Package: ospsuite (via r-universe)

July 15, 2026

Type Package

Title R package to manipulate OSPSuite Models

Version 12.4.3.9015

Description The ospsuite-R package provides the functionality of loading, manipulating, and simulating the simulations created in the Open Systems Pharmacology Software tools PK-Sim and MoBi.

License GPL-2 | file LICENSE

URL https://github.com/open-systems-pharmacology/ospsuite-r, https://www.open-systems-pharmacology.org/OSPSuite-R/ (release), https://www.open-systems-pharmacology.org/OSPSuite-R/dev (development)

BugReports https://github.com/open-systems-pharmacology/ospsuite-r/issues

Depends R (>= 4.4)

Imports dplyr (>= 1.0.0), ospsuite.utils (>= 1.10.0), purrr, readr, stringr, tidyr, ggplot2, rlang, tlf (>= 1.6.0), ospsuite.plots (>= 1.2.0.9003), data.table, xml2, openxlsx, lifecycle, cli, rSharp, checkmate, R6

Suggests knitr, rmarkdown, testthat (>= 3.0.3), vdiffr (>= 1.0.0), withr, showtext

Language en-US

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Collate 'aging-data.R' 'api-config.R' 'application.R' 'cache.R'
'container.R' 'data-column.R' 'data-combined.R'
'data-importer-configuration.R' 'data-repository.R'
'data-set.R' 'data.R' 'default-plot-configuration.R'
'dot-net-wrapper.R' 'entity.R' 'enums.R' 'formula.R'
'get-net-task.R' 'globals.R' 'individual-characteristics.R'
'init-package.R' 'interval.R' 'messages.R'
'molecule-ontogeny.R' 'molecule.R' 'object-base.R'
'utilities.R' 'ospsuite-env.R' 'output-schema.R'
'output-selections.R' 'parameter-range.R' 'parameter.R'
'pk-parameter-sensitivity.R' 'pk-parameter.R'
'plot-individual-time-profile.R' 'plot-observed-vs-simulated.R'
'plot-population-time-profile.R'
'plot-residuals-vs-simulated.R' 'plot-residuals-vs-time.R'
'plot-with-ospsuite-plots.R' 'population-characteristics.R'
'population.R' 'quantity-pk-parameter.R' 'quantity-selection.R'
'quantity.R' 're-exports.R' 'sensitivity-analysis-results.R'
'sensitivity-analysis-run-options.R' 'sensitivity-analysis.R'
'simulation-batch-options.R' 'simulation-batch-run-values.R'
'simulation-batch.R' 'simulation-pk-analyses.R'
'simulation-results.R' 'simulation-run-options.R'
'simulation-settings.R' 'simulation.R' 'snapshot-parameter.R'
'solver-settings.R' 'utilities-path.R' 'standard-path.R'
'table-formula.R' 'tasks-env.R' 'user-defined-pk-parameter.R'
'utilities-container.R' 'utilities-data-combined.R'
'utilities-data-importer-configuration.R'
'utilities-data-repository.R' 'utilities-data-set.R'
'utilities-dot-net.R' 'utilities-entity.R' 'utilities-file.R'
'utilities-individual.R' 'utilities-molecule.R'
'utilities-output-schema.R' 'utilities-output-selections.R'
'utilities-parameter-value.R' 'utilities-parameter.R'
'utilities-pk-analysis.R' 'utilities-pk-parameter.R'
'utilities-pksim.R' 'utilities-plotting.R'
'utilities-population.R' 'utilities-quantity.R'
'utilities-sensitivity-analysis.R'
'utilities-simulation-results.R' 'utilities-simulation.R'
'utilities-snapshots.R' 'utilities-units.R' 'value-point.R'
'zzz.R' 'zzz_deprecated.R'

Remotes rSharp=Open-Systems-Pharmacology/rSharp,
ospsuite.utils=Open-Systems-Pharmacology/OSPSuite.RUtils,
tlf=Open-Systems-Pharmacology/TLF-Library,
ospsuite.plots=Open-Systems-Pharmacology/OSPSuite.Plots

Config/roxygen2/version 8.0.0

Repository https://open-systems-pharmacology.r-universe.dev

Date/Publication 2026-07-15 12:08:49 UTC

RemoteUrl https://github.com/Open-Systems-Pharmacology/OSPSuite-R

RemoteRef graceful-load-no-runtime

RemoteSha f5f251fd682f0e221d048cc9c50665ca55e0db85

Contents

`.gatherFiles` *Gather files and files from folders to one location*

Description

Gather files and files from folders to one location

Usage

```
.gatherFiles(...)
```

Arguments

`...`	character strings of file paths or folder paths

Value

A temporary directory with all files copied to it

`.getConcurrentSimulationRunnerResults` *Get SimulationResults from ConcurrentSimulationRunner*

Description

Get SimulationResults from ConcurrentSimulationRunner

Usage

```
.getConcurrentSimulationRunnerResults(
  results,
  resultsIdSimulationIdMap,
  simulationIdSimulationMap,
  silentMode,
  stopIfFails
)
```

Arguments

results	.NET object created by RunConcurrently()
resultsIdSimulationIdMap	Map of results ids as keys with values being the ids of simulations the respective batch was created with. The order of IDs is as they were added to the batch.
simulationIdSimulationMap	A named list of simulation ids as keys and simulation objects as values to the id of a result
silentMode	If TRUE, no warnings are displayed if a simulation fails. Default is FALSE. Has no effect if stopIfFails is TRUE.
stopIfFails	Whether to stop the execution if one of the simulations failed. Default is FALSE.

Details

Create a list of SimulationResults-objects from the results of a ConcurrentSimulationRunner

Value

A named list of SimulationResults objects with the names being the ids of simulations or simulation-batch values pairs they were produced by

---

.getOspDimensions *get OSP Dimensions from OSPSuite.Dimensions.xml data*

Description

get OSP Dimensions from OSPSuite.Dimensions.xml data

Usage

```
.getOspDimensions(xmlData)
```

Arguments

xmlData	XML data from .parseDimensionsXML()

Value

a list of supported dimensions

.getOspUnits *get OSP Units from OSPSuite.Dimensions.xml data*

Description

get OSP Units from OSPSuite.Dimensions.xml data

Usage

```
.getOspUnits(xmlData)
```

Arguments

xmlData	XML data from .parseDimensionsXML()

Value

a list of supported units

.getQuantityDisplayPaths *Retrieves the display path of the quantity defined by path in the simulation*

Description

Retrieves the display path of the quantity defined by path in the simulation

Usage

```
.getQuantityDisplayPaths(paths, simulation)
```

Arguments

paths	A single string or array of paths path relative to the Simulation
simulation	A Simulation used to find the entities

Value

a display path for each entry in paths

`.parameterValueListFrom`

Converts a list of .NET `ParameterValue` *into a list with 2 entries:* `paths`*,* `values`*. A 3rd optional entry* `units` *will be defined if the parameter* `addUnits` *is set to* `TRUE`*. Note: Units are only available for .NET object of type* `ParameterValueWithUnit`

Description

Converts a list of .NET `ParameterValue` into a list with 2 entries: `paths`, `values`. A 3rd optional entry `units` will be defined if the parameter `addUnits` is set to `TRUE`. Note: Units are only available for .NET object of type `ParameterValueWithUnit`

Usage

```
.parameterValueListFrom(netParameterValues, addUnits = FALSE)
```

Arguments

`netParameterValues`	List of.NET `ParameterValue` or `ParameterValueWithUnit`
`addUnits`	If `TRUE`, a a third list will be returned containing the units in which the parameters are defined. Default is `FALSE`

Value

A list with 3 sublist: `paths`, `values`, and optionally `units` containing the corresponding values from each parameter value

---

`.parseDimensionsXML` *parse OSPSuite.Dimensions.xml containing dimensions and units*

Description

parse OSPSuite.Dimensions.xml containing dimensions and units

Usage

```
.parseDimensionsXML()
```

Value

An XML document

.savePKAnalysesToCSV *Saves the pK-analyses to csv file*

Description

Saves the pK-analyses to csv file

Usage

```
.savePKAnalysesToCSV(pkAnalyses, filePath)
```

Arguments

pkAnalyses	pK-Analyses to export (typically calculated using calculatePKAnalyses or imported from file)
filePath	Full path where the pK-Analyses will be saved.

.savePopulationToCSV *Saves the population to csv file*

Description

Saves the population to csv file

Usage

```
.savePopulationToCSV(population, filePath)
```

Arguments

population	Population to export to csv (typically imported from file using loadPopulation)
filePath	Full path where the population will be saved.

