description

Rename data.frame columns of list of data.frames.

Usage

```
renameSets(x, newNames = NULL)
```

Arguments

x	a data.frame or a list with data frames.
newNames	named character vector or list with new names as values, and old names as names.

Value

depending on input, the renamed data.frame or the list with renamed data.frames.

Examples

```
## Not run:
x <- data.frame(a = letters, n = 1:length(letters))
x
renameSets(x[1:3, ], c(a = "A", n = "N"))
renameSets(x[1:3, ], list(a = "B", n = "M"))

## End(Not run)
```

revalueSets — *Replace specified values with new values in factor or character columns of a data.frame.*

Description

Replace specified values with new values in factor or character columns of a data.frame.

Usage

```
revalueSets(x, newValues = NULL)
```

Arguments

x	vector
newValues	a names list with named vectors. The names of the list should be equal to the names of the data.frame columns in wich values will be replaced. The named vector should have new names as values and old values as names.

Value

the x data.frame with revalued variables.

Examples

```
## Not run:
x <- data.frame(a = letters, n = 1:length(letters))
nw1 <- LETTERS[1:10]
names(nw1) <- letters[1:10]
nw2 <- formatC(1:9, width = 3, flag = "0")
names(nw2) <- 1:9
newValues <- list(a = nw1, n = nw2)
newValues
revalueSets(x, newValues)

## End(Not run)
```

save_scenario — *Save scenario object on disk in parquet format using* arrow *package.*

Description

Save scenario object on disk in parquet format using arrow package.

Usage

```
save_scenario(
  scen,
  path = scen@path,
  format = "parquet",
  overwrite = TRUE,
  clean_start = FALSE,
  write_log = TRUE,
  verbose = TRUE
)
```

Arguments

scen	scenario object.
path	character. Path to scenario directory.
format	file format (currently parquet only, arrow or feather will be implemented in further releases).
overwrite	logical. Overwrite existing scenario directory.
clean_start	logical. Clean scenario directory before saving.
write_log	logical. Write (update) logfile.
verbose	logical. Print messages.

Value

scenario object with most of the slots saved on disk.

Examples

```
## Not run:
scen_BASE@path # check the scenarion directory
scen_BASE <- save_scenario(scen_BASE) # saving in the default directory

## End(Not run)
```

---

set_gams_path *Set GAMS and GDX library directory*

Description

This (optional) function sets path to GAMS directory to R-options. It might be useful if for the cases when several different version (and licenses) of GAMS installed, to easily switch between them. It is also possible to set different path for GAMS and GAMS Data Exchange (GDX) libraries. If GDX path is not set, the GAMS path will be used. If GAMS path is not set, the default system GAMS-path (OS environment variables) instead.

Usage

```
set_gams_path(path = NULL)

get_gams_path()

set_gdxlib_path(path = NULL)

get_gdxlib_path()

set_glpk_path(path = NULL)

get_glpk_path()

set_julia_path(path = NULL)

get_julia_path()

set_python_path(path = NULL)

get_python_path()
```

Arguments

path	character path to the python installation. If NULL, the global operation path is used.

Value

Sets path to GAMS library in R-options

The current path to GAMS library, set in R-options

Sets path to GDX library in R-options

The current path to GDX library, set in R-options

sets the path to the GLPK library in R options and returns NULL.

returns the path to the GLPK library.

Sets the path to Julia installation in the energyRt environment options and returns NULL.

character. Path to Julia installation.

Writes or reads the path to python installation or environment to/from `energyRt` options.

Examples

```
# set_gams_path("C:/GAMS/win64/32.2/")

# get_gams_path()
# set_gdxlib("C:/GAMS/35")
# get_gdxlib()
## Not run:
set_glpk_path("/usr/local/bin/glpk") # Linux & Mac
set_glpk_path("C:/Program Files/glpk/bin") # Windows
get_glpk_path()

## End(Not run)
```

```
## Not run:
set_julia_path("C:/Program Files/Julia-1.10.1/bin/")
get_julia_path()

## End(Not run)
## Not run:
set_python_path("C:/Python3")
set_python_path()
get_python_path()

## End(Not run)
```

`set_progress_bar` *Switch on/off and select/customize progress bar*

Description

Switch on/off and select/customize progress bar

Usage

```
set_progress_bar(type = "bw", show = TRUE, clear = FALSE)

show_progress_bar(show = TRUE)
```

Arguments

`type`	character, type of the progress bar to display. Existing options: "bw", "default", "cli", "progress".
`show`	logical, the progress bar is visible if `TRUE`.
`clear`	logical, sets `progressr.clear` global option. If `TRUE`, all outout from the progress bar will be cleared.

Value

sets the progress bar and returns `NULL`

Examples

```
## Not run:
set_progress_bar("bw")
set_progress_bar("default")
set_progress_bar("cli")
set_progress_bar("progress")
set_progress_bar("pbcol")

## End(Not run)
## Not run:
show_progress_bar()
show_progress_bar(FALSE)

## End(Not run)
```

set_scenarios_path *Set or get directory for/with scenarios*

Description

Set or get directory for/with scenarios

Usage

```
set_scenarios_path(path = NULL)

get_scenarios_path()
```

Arguments

path	character, path to the directory with scenarios, default is NULL

Value

sets or gets the path to the directory with scenarios

Examples

```
## Not run:
set_scenarios_path("path/to/scenarios")
get_scenarios_path()

## End(Not run)
```

size *Size of an object*

Description

Size of an object

Usage

```
size(
  x,
  level1 = FALSE,
  units = "auto",
  sort = TRUE,
  decreasing = FALSE,
  byteTol = 0,
  asNumeric = FALSE
)
```

Arguments

x	any R object
level1	logical, if TRUE, the function will return the size of the object and its slots (if any)
units	character, units to display the size, default is "auto"
sort	logical, if TRUE, the function will sort the slots by size
decreasing	logical, if TRUE, the function will sort the slots in decreasing order
byteTol	numeric, threshold in bytes to filter the slots
asNumeric	logical, if TRUE, the function will return the size of the object and its slots in bytes

Value

character value or vector, size of the object or its slots

Examples

```
size(1)
size(rep(1, 1e3))
size(rep(1L, 1e3))
```

---

solve_model *Functions and methods to solve model and scenario objects*

Description

The function interpolates model, writes the script in a directory, runs the external software to solve the model, reads the solution results, and returns a scenario object with the solution.

Usage

```
solve_model(obj, name = NULL, ...)

## S4 method for signature 'model,character'
solve(a, b, ...)

solve_scenario(obj, name = obj@name, ...)

## S4 method for signature 'scenario,character'
solve(a, b, ...)
```

Arguments

obj	model or scenario object
name	character name of scenario to return
...	
solver	a character or list with solver settings
tmp.dir	character path to temporary directory
tmp.del	logical delete temporary directory after the run

Value

When the first argument is a model object, the function

See Also

`read_solution()`

`storage-class` *An S4 class to represent storage type of technological process.*

Description

Storage type of technological processes with accumulating capacity of a commodity.

Details

Storage can be used in combination with other processes, such as technologies, supply, or demand to represent complex technological chains, demand or supply technologies with time-shift. Operation of storage includes accumulation, storing, and release of the stored commodity. The storing cycle operates on the ordered time-slices of the commodity timeframe. The cycle is looped either on an annual basis (last time-slice of a year follows the first time slice of the same year) or within the parent time-frame (for example, when commodity time-frame is "HOUR" and the parent time-frame is "DAY" then the storage cycle will be a calendar day).

Slots

`name` character. Name of the storage (used in sets).

`desc` character. Description of the storage.

`commodity` character. Name of the stored commodity.

`aux` data.frame. Auxiliary commodities.

- **acomm** character. Name of the auxiliary commodity (used in sets).
- **unit** character. Unit of the auxiliary commodity.

`region` character. Region where the storage technology exists or can be installed.

`start` data.frame. Start year when the storage is available for installation.

- **region** character. Regions where the storage is available for investment.
- **start** integer. The first year when the storage is available for investment.

`end` data.frame. Last year when the storage is available for investment.