Examples

```
csvPath <- system.file("extdata", "pop.csv", package = "ospsuite")

# Load the population
population <- loadPopulation(csvPath)

# Exports the population
exportPopulationToCSV(population, tempfile())
```

.saveResultsToCSV *Saves the simulation results to csv file*

Description

Saves the simulation results to csv file

Usage

```
.saveResultsToCSV(results, filePath)
```

Arguments

results	Results to export (typically calculated using runSimulations or imported from file).
filePath	Full path where the results will be saved.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath)

# Add some outputs to the simulation
addOutputs("Organism|**|*", sim)

# Run the simulation
results <- runSimulations(sim)[[1]]

# Export the results to csv file
exportResultsToCSV(results, tempfile())
```

.saveSensitivityAnalysisResultsToCSV *Saves the simulation analysis results to csv file*

Description

Saves the simulation analysis results to csv file

Usage

```
.saveSensitivityAnalysisResultsToCSV(results, filePath)
```

Arguments

results	Results to export (typically calculated using runSensitivityAnalysis or imported from file)
filePath	Full path where the results will be saved.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath)

# Create a new sensitivity object for the simulation
sensitivity <- SensitivityAnalysis$new(sim)

# Runs the sensitivity analysis
results <- runSensitivityAnalysis(sensitivity)

# Export the results to csv file
exportSensitivityAnalysisResultsToCSV(results, tempfile())
```

.scaleQuantityValues *Scale current values of quantities using a factor*

Description

Scale current values of quantities using a factor

Usage

```
.scaleQuantityValues(quantities, factor)
```

Arguments

quantities	A single or a list of Quantity
factor	A numeric value that will be used to scale all quantities

.setEndSimulationTime *Set end time of the simulation*

Description

Either extends or shortens the simulation time to the specified end time. Time points that are later than the specified end time are removed. Intervals that start after the specified end time are removed. Intervals that start before the specified end time are shortened to the specified end time.

Usage

```
.setEndSimulationTime(simulation, endTime)
```

Arguments

simulation	Simulation for which the end time should be set
endTime	End time of the simulation in min

.setQuantityValues *Set values of quantity*

Description

Set values of quantity

Usage

```
.setQuantityValues(quantities, values, units = NULL)
```

Arguments

quantities	A single or a list of Quantity
values	A numeric value that should be assigned to the quantity or a vector of numeric values, if the value of more than one quantity should be changed. Must have the same length as 'quantities'. Alternatively, the value can be a unique number. In that case, the same value will be set in all parameters
units	A string or a list of strings defining the units of the values. If NULL (default), values are assumed to be in base units. If not NULL, must have the same length as quantities.

.validateHasUnit *Check if quantity can be represented in the unit*

Description

Check if quantity can be represented in the unit

Usage

```
.validateHasUnit(quantity, unit)
```

Arguments

quantity	Quantity object
unit	Unit name to check for

Value

If validations are successful, NULL is returned. Otherwise, error is signaled.

addOutputInterval *Adds an interval to the output schema of the simulation*

Description

Adds an interval to the output schema of the simulation

Usage

```
addOutputInterval(
  simulation,
  startTime,
  endTime,
  resolution,
  intervalName = NULL
)
```

Arguments

simulation	Simulation for which a new interval should be created
startTime	Start time of the interval in min
endTime	End time of the interval in min
resolution	resolution in points/min
intervalName	Optional Name of interval. If not specified, a unique name will be assigned.

Value

Returns the interval created.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath, addToCache = FALSE, loadFromCache = FALSE)

# clears the previous output schema
clearOutputIntervals(sim)

# Adds a new interval starting at 1h and ending at 10h with a resolution of 10 points per hour
addOutputInterval(sim, 1 * 60, 10 * 60, 1 / 6)

# Adds another interval starting at 10h and ending at 17h with a resolution of 4 points per hour
# and a specified name
addOutputInterval(sim, 10 * 60, 17 * 60, 4 / 60, intervalName = "Second Interval")
```

---

addOutputs *Adds the quantities as output into the* simulation*. The quantities can either be specified using explicit instances or using paths.*

Description

Adds the quantities as output into the simulation. The quantities can either be specified using explicit instances or using paths.

Usage

```
addOutputs(quantitiesOrPaths, simulation, stopIfNotFound = TRUE)
```

Arguments

quantitiesOrPaths	Quantity instances (element or vector) (typically retrieved using getAllQuantitiesMatching) or quantity path (element or vector) to add.
simulation	Instance of a simulation for which output selection should be updated.
stopIfNotFound	Boolean. If TRUE (default) and no quantity exists for the given path, an error is thrown. If FALSE, NULL is returned.

Examples

```
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

paths <- c("Organism|VenousBlood|Plasma|Aciclovir", "Organism|ArterialBlood|**|Aciclovir")
addOutputs(paths, sim)

parameter <- getParameter("Organism|Liver|Volume", sim)
addOutputs(parameter, sim)
```

---

addResidualColumn *Add a residual column to paired observed/predicted data*

Description

Calculates residuals between observed and predicted values and adds them as a new column to the provided data frame or data table. The calculation supports three scaling methods: `"log"`, `"linear"` (alias `"lin"`), and `"ratio"`.

Usage

```
addResidualColumn(
  pairedData,
  observed = "yValuesObserved",
  predicted = "yValuesSimulated",
  residuals = "residualValues",
  scaling = "log"
)
```

Arguments

`pairedData`	A `data.frame` or `data.table` containing paired observed and predicted values. The columns specified by `observed` and `predicted` must exist.
`observed`	A string specifying the column name for observed values. Default is `"yValuesObserved"`.
`predicted`	A string specifying the column name for predicted (simulated) values. Default is `"yValuesSimulated"`.
`residuals`	A string specifying the name for the new residuals column to be added. Default is `"residualValues"`.
`scaling`	A character string specifying the scaling method. One of `"log"` (default), `"linear"` (or `"lin"` / `"identity"`), or `"ratio"`.

Details

Scaling Methods:

- `"linear"` / `"lin"`: Absolute residuals as $residual = predicted - observed$.

- `"log"`: Log-scale residuals as $residual = \log(predicted) - \log(observed)$. Data points where the observed or predicted value is zero or negative produce undefined logarithms; these are set to `NaN` and a warning is issued reporting the number of such points.
- `"ratio"`: Ratio of observed to predicted as $residual = observed/predicted$. A warning is issued when predicted values are zero or negative.

Value

The input `pairedData` with an additional column named according to the `residuals` argument. The column carries a `"label"` attribute describing how residuals were computed (e.g. `"residuals\nlog(predicted) - log(observed)"`), which can be used as a y-axis label in plots.

See Also

Other data-combined: `DataCombined`, `calculateResiduals()`, `convertUnits()`

Examples

```
## Not run:
paired <- data.frame(
  yValuesObserved  = c(1, 2, 4),
  yValuesSimulated = c(1.1, 1.9, 3.8)
)
addResidualColumn(paired, scaling = "log")
addResidualColumn(paired, scaling = "linear")
addResidualColumn(paired, scaling = "ratio")

## End(Not run)
```

---

addUserDefinedPKParameter

Adds and returns a User-Defined PK-Parameter to the managed list of PK-Parameters

---

Description

Adds and returns a User-Defined PK-Parameter to the managed list of PK-Parameters

Usage

```
addUserDefinedPKParameter(
  name,
  standardPKParameter,
  displayName = NULL,
  displayUnit = NULL
)
```

Arguments

`name`	Name of the user defined PK-Parameter
`standardPKParameter`	Defined the standard PK-Parameter to use to perform the calculations
`displayName`	Display Name to use when exporting the values (optional, default value is name)
`displayUnit`	Unit in which the value will be exported

Value

The newly created `UserDefinedPKParameter`that was added to the list of PK-Parameters

Examples

```
# Adds a user defined parameter named MyAuc that will calculate the value of AUC
# between t=50 min and t=80min
myAUC <- addUserDefinedPKParameter(
  name = "MyAUC",
  standardPKParameter = StandardPKParameter$AUC_tEnd
)
myAUC$startTime <- 50
myAUC$endTime <- 80

# Adds a user defined parameter named MyCMax that will calculate the value of Cmax
# between the 4th and 5th application
myCMax <- addUserDefinedPKParameter(
  name = "MyCMax",
  standardPKParameter = StandardPKParameter$C_max
)
myCMax$startApplicationIndex <- 4
myCMax$endApplicationIndex <- 5
```

`AgingData` *AgingData*

Description

Results of a sensitivity analysis run (either individual or population simulation)

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `AgingData`

Active bindings

`individualIds` Array of individual ids (corresponding to the individualId column of the aging table)

`parameterPaths` Array of parameter paths (corresponding to the ParameterPath column of the aging table)

`times` Array of time values (corresponding to the Time column of the aging table)

`values` Array of parameter values (corresponding to the Value column of the aging table)

Methods

Public methods:

- `AgingData$new()`
- `AgingData$print()`

`AgingData$new()`: Initialize a new instance of the class

Usage:

```
AgingData$new()
```

Returns: A new `OSPSuite.R.Domain.AgingData` object.

`AgingData$print()`: Print the object to the console

Usage:

```
AgingData$print(...)
```

Arguments:

`...` Rest arguments.

`allAvailableDimensions` *List all available dimensions in the* `OSPSuite` *platform*

Description

List all available dimensions in the `OSPSuite` platform

Usage

```
allAvailableDimensions()
```

Value

Returns the names of all available dimensions defined in the `OSPSuite` platform.

Examples

```
allAvailableDimensions()
```

allPKParameterNames *Returns the name of all pk parameters defined in the system*

Description

Returns the name of all pk parameters defined in the system

Usage