- **region** character. Region name to apply the parameter, NA for every region.
- **end** integer. The last year when the storage is available for investment.

`olife` data.frame. Operational life of the storage technology, applicable to the new investment only, the operational life (retirement) of preexiting capacity is described in the `stock` slot.

- **region** character. Region name to apply the parameter, NA for every region.
- **olife** integer. Operational life of the storage technology in years.

`capacity` data.frame. Capacity parameters of the storage technology.

- **region** character. Region name to apply the parameter, NA for every region.
- **year** integer. Year to apply the parameter, NA for every year.

cap numeric. Capacity of the storage technology.

cap.lo numeric. Lower bound of the storage capacity.

cap.up numeric. Upper bound of the storage capacity.

cap.fx numeric. Fixed value of the storage capacity. This parameter overrides `cap.lo` and `cap.up`.

ncap.lo numeric. Lower bound of the new storage capacity.

ncap.up numeric. Upper bound of the new storage capacity.

ncap.fx numeric. Fixed value of the new storage capacity. This parameter overrides `ncap.lo` and `ncap.up`.

ret.lo numeric. Lower bound of the storage capacity retirement.

ret.up numeric. Upper bound of the storage capacity retirement.

ret.fx numeric. Fixed value of the storage capacity retirement. This parameter overrides `ret.lo` and `ret.up`.

`charge` data.frame. Pre-charged level at the beginning of the operational cycle.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice for which the charged level will be specified.

charge numeric. Pre-charged or targeted level at the specified slice.

`seff` data.frame. Storage efficiency parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

stgeff numeric. Storage decay annual rate.

inpeff numeric. Input efficiency rate.

outeff numeric. Output efficiency rate.

`af` data.frame. Availability factor parameters.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

af.lo numeric. Lower bound of the availability factor.

af.up numeric. Upper bound of the availability factor.

af.fx numeric. Fixed value of the availability factor. This parameter overrides `af.lo` and `af.up`.

cinp.lo numeric. Lower bound of the input commodity availability factor.

cinp.up numeric. Upper bound of the input commodity availability factor.

cinp.fx numeric. Fixed value of the input commodity availability factor. This parameter overrides `cinp.lo` and `cinp.up`.

cout.lo numeric. Lower bound of the output commodity availability factor.

cout.up numeric. Upper bound of the output commodity availability factor.

cout.fx numeric. Fixed value of the output commodity availability factor. This parameter overrides `cout.lo` and `cout.up`.

`aeff` data.frame. Auxiliary commodities efficiency parameters.

acomm character. Name of the auxiliary commodity (used in sets).

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

stg2ainp numeric. Storaage-level-to-auxiliary-input-commodity coefficient (multiplier).

cinp2ainp numeric. Input-commodity-to-auxiliary-input-commodity coefficient (multiplier).

cout2ainp numeric. Output-commodity-to-auxiliary-input-commodity coefficient (multiplier).

stg2aout numeric. Storage-level-to-auxiliary-output-commodity coefficient (multiplier).

cinp2aout numeric. Input-commodity-to-auxiliary-output-commodity coefficient (multiplier).

cout2aout numeric. Output-commodity-to-auxiliary-output-commodity coefficient (multiplier).

cap2ainp numeric. Capacity-to-auxiliary-input-commodity coefficient (multiplier).

cap2aout numeric. Capacity-to-auxiliary-output-commodity coefficient (multiplier).

ncap2ainp numeric. New-capacity-to-auxiliary-input-commodity coefficient (multiplier).

ncap2aout numeric. New-capacity-to-auxiliary-output-commodity coefficient (multiplier).

ncap2stg numeric. New-capacity-to-storage-level coefficient (multiplier).

`fixom` data.frame. Fixed operation and maintenance cost.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

fixom numeric. Fixed operation and maintenance cost for the specified sets.

`varom` data.frame. Variable operation and maintenance cost.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

inpcost numeric. Costs associated with the input commodity.

outcost numeric. Costs associated with the output commodity.

stgcost numeric. Costs associated with the storage level.

`invcost` data.frame. Investment cost.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

invcost numeric. Overnight investment cost for the specified region and year.

wacc numeric. Weighted average cost of capital. If not supplied, the discount from the model or scenario is used. (currently ignored)

`fullYear` logical. If TRUE (default), the storage technology operates between parent timeframes through the year. The last time-slice in the timeframe is used as a preciding time-slice for the first time-slice in the the same group of time-slices within the parent timeframe. if FALSE, the storage charge and discchare cycle is limited to the parent timeframe. The last time-slice in the timeframe is used as a preciding time-slice for the first time-slice in the the same group of time-slices within the parent timeframe.

`cap2stg` numeric. Charging and discharging capacity to the storing capacity inverse ratio. Can be used to define the storage duration.

`weather` data.frame. Weather factors multipliers.

weather character. Name of the weather factor to apply.

waf.lo numeric. Coefficient that links the weather factor with the lower bound of the availability factor.

waf.up numeric. Coefficient that links the weather factor with the upper bound of the availability factor.

waf.fx numeric. Coefficient that links the weather factor with the fixed value of the availability factor. This parameter overrides `waf.lo` and `waf.up`.

wcinp.lo numeric. Coefficient that links the weather factor with the lower bound of the input commodity availability factor.

wcinp.up numeric. Coefficient that links the weather factor with the upper bound of the input commodity availability factor.

wcinp.fx numeric. Coefficient that links the weather factor with the fixed value of the input commodity availability factor. This parameter overrides `wcinp.lo` and `wcinp.up`.

wcout.lo numeric. Coefficient that links the weather factor with the lower bound of the output commodity availability factor.

wcout.up numeric. Coefficient that links the weather factor with the upper bound of the output commodity availability factor.

wcout.fx numeric. Coefficient that links the weather factor with the fixed value of the output commodity availability factor. This parameter overrides `wcout.lo` and `wcout.up`.

`optimizeRetirement` logical. Incidates if the retirement of the storage should be optimized. Also requires the same parameter in the `model` or `scenario` class to be set to TRUE to be effective.

`misc` list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the storage data, or other metadata.

`subsidy-class` *An S4 class to represent a commodity subsidy*

Description

Subsidies are used to represent the financial support provided to production, consumption, or balance of a commodity.

Slots

`name` character. Name of the subsidy object, used in sets.

`desc` character. Description of the subsidy object.

`comm` character. Name of the subsidized commodity.

`region` character. Region where the subsidy is applied.

`defVal` numeric. Default value of the subsidy.

`sub` data.frame. Subsidy values.

region character. Region name to apply the parameter, NA for every region.

year integer. Year to apply the parameter, NA for every year.

slice character. Time slice to apply the parameter, NA for every slice.

inp numeric. Input subsidy, e.g., per unit of commodity consumed by all processes.

out numeric. Output subsidy, e.g., per unit of commodity produced by all processes.

bal numeric. Balance subsidy, e.g., per unit of commodity balance (production - consumption).

`misc` list. Any additional information or data to store in the subsidy object.

See Also

Other class constraint policy: `class-constraint`, `class-costs`, `newConstraint()`, `newCosts()`, `newSubsidy()`, `newTax()`, `tax-class`

supply-class *An S4 class to represent a supply of a commodity*

Description

An S4 class to represent a supply of a commodity

Value

supply object with given specifications.

Slots

`name` character. Name of the supply object, used in sets.

`desc` character. Description of the supply object.

`commodity` character. The supplied commodity short name.

`unit` character. The main unit of the commodity used in the model.

`weather` data.frame. Weather factors to apply to the supply.

- **weather** character. Name of the weather factor to apply. Must match the weather factor names in a `weather` class in the model.
- **wava.lo** numeric. Coefficient that links the weather factor with the lower bound of the availability factor `ava.lo`.
- **wava.up** numeric. Coefficient that links the weather factor with the upper bound of the availability factor `ava.up`.
- **wava.fx** numeric. Coefficient that links the weather factor with the fixed value of the availability factor `ava.fx`. This parameter overrides `wava.lo` and `wava.up`.

`reserve` data.frame. Total available resource. Applicable to exhaustible resources. Set for each region. If not set, the resource is considered infinite.