```
allPKParameterNames()
```

Examples

```
pkParameterNames <- allPKParameterNames()
```

Application *Application*

Description

Application representing one instance of an application in a Simulation

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `Application`

Active bindings

`startTime` Start time of application instance (Read-Only)

`infusionTime` Infusion time of application instance or null if undefined (Read-Only)

`drugMass` Applied drugmass of the application instance of null if undefined (Read-Only)

Methods

Public methods:

- `Application$print()`

`Application$print()`: Print the object to the console

Usage:

```
Application$print(...)
```

Arguments:

`...` Rest arguments.

calculatePKAnalyses *Calculates the pkAnalyses for all output values available in* results.

Description

Calculates the pkAnalyses for all output values available in results.

Usage

```
calculatePKAnalyses(results)
```

Arguments

results	Results of simulation. Typically the results are calculated using runSimulations or imported from csv file via importResults.

Value

An instance of SimulationPKAnalyses class.

Examples

```
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

addOutputs("Organism|VenousBlood|*|Aciclovir", sim)
results <- runSimulations(sim)[[1]]
pkAnalyses <- calculatePKAnalyses(results)
```

calculateResiduals *Calculate residuals for datasets in* DataCombined

Description

Computes residuals between observed and simulated datasets by interpolating simulated values to observed time points and calculating the difference according to the specified scaling method.

Usage

```
calculateResiduals(dataCombined, scaling, xUnit = NULL, yUnit = NULL)
```

Arguments

`dataCombined`	A single instance of `DataCombined` class containing both observed and simulated datasets to be compared.
`scaling`	A character specifying the scaling method for residual calculation. Accepted values are `"lin"` / `"linear"` for linear residuals (simulated - observed), `"log"` for logarithmic residuals (log(simulated) - log(observed)), or `"ratio"` for the ratio of observed to simulated (observed / simulated).
`xUnit`, `yUnit`	Target units for `xValues` and `yValues`, respectively. If not specified (`NULL`), the first existing unit in the respective columns will be selected as the common unit. For available dimensions and units, see `ospsuite::ospDimensions` and `ospsuite::ospUnits`.

Details

Algorithm Overview:

The function performs the following steps to calculate residuals:

1. **Data Validation and Pairing**: For each group in the data, the function pairs observed datasets with simulated datasets. Any unpaired datasets (observed without corresponding simulated or vice versa) are removed.
2. **Unit Harmonization**: All datasets are converted to common units (specified by `xUnit` and `yUnit` parameters) to ensure consistent calculations.
3. **Interpolation**: For each observed-simulated pair, the function uses linear interpolation to estimate simulated values at the exact time points where observations exist:
   - With 2+ simulated points: Linear interpolation via `stats::approx()`
   - With 1 simulated point: Direct matching for identical x-values only
   - With 0 simulated points: All residuals set to NA
4. **Residual Calculation**: Residuals are computed based on the scaling method:
   - **Linear scaling** (`"lin"` / `"linear"`):

$$residual = y_{simulated} - y_{observed}$$

   - **Logarithmic scaling** (`"log"`):

$$residual = \log(y_{simulated}) - \log(y_{observed})$$

     Data points where the observed or predicted value is zero or negative produce undefined logarithms. These residuals are set to `NaN` and a warning is emitted reporting the number of such points. The affected rows are excluded from the returned data frame.
   - **Ratio scaling** (`"ratio"`):

$$residual = y_{observed}/y_{simulated}$$

Important Notes:

- Residuals can only be computed when both observed and simulated data exist for the same group
- Interpolation does not extrapolate beyond the range of simulated data (returns NA for observed points outside simulated time range)
- NA and NaN residual values are automatically filtered from the output
- When multiple observed/simulated datasets exist in a group, all possible pairs are evaluated

Value

A tibble (data frame) containing paired observed-simulated data with calculated residuals. The following columns will be present:

group Grouping identifier from the original DataCombined object

name Name of the observed dataset

nameSimulated Name of the paired simulated dataset

xValues X-axis values (typically time points) from observed data

xUnit Unit of x-values after harmonization

xDimension Dimension of x-values (e.g., "Time")

yValuesObserved Observed y-values at each x-point

yValuesSimulated Simulated y-values interpolated to observed x-points

residualValues Calculated residuals (method depends on scaling parameter)

yUnit Unit of y-values after harmonization

yDimension Dimension of y-values (e.g., "Concentration")

yErrorValues Error values from observed data (if available)

yErrorType Type of error (e.g., "SD", "SE")

yErrorUnit Unit of error values

lloq Lower limit of quantification (if available)

Returns `NULL` with a warning if no pairable datasets are found.

See Also

Other data-combined: `DataCombined`, `addResidualColumn()`, `convertUnits()`

Examples

```
calculateResiduals(dataCombinedAciclovir, scaling = "linear")
```

`clearMemory` *Clears the memory used by all underlying objects*

Description

Clears the memory used by all underlying objects

Usage

```
clearMemory(clearSimulationsCache = FALSE)
```

Arguments

clearSimulationsCache
: optional - Should the simulation cache also be cleared? Default is FALSE.

Details

The function aims at clearing the memory used by object references allocated during some workflows. The memory should typically be freed automatically when the system is under memory pressure or when the garbage collection is kicking in. However, it may be necessary sometimes to explicitly start the garbage collection process.

Examples

```
# This will clear the memory and also clear the simulations cache but leave
# the environment intact.
clearMemory(clearSimulationsCache = TRUE)
```

---

clearOutputIntervals *Removes all intervals as well as all single time points from the output schema defined in* simulation

---

Description

Removes all intervals as well as all single time points from the output schema defined in simulation

Usage

```
clearOutputIntervals(simulation)
```

Arguments

simulation
: Instance of a simulation for which output intervals should be cleared

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Make sure we create a new simulation so that we do not impact other examples
sim <- loadSimulation(simPath, addToCache = FALSE, loadFromCache = FALSE)

clearOutputIntervals(sim)
```

clearOutputs *Removes all selected output from the given* simulation

Description

Removes all selected output from the given simulation

Usage

```
clearOutputs(simulation)
```

Arguments

simulation	Instance of a simulation for which output selection should be cleared.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath, )

clearOutputs(sim)
```

CompareBy *How should comparison of entities be performed*

Description

How should comparison of entities be performed

Usage

```
CompareBy
```

convertSnapshot *Convert between snapshot and project formats*

Description

[Deprecated]

`convertSnapshot()` is deprecated and will be removed in a future release. Use `loadProjectFromSnapshot()` to convert a snapshot to a project, and `exportProjectToSnapshot()` to convert a project to a snapshot.

Usage

```
convertSnapshot(..., format, output = ".", runSimulations = FALSE)
```

Arguments

`...`	character strings, path to files or a directory containing files to convert
`format`	character string, target format either "snapshot" or "project".
`output`	character string, path to the output directory where to write the converted files
`runSimulations`	logical, whether to run simulations during conversion (default = FALSE). Only when converting from snapshot to project.

convertUnits *Convert datasets in* `DataCombined` *to common units*

Description

When multiple (observed and/or simulated) datasets are present in a data frame, they are likely to have different units. This function helps to convert them to a common unit specified by the user.

This is especially helpful while plotting since the quantities from different datasets to be plotted on the X-and Y-axis need to have same units to be meaningfully compared.

Usage

```
convertUnits(dataCombined, xUnit = NULL, yUnit = NULL)
```

Arguments

`dataCombined`	A single instance of `DataCombined` class.
`xUnit, yUnit`	Target units for `xValues` and `yValues`, respectively. If not specified (`NULL`), first of the existing units in the respective columns (`xUnit` and `yUnit`) will be selected as the common unit. For available dimensions and units, see `ospsuite::ospDimensions` and `ospsuite::ospUnits`, respectively.

Value

A data frame with measurement columns transformed to have common units.

In the returned tibble data frame, the following columns will always be present:

name - group - dataType - xValues - xDimension - xUnit - yValues - yErrorValues - yDimension - yUnit - yErrorType - yErrorUnit - molWeight

Importantly, the `xUnit` and `yUnit` columns will have unique entries.

Note

Molecular weight is **required** for the conversion between certain dimensions (`Amount`, `Mass`, `Concentration (molar)`, and `Concentration (mass)`). Therefore, if molecular weight is missing for these dimension, the unit conversion will fail.

See Also

Other data-combined: `DataCombined`, `addResidualColumn()`, `calculateResiduals()`

Examples

```
# simulated data
simFilePath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim <- loadSimulation(simFilePath)
simResults <- runSimulations(sim)[[1]]
outputPath <- "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)"

# observed data
obsData <- lapply(
  c("ObsDataAciclovir_1.pkml", "ObsDataAciclovir_2.pkml", "ObsDataAciclovir_3.pkml"),
  function(x) loadDataSetFromPKML(system.file("extdata", x, package = "ospsuite"))
)
names(obsData) <- lapply(obsData, function(x) x$name)


# Create a new instance of `DataCombined` class
myDataCombined <- DataCombined$new()

# Add simulated results
myDataCombined$addSimulationResults(
  simulationResults = simResults,
  quantitiesOrPaths = outputPath,
  groups = "Aciclovir PVB"
)

# Add observed data set
myDataCombined$addDataSets(obsData$`Vergin 1995.Iv`, groups = "Aciclovir PVB")

convertUnits(
  myDataCombined,
  xUnit = ospUnits$Time$s,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`
```

)

---

createDistributions *Creates the parameter distributions based on the given individual* individualCharacteristics

---

Description

Creates the parameter distributions based on the given individual individualCharacteristics

Usage

```
createDistributions(individualCharacteristics)
```

Arguments

individualCharacteristics
: Characteristics of the individual to create as an instance of OriginData

Value

An array of ParameterValue containing the value of each individual parameter

---

createImporterConfigurationForFile
Create a DataImporterConfiguration *for an XLS sheet*

Description

Create a DataImporterConfiguration for an XLS sheet

Usage

```
createImporterConfigurationForFile(filePath, sheet = NULL)
```

Arguments

filePath	Path to XLS file
sheet	optional - name of the sheet. If no sheet is specified, the first sheet of the XLS file is used.

Details

The function tries to parse the structure of the excel sheet and creates a default configuration for this sheet. It is advised to check the configuration and adjust if necessary before using with loadDataSetsFromExcel().

Value

DataImporterConfiguration object for XLS file to be used in loadDataSetsFromExcel().

Examples

```
xlsFilePath <- system.file("extdata", "CompiledDataSet.xlsx", package = "ospsuite")
# When sheet is specified, it is automatically added to the configuration
importerConfiguration <- createImporterConfigurationForFile(
  xlsFilePath,
  sheet = "TestSheet_1"
)

dataSets <- loadDataSetsFromExcel(
  xlsFilePath = xlsFilePath,
  importerConfigurationOrPath = importerConfiguration,
  importAllSheets = FALSE
)
```

---

createIndividual *Creates a set of parameter values describing an individual using the PK-Sim Database*

Description

Creates a set of parameter values describing an individual using the PK-Sim Database

Usage

```
createIndividual(individualCharacteristics)
```

Arguments

individualCharacteristics
: Characteristics of the individual to create as an instance of IndividualCharacteristics

Value

A list with three entries:

- distributedParameters containing the actual parameter values modified by the create individual algorithm.
- derivedParameters containing the parameter values modified indirectly by the algorithm. Those parameters are typically formula parameters.
- seed containing the seed value used to generate random values

Note

When updating a simulation with the value for a new individual, only use the distributedParameters to ensure that you do not override formula parameters.

createIndividualCharacteristics *Create the characteristics of an individual using the PK-Sim Database.*

Description

Create the characteristics of an individual using the PK-Sim Database.

Usage

```
createIndividualCharacteristics(
  species,
  population = NULL,
  gender = NULL,
  weight = NULL,
  weightUnit = "kg",
  height = NULL,
  heightUnit = "cm",
  age = NULL,
  ageUnit = "year(s)",
  gestationalAge = 40,
  gestationalAgeUnit = "week(s)",
  moleculeOntogenies = NULL,
  seed = NULL
)
```

Arguments

species	Species of the individual as defined in PK-Sim (see Species enum)
population	Population to use to create the individual. This is required only when the species is Human. (See HumanPopulation enum)
gender	Gender to use to create the individual. (See Gender enum)
weight	Weight of the created individual
weightUnit	Unit in which the weight value is defined. Default is kg
height	Height of the created individual (for human species only)
heightUnit	Unit in which the height value is defined. Default is cm
age	Age of the created individual (for human species only)
ageUnit	Unit in which the age value is defined. Default is year(s)
gestationalAge	Gestational age of the created individual (for human species only using the Preterm population). Default is 40 Weeks
gestationalAgeUnit	Unit in which the gestational age value is defined. Default is week(s)

moleculeOntogenies
: Optional list of MoleculeOntogeny that will be used to retrieve ontogeny information for molecules. A MoleculeOntogeny is an object with a molecule property (e.g. the name of the molecule as defined in your simulation) and an ontogeny property (e.g. the name of the predefined ontogeny to use for this molecule). The list of all available ontogenies can be accessed programmatically using the enum StandardOntogeny

seed
: Optional seed parameter to use to generate start values for the created individual algorithm.

Value

An IndividualCharacteristics object.

---

createPopulation *Creates an population using the PK-Sim Database*

Description

Creates an population using the PK-Sim Database

Usage

```
createPopulation(populationCharacteristics)
```

Arguments

populationCharacteristics
: Characteristics of the population to create as an instance of OriginData that are actually distributed parameters

Value

An list with three entries:

- population An instance of a population object.
- derivedParameters containing the parameter values modified indirectly by the algorithm. Those parameters are typically formula parameters.
- seed containing the seed value used to generate random values

createPopulationCharacteristics
Creates the population characteristics used to create a population

Description

Creates the population characteristics used to create a population

Usage

```
createPopulationCharacteristics(
  species,
  population = NULL,
  numberOfIndividuals,
  proportionOfFemales = 50,
  weightMin = NULL,
  weightMax = NULL,
  weightUnit = "kg",
  heightMin = NULL,
  heightMax = NULL,
  heightUnit = "cm",
  ageMin = NULL,
  ageMax = NULL,
  ageUnit = "year(s)",
  BMIMin = NULL,
  BMIMax = NULL,
  BMIUnit = "kg/m²",
  gestationalAgeMin = NULL,
  gestationalAgeMax = NULL,
  gestationalAgeUnit = "week(s)",
  moleculeOntogenies = NULL,
  seed = NULL
)
```

Arguments

species	Species of the individual as defined in PK-Sim (see Species enum)
population	Population to use to create the individual. This is required only when the species is Human. (See HumanPopulation enum)
numberOfIndividuals	Number of individuals in the population
proportionOfFemales	Proportions of females. Default is 50 (50%)
weightMin	min weight for the population (optional)
weightMax	max weight for the population (optional)

weightUnit	Unit in which the weight value is defined. Default is kg
heightMin	min height for the population (optional, for human species only)
heightMax	max height for the population (optional, for human species only)
heightUnit	Unit in which the height value is defined. Default is cm
ageMin	min age for the population (optional, for human species only)
ageMax	max age for the population (optional, for human species only)
ageUnit	Unit in which the age value is defined. Default is year(s)
BMIMin	min BMI for the population (optional, for human species only)
BMIMax	max BMI for the population (optional, for human species only)
BMIUnit	Unit in which the BMI value is defined. Default is kg/m2
gestationalAgeMin	min gestational age for the population (optional, for human species only)
gestationalAgeMax	max gestational age for the population (optional, for human species only)
gestationalAgeUnit	Unit in which the gestational age value is defined. Default is kg/m2
moleculeOntogenies	Optional list of MoleculeOntogeny that will be used to retrieve ontogeny information for molecules.
seed	Optional Seed parameter used to generate random values. This is only useful in order to reproduce the same population

Value

An instance of PopulationCharacteristics to be used in conjunction with createPopulation

---

createSimulationBatch *Creates and returns an instance of a* SimulationBatch *that can be used to efficiently vary parameters and initial values in a simulation*

Description

Creates and returns an instance of a SimulationBatch that can be used to efficiently vary parameters and initial values in a simulation

Usage

```
createSimulationBatch(
  simulation,
  parametersOrPaths = NULL,
  moleculesOrPaths = NULL
)
```

Arguments

simulation	Instance of a Simulation to simulate in a batch mode
parametersOrPaths	Parameter instances (element or vector) typically retrieved using getAllParametersMatching or parameter path (element or vector of strings) that will be varied in the simulation. (optional) When providing the paths, only absolute full paths are supported (i.e., no matching with '*' possible). If parametersOrPaths is NULL, you will not be able to set parameter values during batch run.
moleculesOrPaths	Molecule instances (element or vector) typically retrieved using getAllMoleculesMatching or molecule path (element or vector of strings) that will be varied in the simulation. (optional) When providing the paths, only absolute full paths are supported (i.e., no matching with '*' possible). If moleculesOrPaths is NULL, you will not be able to set molecule initial values during batch run.

Value

SimulationBatch that can be used to vary parameter values or molecule initial values and run simulation in an optimized manner

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Create a simulation batch that will allow batch run for one parameter value
simulationBatch <- createSimulationBatch(sim, "Organism|Liver|Volume")

# Create a simulation batch that will allow batch run for multiple parameter
# values and initial values
simulationBatch <- createSimulationBatch(
  sim,
  c("Organism|Liver|Volume", "R1|k1"),
  c("Organism|Liver|A")
)
```

DataAggregationMethods

Names of aggregation available for plotPopulationTimeProfile()

Description

Names of aggregation available for plotPopulationTimeProfile()

Usage

```
DataAggregationMethods
```

DataColumn *DataColumn*

Description

One column defined in a DataRepository

Super classes

rSharp::NetObject -> DotNetWrapper -> DataColumn

Active bindings

values Returns the values defined in the column

name Returns the name of the column (Read-Only)

unit The base unit in which the values are defined (Read-Only)

displayUnit The unit in which the values should be displayed

dimension The dimension of the values

molWeight Molecular weight of associated observed data in internal unit In no molecular weight is defined, the value is NULL

LLOQ Lower Limit Of Quantification. In no LLOQ is defined, the value is NULL

Methods

Public methods:

- DataColumn$print()

DataColumn$print(): Print the object to the console

Usage:

```
DataColumn$print(...)
```

Arguments:

... Rest arguments.

DataCombined *Object combining simulated and observed data*

Description

A class for storing simulated and/or observed in a single data frame, which can be further used in data wrangling or data visualization pipelines.