- **region** character. Region name to apply the parameter. Use NA to apply to all regions.
- **res.lo** numeric. Lower bound of the total available resource.
- **res.up** numeric. Upper bound of the total available resource.
- **res.fx** numeric. Fixed value of the total available resource. This parameter overrides `res.lo` and `res.up`.

`availability` data.frame. Availability of the resource in physical units.

- **region** character. Region name to apply the parameter. Use NA to apply to all regions.
- **year** integer. Year to apply the parameter. Use NA to apply to all years.
- **slice** character. Time slice to apply the parameter. Use NA to apply to all slices.
- **ava.lo** numeric. Lower bound of the availability factor.
- **ava.up** numeric. Upper bound of the availability factor.
- **ava.fx** numeric. Fixed value of the availability factor. This parameter overrides `ava.lo` and `ava.up`.
- **cost** numeric. Cost of the resource extraction, if not set, the resource is considered free.

`region` character. Regions where the supply process exists. Must include all regions used in other slots. `availability` and `reserve` slots also limit possible regions.

`misc` list. List of additional parameters that are not used in the model but can be used for reference or user-defined functions. For example, links to the source of the supply data, or other metadata.

tax-class *An S4 class to represent a commodity tax*

Description

Taxes are used to represent the financial levy imposed on production, consumption, or balance of a commodity.

Slots

`name` character. Name of the tax object, used in sets.

`desc` character. Description of the tax object.

`comm` character. Name of the taxed commodity.

`region` character. Region where the tax is applied.

`defVal` numeric. Default value of the tax for not specified sets, 0 if not specified.

`tax` data.frame. Tax values.

- **region** character. Region name to apply the parameter, NA for every region.
- **year** integer. Year to apply the parameter, NA for every year.
- **slice** character. Time slice to apply the parameter, NA for every slice.
- **inp** numeric. Input tax, e.g., per unit of commodity consumed by all processes.
- **out** numeric. Output tax, e.g., per unit of commodity produced by all processes.
- **bal** numeric. Balance tax, e.g., per unit of commodity balance (production - consumption).

`misc` list. Additional information.

See Also

Other class constraint policy: `class-constraint`, `class-costs`, `newConstraint()`, `newCosts()`, `newSubsidy()`, `newTax()`, `subsidy-class`

tsl2year *Mapping function between time-slices and day of the year*

Description

Mapping function between time-slices and day of the year

Mapping function between time-slices and hour

Mapping function between time-slices and month

Usage

```
tsl2year(tsl, return.null = TRUE)

tsl2yday(tsl, return.null = TRUE)

tsl2hour(tsl, return.null = TRUE)

tsl2month(tsl, format = tsl_guess_format(tsl), return.null = TRUE)
```

Arguments

tsl	character vector with time slices
return.null	logical, valid for the cased then all values are NA, then NULL will be returned if return.null = TRUE,
format	character, the time slices format

Value

Integer vector of years, the same length as the input vector

Integer vector of days of the year, the same length as the input vector

Integer vector of hours, the same length as the input vector

Integer vector of months, the same length as the input vector

Functions

- tsl2year(): Extract year from time-slices
- tsl2yday(): Extract the day of the year from time-slices
- tsl2hour(): Extract hour from time-slices
- tsl2month(): Extract month from time-slices

Examples

```
tsl <- c("y2007_d365_h15", NA, "d151_h22", "d001", "m10_h12")
tsl2year(tsl)
tsl
tsl2yday(tsl)
tsl
tsl2hour(tsl)
tsl2month(c("d001_h00", "d151_h22", "d365_h23"))
tsl2month(c("m01_h12", "m05_h02", "m10_h01"))
```

---

tsl_formats *Common formats of time-slices.*

Description

This set of functions converts date-time objects to model's time-slices in a given format, and vice versa, maps time-slices to date-time, and extracts year, month, day of the year, hour.