Additionally, it allows:

- Grouping different simulated and/or observed datasets.
- Transforming data with given offsets and scale factors.

Active bindings

`names` A vector of unique names of datasets contained in the `DataCombined` class instance.

`groupMap` A data frame specifying which datasets have been grouped together and the name and the nature (observed or simulated?) of the data. If a dataset was not assigned to any group, this is denoted by `NA` in the data frame.

`dataTransformations` A data frame with offset and scale factor values were specified by the user for each dataset.

Methods

Public methods:

- `DataCombined$addDataSets()`
- `DataCombined$addSimulationResults()`
- `DataCombined$setGroups()`
- `DataCombined$setDataTypes()`
- `DataCombined$removeGroupAssignment()`
- `DataCombined$setDataTransformations()`
- `DataCombined$toDataFrame()`
- `DataCombined$print()`
- `DataCombined$clone()`

`DataCombined$addDataSets()`**:** Adds observed data.

Usage:

```
DataCombined$addDataSets(dataSets, names = NULL, groups = NULL, silent = FALSE)
```

Arguments:

`dataSets` An instance (or a `list` of instances) of the `DataSet` class.

`names` A string or a `list` of strings assigning new names. These new names can be either for renaming `DataSet` objects, or for renaming quantities/paths in `SimulationResults` object. If an entity is not to be renamed, this can be specified as `NULL`. E.g., in `names = list("oldName1" = "newName1", "oldName2" = NULL)`), dataset with name `"oldName2"` will not be renamed. The list can either be named or unnamed. Names act as unique identifiers for data sets in the `DataCombined` object and, therefore, duplicate names are not allowed.

groups A string or a list of strings specifying group name corresponding to each data set. If an entry within the list is NULL, the corresponding data set is not assigned to any group (and the corresponding entry in the group column will be an NA). If provided, groups must have the same length as dataSets and/or simulationResults$quantityPath. If no grouping is specified for any of the dataset, the column group in the data frame output will be all NA.

silent A binary flag showing if warnings should be triggered when data sets are overwritten in the DataCombined object

Returns: DataCombined object containing observed data.

`DataCombined$addSimulationResults()`**:** Add simulated data using instance of SimulationResults class.

Usage:

```
DataCombined$addSimulationResults(
  simulationResults,
  quantitiesOrPaths = NULL,
  population = NULL,
  individualIds = NULL,
  names = NULL,
  groups = NULL,
  silent = FALSE
)
```

Arguments:

simulationResults Object of type SimulationResults produced by calling runSimulations on a Simulation object.

quantitiesOrPaths Quantity instances (element or vector) typically retrieved using getAllQuantitiesMatching or quantity path (element or vector of strings) for which the results are to be returned. (optional) When providing the paths, only absolute full paths are supported (i.e., no matching with '*' possible). If quantitiesOrPaths is NULL (default value), returns the results for all output defined in the results.

population population used to calculate the simulationResults (optional). This is used only to add the population covariates to the resulting data table.

individualIds numeric IDs of individuals for which the results should be extracted. By default, all individuals from the results are considered. If the individual with the provided ID is not found, the ID is ignored.

names A string or a list of strings assigning new names. These new names can be either for renaming DataSet objects, or for renaming quantities/paths in SimulationResults object. If an entity is not to be renamed, this can be specified as NULL. E.g., in names = list("oldName1" = "newName1", "oldName2" = NULL)), dataset with name "oldName2" will not be renamed. The list can either be named or unnamed. Names act as unique identifiers for data sets in the DataCombined object and, therefore, duplicate names are not allowed.

groups A string or a list of strings specifying group name corresponding to each data set. If an entry within the list is NULL, the corresponding data set is not assigned to any group (and the corresponding entry in the group column will be an NA). If provided, groups must have the same length as dataSets and/or simulationResults$quantityPath. If no grouping is specified for any of the dataset, the column group in the data frame output will be all NA.

`silent` A binary flag showing if warnings should be triggered when data sets are overwritten in the `DataCombined` object

Returns: `DataCombined` object containing simulated data.

`DataCombined$setGroups()`**:** Adds grouping information to (observed and/or simulated) datasets.

Usage:

```
DataCombined$setGroups(names, groups)
```

Arguments:

`names` A list of dataset names which need to be grouped. Note that if you have specified new `names` while adding datasets (using `$addDataSets()` and `$addSimulationResults()` methods), you will need to use these new names to specify group assignment. The same dataset can't be assigned to two different groupings in the *same* `$setGroups()` call. In other words, elements of `names` argument should be unique.

`groups` A list specifying which datasets belong to which group(s). Please note that the order in which groups are specified should match the order in which datasets were specified for `names` parameter. For example, if data sets are named `"x"`, `"y"`, `"z"`, and the desired groupings for them are, respectively, `"a"`, `"b"`, this can be specified as `names = list("x", "y"), groups = list("a", "b`. Datasets for which no grouping is to be specified, can be left out of the `groups` argument. The column `group` in the data frame output will be `NA` for such datasets. If you wish to remove an *existing* grouping assignment for a given dataset, you can specify it as following: `list("x" = NA)` or `list("x" = NULL)`. This will not change any of the other groupings.

Returns: `DataCombined` object with grouped datasets.

`DataCombined$setDataTypes()`**:** set the type of data (observed or simulated) for datasets.

Usage:

```
DataCombined$setDataTypes(names, dataTypes)
```

Arguments:

`names` a character vector of dataset names which dataTypes need to be changed.

`dataTypes` a character vector of dataTypes (`"observed"` or `"simulated"`) to be assigned to the datasets (in order of `names`.

Returns: `DataCombined` object with modified dataTypes datasets.

`DataCombined$removeGroupAssignment()`**:** Remove existing groupings for (observed and/or simulated) datasets.

Usage:

```
DataCombined$removeGroupAssignment(names)
```

Arguments:

`names` A list of dataset names whose group assignment needs to be removed. Note that if you have specified new `names` while adding datasets (using `$addDataSets()` and `$addSimulationResults()` methods), you will need to use these new names to specify group assignment. The elements of `names` argument should be unique.

Returns: `DataCombined` object with updated group assignments.

`DataCombined$setDataTransformations()`**:** Transform raw data with required offsets and scale factors.

Usage:

```
DataCombined$setDataTransformations(
  forNames = NULL,
  xOffsets = 0,
  yOffsets = 0,
  xScaleFactors = 1,
  yScaleFactors = 1,
  reset = FALSE
)
```

Arguments:

`forNames` A list of names specifying which observed datasets and/or paths in simulated dataset to transform with the specified transformations. Default is `NULL`, i.e., the transformations, if any specified, will be applied to all rows of the data frame.

`xOffsets`, `yOffsets`, `xScaleFactors`, `yScaleFactors` Either a single numeric value or a list of numeric values specifying offsets and scale factors to apply to raw values. The default offset is `0`, while default scale factor is `1`, i.e., the data will not be modified. If a list is specified, it should be the same length as `forNames` argument.

`reset` IF `TRUE`, only data transformations that are specified will be retained. Not specified transformations will be reset to their defaults. Default behavior is `FALSE`, e.g., setting only `xOffsets` will not reset `xScaleFactors` if those have been set previously.

Details: A data frame with respective raw quantities transformed using specified offset and scale factor values.

- For X and Y variables: `newValue = (rawValue + offset) * scaleFactor`
- For error term: `newErrorValue = rawErrorValue * scaleFactor`

`DataCombined$toDataFrame()`: A method to extract a tibble data frame of simulated and/or observed data (depending on instances of which classes have been added to the object).

Note that the order in which you enter different object doesn't matter because the returned data frame is arranged alphabetically by dataset name.

Usage:

```
DataCombined$toDataFrame()
```

Returns: In the returned tibble data frame, the following columns will always be present: name - group - dataType - xValues - xDimension - xUnit - yValues - yErrorValues - yDimension - yUnit - yErrorType - yErrorUnit - molWeight

`DataCombined$print()`: Print the object to the console.

Usage:

```
DataCombined$print()
```

`DataCombined$clone()`: The objects of this class are cloneable with this method.

Usage:

```
DataCombined$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

The molecular weight (in `molWeight` column) is in `g/mol` units.

See Also

Other data-combined: `addResidualColumn()`, `calculateResiduals()`, `convertUnits()`

Examples

```
# simulated data
simFilePath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim <- loadSimulation(simFilePath)
simResults <- runSimulations(sim)[[1]]
outputPath <- "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)"

# observed data
obsData <- lapply(
  c("ObsDataAciclovir_1.pkml", "ObsDataAciclovir_2.pkml", "ObsDataAciclovir_3.pkml"),
  function(x) loadDataSetFromPKML(system.file("extdata", x, package = "ospsuite"))
)
names(obsData) <- lapply(obsData, function(x) x$name)


# Create a new instance of `DataCombined` class
myDataCombined <- DataCombined$new()

# Add simulated results
myDataCombined$addSimulationResults(
  simulationResults = simResults,
  quantitiesOrPaths = outputPath,
  groups = "Aciclovir PVB"
)

# Add observed data set
myDataCombined$addDataSets(obsData$`Vergin 1995.Iv`, groups = "Aciclovir PVB")

# Looking at group mappings
myDataCombined$groupMap

# Looking at the applied transformations
myDataCombined$dataTransformations

# Accessing the combined data frame
myDataCombined$toDataFrame()
```

`dataCombinedAciclovir` *dataCombinedAciclovir*

Description

Example `DataCombined` object for Aciclovir created from the code chunk below and that is re-used throughout the package examples and documentation

```
# simulated data
simFilePath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim <- loadSimulation(simFilePath)
simResults <- runSimulations(sim)[[1]]
outputPath <- "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)"

# observed data
obsData <- lapply(
 c("ObsDataAciclovir_1.pkml", "ObsDataAciclovir_2.pkml", "ObsDataAciclovir_3.pkml"),
 function(x) loadDataSetFromPKML(system.file("extdata", x, package = "ospsuite"))
)
names(obsData) <- lapply(obsData, function(x) x$name)

# Create a new instance of `DataCombined` class
dataCombinedAciclovir <- DataCombined$new()

# Add simulated results
dataCombinedAciclovir$addSimulationResults(
  simulationResults = simResults,
  quantitiesOrPaths = outputPath,
  groups = "Aciclovir PVB"
)

# Add observed data set
dataCombinedAciclovir$addDataSets(obsData$`Vergin 1995.Iv`, groups = "Aciclovir PVB")
```

Usage

```
dataCombinedAciclovir
```

Format

`dataCombinedAciclovir`**:**

A `DataCombined` object with 2 sets

simulated Simulated Aciclovir data

observed Observed data from Vergin 1995.Iv

Source

Working with DataCombined class

DataErrorType *Supported types of the error*

Description

Supported types of the error

Usage

```
DataErrorType
```

DataImporterConfiguration *DataImporterConfiguration*

Description

Configuration of data import from excel or csv files. To be used with `loadDataSetsFromExcel`.

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `DataImporterConfiguration`

Active bindings

`timeColumn` Name of the column for time values

`timeUnit` If `isTimeUnitFromColumn` is `FALSE`, unit of the values in time column If `isTimeUnitFromColumn` is `TRUE`, name of the column with units of the values in time column.

`isTimeUnitFromColumn` If `TRUE`, units of the values in time column are defined in the column `timeUnit`. If `FALSE`, the unit is defined by the value of `timeUnit`.

`measurementColumn` Name of the column for measurement values

`lloqColumn` Name of the column for LLOQ values If the column name is not set (value `NULL`), LLOQ values will be imported from the measurement column if values are written in the form '< xxx' (e.g., '<0.001'). Otherwise, the values will be imported from the specified column

`measurementDimension` If `isMeasurementUnitFromColumn` is `FALSE`, dimension of the values in measurement column If `isMeasurementUnitFromColumn` is `TRUE`, the dimension is guessed from the unit defined in the column `measurementUnit` during import process and `$measurementDimension` is `NULL`. When changing dimension, the unit is set to the base unit of this dimension.

`measurementUnit` If `isMeasurementUnitFromColumn` is `FALSE`, unit of the values in measurement column If `isMeasurementUnitFromColumn` is `TRUE`, name of the column with units of the values in measurement column

`isMeasurementUnitFromColumn` If `TRUE`, units of the values in measurement column are defined in the column `measurementUnit`. If `FALSE`, the unit is defined by the value of `measurementUnit`.

`errorColumn` Name of the column for measurement error values If no error column is defined, the value is `NULL`. Setting the value to `NULL` removes an existing error column.

`errorUnit` If `isMeasurementUnitFromColumn` is `FALSE`, unit of the values in the error column If `isMeasurementUnitFromColumn` is `TRUE`, name of the column with units of the values in error column If no error column is present, the value is `NULL`

`errorType` Type of the measurement error values. See enum `DataErrorType` for possible values If no error column is present, the value is `NULL`

`groupingColumns` Column names by which the data will be grouped

`sheets` Names of the sheets (list of strings) of the excel workbook for which the configuration will be applied.

`namingPattern` Regular expression used for naming of loaded data sets. Words between curly brackets (e.g. `{Group Id}`) will be replaced by the value in the corresponding column. Further keywords are `{Source}` for the file name and `{Sheet}` for sheet name.

Methods

Public methods:

- `DataImporterConfiguration$new()`
- `DataImporterConfiguration$saveConfiguration()`
- `DataImporterConfiguration$addGroupingColumn()`
- `DataImporterConfiguration$removeGroupingColumn()`
- `DataImporterConfiguration$print()`

`DataImporterConfiguration$new()`: Initialize a new instance of the class

Usage:

```
DataImporterConfiguration$new(netObject = NULL)
```

Arguments:

`netObject` A `NetObject` with the reference to .NET DataImporterConfiguration object If `NULL` (default), an empty configuration with columns "Time" and "Measurement" is created.

Returns: A new `DataImporterConfiguration` object.

`DataImporterConfiguration$saveConfiguration()`: Save configuration to a XML file that can be used in PK-Sim/MoBi

Usage:

```
DataImporterConfiguration$saveConfiguration(filePath)
```

Arguments:

`filePath` Path (incl. file name) to the location where the configuration will be exported to.

`DataImporterConfiguration$addGroupingColumn()`: Add a column for grouping the data sets

Usage:

```
DataImporterConfiguration$addGroupingColumn(column)
```

Arguments:

`column` Name of the column

`DataImporterConfiguration$removeGroupingColumn()`: Remove a column for grouping the data sets

Usage:

```
DataImporterConfiguration$removeGroupingColumn(column)
```

Arguments:

`column` Name of the column

`DataImporterConfiguration$print()`: Print the object to the console

Usage:

```
DataImporterConfiguration$print(...)
```

Arguments:

`...` Rest arguments.

`DataRepository` *DataRepository*

Description

An object typically holding observed data

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `DataRepository`

Active bindings

`name` The name of the object.

`baseGrid` Returns the base column for the data repository (typically time column).

`columns` Returns all columns (including baseGrid) defined in the data repository.

`allButBaseGrid` Returns all columns excluding baseGrid defined on the data repository.

`metaData` Returns a named list of meta data defined for the data repository. where the name is the name of the metaData and the value is the meta data value.

Methods

Public methods:

- `DataRepository$addColumn()`
- `DataRepository$new()`
- `DataRepository$print()`
- `DataRepository$addMetaData()`
- `DataRepository$removeMetaData()`

`DataRepository$addColumn()`: Adds a column to the data repository

Usage:

```
DataRepository$addColumn(column)
```

Arguments:

`column` Column to add

`DataRepository$new()`: Initialize a new instance of the class

Usage:

```
DataRepository$new(netObj = NULL)
```

Arguments:

`netObj` Optional `NetObject` to the pointer of the underlying `DataRepository`. If it is not provided, a new instance will be created

Returns: A new `DataRepository` object.

`DataRepository$print()`: Print the object to the console

Usage:

```
DataRepository$print(...)
```

Arguments:

`...` Rest arguments.

`DataRepository$addMetaData()`: Adds a new entry to meta data list or changes its value if the name is already present.

Usage:

```
DataRepository$addMetaData(name, value)
```

Arguments:

`name` Name of new meta data list entry

`value` Value of new meta data list entry

`DataRepository$removeMetaData()`: Removes the meta data entry in the list if one is defined with this name

Usage:

```
DataRepository$removeMetaData(name)
```

Arguments:

`name` Name of meta data entry to delete

DataSet *DataSet*

Description

A class for storage of numerical x- and y-value pairs and optional error for y-values.

Super classes

rSharp::NetObject -> DotNetWrapper -> DataSet

Active bindings

`name` The name of the DataSet

`dataRepository` The underlying DataRepository object

`xDimension` Dimension in which the xValues are defined

`xUnit` Unit in which the xValues are defined

`xValues` Values stored in the xUnit. This field is read-only. Use `$setValues()` to change the values.

`yDimension` Dimension in which the yValues are defined

`yUnit` Unit in which the yValues are defined

`yValues` Values stored in the yUnit. This field is read-only. Use `$setValues()` to change the values.

`yErrorType` Type of the error - geometric or arithmetic. When changing from arithmetic to geometric error, the values are considered in as fraction (1 = 100%). When changing from geometric to arithmetic, the values are set to the same unit as `yErrorUnit`. In case no yError is defined, the value is `NULL` and cannot be changed

`yErrorUnit` Unit in which the yErrorValues are defined. For arithmetic error, the unit must be valid for `yDimension`. For geometric error, the unit must be valid for `Dimensionless`. In case no yError is defined, the value is `NULL` and cannot be changed

`yErrorValues` Values of error stored in the yErrorUnit unit. This field is read-only. Use `$setValues()` to change the values. In case no yError is defined, the value is `NULL` and cannot be changed. Use `$setValues()` to change the values.

`molWeight` Molecular weight of the yValues in g/mol

`LLOQ` Lower Limit Of Quantification. Value in yUnit associated with the yValues

`metaData` Returns a named list of meta data defined for the data set.

Methods

Public methods:

- DataSet$new()
- DataSet$addMetaData()
- DataSet$removeMetaData()

- DataSet$setValues()
- DataSet$print()

`DataSet$new()`: Initialize a new instance of the class. Either create a `DataSet` from a `DataRepository` (e.g. loaded from a PKML) or an empty `DataSet`. In case of an empty `DataSet`, a `name` must be provided.

Usage:

```
DataSet$new(name = NULL, dataRepository = NULL)
```

Arguments:

`name` Name of the `DataSet` if created from scratch (no `dataRepository`) provided. Ignored if `dataRepository` is not `NULL`.

`dataRepository` Instance of the `DataRepository` object to wrap. If `NULL`, an empty `DataRepository` is created.

Returns: A new `DataSet` object.

`DataSet$addMetaData()`: Adds a new entry to meta data list or changes its value if the name is already present.

Usage:

```
DataSet$addMetaData(name, value)
```

Arguments:

`name` Name of new meta data list entry

`value` Value of new meta data list entry

`DataSet$removeMetaData()`: Removes the meta data entry in the list if one is defined with this name

Usage:

```
DataSet$removeMetaData(name)
```

Arguments:

`name` Name of meta data entry to delete

`DataSet$setValues()`: Sets the xValues, yValues, and (optionally) yErrorValues into the dataSet. Note: xValues, yValues and yErrorValues must have the same length

Usage:

```
DataSet$setValues(xValues, yValues, yErrorValues = NULL)
```

Arguments:

`xValues` xValues to use

`yValues` yValues to use

`yErrorValues` Optional error values associated with yValues

`DataSet$print()`: Print the object to the console

Usage:

```
DataSet$print(...)
```

Arguments:

`...` Rest arguments.

dataSetsFromDataFrame *Creates a list of* DataSet *objects from a* data.frame

Description

Creates a list of DataSet objects from a data.frame

Usage

```
dataSetsFromDataFrame(data)
```

Arguments

data	A data.frame with at minimum the columns name, xValues, and yValues. Optional standard columns: yErrorValues, xDimension, xUnit, yDimension, yUnit, yErrorType, yErrorUnit, molWeight, lloq. Any additional columns are treated as meta data entries.

Details

Creates DataSet objects from a data.frame with the same structure as returned by dataSetToDataFrame(). Each unique value in the name column results in one DataSet object. Any columns beyond the standard columns (name, xValues, yValues, yErrorValues, xDimension, xUnit, yDimension, yUnit, yErrorType, yErrorUnit, molWeight, lloq) will be added as meta data.

Value

A named list of DataSet objects, named by the name column.

Examples

```
dataSet <- DataSet$new(name = "MyData")
dataSet$setValues(xValues = c(1, 2, 3), yValues = c(10, 20, 30))
df <- dataSetToDataFrame(dataSet)
dataSets <- dataSetsFromDataFrame(df)
```

dataSetToDataFrame *Converts a list of* DataSet *objects to a data.frame*

Description

Converts a list of DataSet objects to a data.frame

Usage

```
dataSetToDataFrame(dataSets)

dataSetToTibble(dataSets, names = NULL)
```

Arguments

dataSets	A list of DataSet objects or a single DataSet
names	Optional character vector of custom names to assign to the datasets. If provided, must have the same length as the number of DataSet objects. This allows renaming datasets, which is particularly useful when multiple datasets have the same original name.

Value

DataSet objects as data.frame with columns name, xValues, yValues, yErrorValues, xDimension, xUnit, yDimension, yUnit, yErrorType, yErrorUnit, molWeight, lloq, and a column for each meta data that is present in any DataSet.

Examples

```
# Create datasets with duplicate names
ds1 <- DataSet$new(name = "Obs")
ds1$setValues(xValues = c(1, 2), yValues = c(10, 20))

ds2 <- DataSet$new(name = "Obs")
ds2$setValues(xValues = c(3, 4), yValues = c(30, 40))

# Convert to tibble with custom names
tibble_data <- dataSetToTibble(list(ds1, ds2), names = c("Study1", "Study2"))
unique(tibble_data$name) # Returns c("Study1", "Study2")
```

DefaultPlotConfiguration

Plot configuration for OSP plots

Description

R6 configuration class defining aesthetic properties of plots that can be created with plotIndividualTimeProfile(), plotPopulationTimeProfile(), plotObservedVsSimulated(), and plotResidualsVsTime().

To interactively explore various aesthetic properties and appearance of plots with these properties, you can use the Shiny app from {tlf} package.

The following sections provide more details on how to customize it further.

Specifying aesthetic properties

Aesthetic mappings describe how groups are mapped to visual properties (color, shape, size, etc.) of the geometries included in the plot (e.g. point, line, ribbon, etc.).

The supported values for each property can be seen using {tlf} lists:

- color, fill: tlf::ColorMaps

- shape: `tlf::Shapes`
- legend position: `tlf::LegendPositions`
- alignments: `tlf::Alignments`
- linetype: `tlf::Linetypes`

For example, all parameters related to color (`titleColor`, `yAxisLabelTicksColor`, etc.) accept any of the palettes available in `tlf::ColorMaps` (e.g. `tlf::ColorMaps$ospDefault`).

Note that these are named lists, and, therefore, if you want to assign a specific element from a list to an object's public field, you will have to extract that element first.

For example, if you want to specify that the legend position should be outside the plot to the left and at bottom, you will have to do the following:

```
myPlotConfiguration <- DefaultPlotConfiguration$new()
myPlotConfiguration$legendPosition <- tlf::LegendPositions$outsideBottomLeft
```

Of course, the extracted element doesn't have to be a single value, and can also be an atomic vector. For example, if you want to assign a different line type to each group in a profile plot, you will have to assign a vector of line types.

```
myPlotConfiguration <- DefaultPlotConfiguration$new()
myPlotConfiguration$linesLinetype <- names(tlf::Linetypes)
```

If there are more number of elements in the vector than the number of groups, the additional elements will be ignored.

Specifying units

The available units for `x`-and `y`-axes depend on the dimensions of these quantities (`ospsuite::ospDimensions`). Supported units can be seen with `ospsuite::ospUnits`.

Specifying fonts

A font is a particular set of glyphs (character shapes), differentiated from other fonts in the same family by additional properties such as stroke weight, slant, relative width, etc.

A font face (aka typeface) is the design of lettering, characterized by variations in size, weight (e.g. bold), slope (e.g. italic), width (e.g. condensed), and so on. The available font faces can seen using `tlf::FontFaces` list.

A font family is a grouping of fonts defined by shared design styles.

The available font families will depend on which fonts have been installed on your computer. This information can be extracted by running the following code:

```
# install.packages("systemfonts")
library(systemfonts)
system_fonts()
```

Specifying scaling

Transformations for both x- and y-axes can be (independently) specified. The default is linear for both axes.

The available transformations can be seen in the `tlf::Scaling` list.

Specifying tick labels

`tlf::TickLabelTransforms` lists of all available tick label transformations. For example, selecting `tlf::TickLabelTransforms$identity` will display tick labels as they are, while selecting `tlf::TickLabelTransforms$log` will display tick labels in logarithmic scale format.

Saving plot

By default, the plots will be shown in plot pane of your IDE, but the plots can also be saved to a file using the `ggplot2::ggsave()` function.