Usage

```
tsl_formats

tsl_sets

dtm2tsl(dtm, format = "d365_h24", d366.as.na = grepl("d365", format))

tsl2dtm(
  tsl,
```

```
  format = tsl_guess_format(tsl),
  tmz = "UTC",
  year = NULL,
  mday = NULL
)
```

Arguments

dtm	vector of timepoints in Date format
format	character, format of the slices
d366.as.na	logical, if
tsl	character vector with time-slices
tmz	time-zone
year	year, used when time-slices don't store year
mday	day of month, for time slices without the information

Format

A character vector with formats:

d365 daily time-slices, 365 a year (leap year's 366th day is disregarded)

d365_h24 time slices with year-day numbers and hours, 8760 in total

... etc.

An object of class `list` of length 1.

Value

Character vector with time-slices names

Vector in Date-Time format

Examples

```
dtm2tsl(lubridate::now())
dtm2tsl(lubridate::ymd("2020-12-31"))
dtm2tsl(lubridate::ymd("2020-12-31"), d366.as.na = FALSE)
dtm2tsl(lubridate::now(tzone = "UTC"), format = "d365")
dtm2tsl(lubridate::ymd("2020-12-31"), format = "d365")
dtm2tsl(lubridate::ymd("2020-12-31"), format = "d365", d366.as.na = FALSE)
dtm2tsl(lubridate::ymd("2020-12-31"), format = "d366")
tsl <- c("y2007_d365_h15", NA, "d151_h22", "d001", "m10_h12")
tsl2dtm(tsl[1])
tsl2dtm(tsl[1:2])
tsl2dtm(tsl[2])
tsl2dtm(tsl[3])
tsl2dtm(tsl[4])
tsl2dtm(tsl[3], year = 2010)
tsl2dtm(tsl[4], year = 1900)
tsl2dtm(tsl[3:4], year = 1900)
```

tsl_guess_format *Guess format of time-slices*

Description

Guess format of time-slices

Usage

```
tsl_guess_format(tsl)
```

Arguments

```
tsl
```

Value

Character vector with the guessed format of the time-slices

Examples

```
tsl <- c("y2007_d365_h15", NA, "d151_h22", "d001", "m10_h12")
tsl_guess_format(tsl)
tsl_guess_format(tsl[1])
tsl_guess_format(tsl[2])
tsl_guess_format(tsl[3])
tsl_guess_format(tsl[4])
tsl_guess_format(tsl[5])
```

update *Update trade object*

Description

Update trade object

Usage

```
## S4 method for signature 'storage'
update(object, ...)

## S4 method for signature 'trade'
update(object, ...)

## S4 method for signature 'import'
update(object, ...)
```

Arguments

object	an S4 class object to be updated.
...	slot-names with data to update the S4 object

update,export-method *Update export object*

Description

The method replaces slots of the export object with new values.

Usage

```
## S4 method for signature 'export'
update(object, ...)

## S4 method for signature 'weather'
update(object, ...)
```

Arguments

object	object of class export
...	slot-names with data to update (see newWeather)

update,supply-method *Update supply object*

Description

Update supply object

Usage

```
## S4 method for signature 'supply'
update(object, ...)
```

write_script *Write scenario object as a Python, Julia, GAMS, or MathProg script with data files to a directory*

Description

Write scenario object as a Python, Julia, GAMS, or MathProg script with data files to a directory

Usage

```
write_script(scen, tmp.dir = NULL, solver = NULL, ...)

write_sc(x, tmp.dir = NULL, solver = NULL, ...)

write.sc(x, tmp.dir = NULL, solver = NULL, ...)
```

Arguments

scen	scenario object, must be interpolated
tmp.dir	character, path
solver	list of character with solver specification.
...	additional solver parameters

See Also

solve() to run the script, solve the scenario. read_solution to read model solution.

---

yday2YDAY *Convert year-days to YDAY set 'dNNN'*

Description

Convert year-days to YDAY set 'dNNN'

Usage

```
yday2YDAY(x, width = 3, prefix = "d", flag = "0")
```

Arguments

x	integer vector, year-days (for example, 1-365 for annual data)
width	integer, width of the output string, default is 3
prefix	character, prefix to add to the name, default is 'd'
flag	character, flag to add to the name, default is '0'

Value

character vector of the same length as x with formatted year-days to be used in the YDAY set.

Examples

```
yday2YDAY(1:365)
```

Index