```
myPlot <- plotIndividualTimeProfile(myDataComb, myPC)
ggplot2::ggsave(filename = "plot_1.png", plot = myPlot)
```

Public fields

`xUnit`, `yUnit` Units for quantities plotted on x- and y-axes, respectively.

`title`, `subtitle`, `caption`, `xLabel`, `yLabel`, `legendTitle`, `watermark` A character string providing plot annotations for plot title, subtitle, caption, x-axis label, y-axis label, plot legend, watermark, respectively.

`titleColor`, `titleSize`, `titleFontFace`, `titleFontFamily`, `titleAngle`, `titleAlign`, `titleMargin` Aesthetic properties for the plot title.

`subtitleColor`, `subtitleSize`, `subtitleFontFace`, `subtitleFontFamily`, `subtitleAngle`, `subtitleAlign`, `subti` Aesthetic properties for the plot subtitle.

`captionColor`, `captionSize`, `captionFontFace`, `captionFontFamily`, `captionAngle`, `captionAlign`, `captionMarg` Aesthetic properties for the plot caption.

`xLabelColor`, `xLabelSize`, `xLabelFontFace`, `xLabelFontFamily`, `xLabelAngle`, `xLabelAlign`, `xLabelMargin` Aesthetic properties for the plot xLabel.

`yLabelColor`, `yLabelSize`, `yLabelFontFace`, `yLabelFontFamily`, `yLabelAngle`, `yLabelAlign`, `yLabelMargin` Aesthetic properties for the plot yLabel.

`legendPosition` A character string defining the legend position. Available options can be seen using `tlf::LegendPositions` list.

`legendTitleSize`, `legendTitleColor`, `legendTitleFontFamily`, `legendTitleFontFace`, `legendTitleAngle`, `leger` Aesthetic properties for the legend title.

`legendKeysSize`, `legendKeysColor`, `legendKeysFontFamily`, `legendKeysFontFace`, `legendKeysAngle`, `legendKeys` Aesthetic properties for the legend caption.

`legendBackgroundColor`, `legendBackgroundAlpha`, `legendBorderColor`, `legendBorderType`, `legendBorderSize` Aesthetic properties for the legend box

`xAxisTicksLabels`, `xAxisLabelTicksSize`, `xAxisLabelTicksColor`, `xAxisLabelTicksFontFamily`, `xAxisLabelTic` Aesthetic properties for the x-axis label.

`yAxisTicksLabels`, `yAxisLabelTicksSize`, `yAxisLabelTicksColor`, `yAxisLabelTicksFontFamily`, `yAxisLabelTic`
Aesthetic properties for the y-axis label.

`xAxisLimits`, `yAxisLimits` A numeric vector of axis limits for the x-and y-axis, respectively. This will preserve all data points but zoom in the plot.

`xValuesLimits`, `yValuesLimits` A numeric vector of values limits for the x-and y-axis, respectively. This will filter out the data points outside the specified ranges before plotting.

`xAxisTicks`, `yAxisTicks` A numeric vector or a function defining where to position x-and y-axis ticks, respectively.

`xAxisScale`, `yAxisScale` A character string defining axis scale. Available options can be seen using `tlf::Scaling` list.

`watermarkSize`, `watermarkColor`, `watermarkFontFamily`, `watermarkFontFace`, `watermarkAngle`, `watermarkAlign`,
A character string specifying the aesthetic properties for the watermark.

`plotBackgroundFill`, `plotBackgroundColor`, `plotBackgroundSize`, `plotBackgroundLinetype`
A character string specifying the aesthetic properties for the plot background.

`plotPanelBackgroundFill`, `plotPanelBackgroundColor`, `plotPanelBackgroundSize`, `plotPanelBackgroundLinety`
A character string specifying the aesthetic properties for the plot panel (inside of plot) background.

`xAxisColor`, `xAxisSize`, `xAxisLinetype` A character string specifying the aesthetic properties for the x-axis.

`yAxisColor`, `yAxisSize`, `yAxisLinetype` A character string specifying the aesthetic properties for the y-axis.

`xGridColor`, `xGridSize`, `xGridLinetype` A character string specifying the aesthetic properties for the x-axis grid.

`yGridColor`, `yGridSize`, `yGridLinetype` A character string specifying the aesthetic properties for the y-axis grid.

`linesColor`, `linesSize`, `linesLinetype`, `linesAlpha` A selection key or values for choice of color, fill, shape, size, linetype, alpha, respectively, for lines.

`pointsColor`, `pointsShape`, `pointsSize`, `pointsAlpha` A selection key or values for choice of color, fill, shape, size, linetype, alpha, respectively, for points.

`ribbonsFill`, `ribbonsSize`, `ribbonsLinetype`, `ribbonsAlpha` A selection key or values for choice of color, fill, shape, size, linetype, alpha, respectively, for ribbons.

`errorbarsSize`, `errorbarsLinetype`, `errorbarsAlpha`, `errorbarsCapSize` A selection key or values for choice of color, fill, shape, size, linetype, alpha, cap width/height, respectively, for error bars.

`displayLLOQ` A Boolean controlling display Lower Limit of Quantification lines. Default to True.

`lloqDirection` A string controlling how the LLOQ lines are plotted. Can be "vertical", "horizontal" or "both". Default to NULL to respect specific plot configurations.

`foldLinesLegend` A Boolean controlling the drawing of the fold lines in the legend. Default to False.

`foldLinesLegendDiagonal` A Boolean controlling whether the fold lines legend should be horizontal or diagonal lines.

Methods

Public methods:

- DefaultPlotConfiguration$clone()

DefaultPlotConfiguration$clone(): The objects of this class are cloneable with this method.

Usage:

```
DefaultPlotConfiguration$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other plotting: plotIndividualTimeProfile(), plotObservedVsSimulated(), plotPopulationTimeProfile(), plotResidualsVsSimulated(), plotResidualsVsTime()

Examples

```
# Create a new instance of this class
myPlotConfiguration <- DefaultPlotConfiguration$new()

# Change defaults
myPlotConfiguration$title <- "My Plot Title"
myPlotConfiguration$pointsSize <- 2.5
myPlotConfiguration$legendTitle <- "My Legend Title"

# Checking new values
myPlotConfiguration$pointsSize

# To check all default values, you can print the object
myPlotConfiguration
```

DotNetWrapper *Wrapper class for .NET objects*

Description

Wrapper class for .NET objects

Super class

rSharp::NetObject -> DotNetWrapper

Methods

Public methods:

- DotNetWrapper$new()

DotNetWrapper$new(): Initialize a new instance of the class

Usage:

```
DotNetWrapper$new(netObject)
```

Arguments:

netObject An rSharp::NetObject object.

Returns: A new Molecule object.

Examples

```
sim <- loadSimulation(system.file("extdata", "simple.pkml", package = "ospsuite"))

# looking at a reference to `.NET` simulation object
sim$pointer

# create a new instance of `DotNetWrapper` class using this reference
DotNetWrapper$new(sim)
```

Entity *Entity*

Description

Abstract wrapper for an OSPSuite.Core Entity class

Super classes

rSharp::NetObject -> DotNetWrapper -> ObjectBase -> Entity

Active bindings

path The path of the entity in the container hierarchy without the simulation name. (read-only)

fullPath Same as path, but with the simulation name. (read-only)

parentContainer Returns a new wrapper instance to the .NET parent container. Multiple call to this method will always return the same instance. However two children of the same parent will return two different instances of Container pointing to the same .NET container

exportIndividualSimulations

Export simulation PKMLs for given individualIds. *Each pkml file will contain the original simulation updated with parameters of the corresponding individual.*

Description

Export simulation PKMLs for given individualIds. Each pkml file will contain the original simulation updated with parameters of the corresponding individual.

Usage

```
exportIndividualSimulations(
  population,
  individualIds,
  outputFolder,
  simulation
)
```

Arguments

population	A population object typically loaded with loadPopulation
individualIds	Ids of individual (single value or array) to export
outputFolder	Folder where the individual simulations will be exported. File format will be simulationName_individualId
simulation	Simulation uses to generate PKML files

Value

An array containing the path of all exported simulations.

Examples

```
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

popPath <- system.file("extdata", "pop.csv", package = "ospsuite")
population <- loadPopulation(popPath)

exportIndividualSimulations(population, c(1, 2), tempdir(), sim)
```

exportPKAnalysesToCSV *Saves the pK-analyses to csv file*

Description

Saves the pK-analyses to csv file

Usage

```
exportPKAnalysesToCSV(pkAnalyses, filePath)
```

Arguments

pkAnalyses	pK-Analyses to export (typically calculated using calculatePKAnalyses or imported from file)
filePath	Full path where the pK-Analyses will be saved.

exportPopulationToCSV *Saves the population to csv file*

Description

Saves the population to csv file

Usage

```
exportPopulationToCSV(population, filePath)
```

Arguments

population	Population to export to csv (typically imported from file using loadPopulation)
filePath	Full path where the population will be saved.

Examples

```
csvPath <- system.file("extdata", "pop.csv", package = "ospsuite")

# Load the population
population <- loadPopulation(csvPath)

# Exports the population
exportPopulationToCSV(population, tempfile())
```

exportProjectToSnapshot *Export a project to a snapshot*

Description

Converts one or more project files into snapshot files (`.json`) and writes them to an output directory.

Only PK-Sim projects (`.pksim5`) are supported for now. Support for MoBi projects is planned.

Usage

```
exportProjectToSnapshot(..., output = ".")
```

Arguments

`...`	character strings, path to project files (`.pksim5`) or a directory containing project files to convert.
`output`	character string, path to the output directory where to write the converted snapshot files.

Examples

```
## Not run:
exportProjectToSnapshot("path/to/project.pksim5", output = "path/to/output")

## End(Not run)
```

exportResultsToCSV *Saves the simulation results to csv file*

Description

Saves the simulation results to csv file

Usage

```
exportResultsToCSV(results, filePath)
```

Arguments

`results`	Results to export (typically calculated using `runSimulations` or imported from file).
`filePath`	Full path where the results will be saved.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath)

# Add some outputs to the simulation
addOutputs("Organism|**|*", sim)

# Run the simulation
results <- runSimulations(sim)[[1]]

# Export the results to csv file
exportResultsToCSV(results, tempfile())
```

`exportSensitivityAnalysisResultsToCSV`
Saves the simulation analysis results to csv file

Description

Saves the simulation analysis results to csv file

Usage

```
exportSensitivityAnalysisResultsToCSV(results, filePath)
```

Arguments

`results`	Results to export (typically calculated using `runSensitivityAnalysis` or imported from file)
`filePath`	Full path where the results will be saved.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath)

# Create a new sensitivity object for the simulation
sensitivity <- SensitivityAnalysis$new(sim)

# Runs the sensitivity analysis
results <- runSensitivityAnalysis(sensitivity)

# Export the results to csv file
exportSensitivityAnalysisResultsToCSV(results, tempfile())
```

`exportSteadyStateToXLS` *Export steady-state to Excel in the format that can be imported in MoBi.*

Description

Export steady-state to Excel in the format that can be imported in MoBi.

Usage

```
exportSteadyStateToXLS(
  simulation,
  quantitiesPaths = NULL,
  resultsXLSPath = "",
  steadyStateTime = NULL,
  ignoreIfFormula = TRUE,
  lowerThreshold = 1e-15,
  simulationRunOptions = NULL
)
```

Arguments

`simulation`	A `Simulation` object for which the steady-state will be simulated. In contrast to `getSteadyState()`, only one simulation is supported.
`quantitiesPaths`	List of quantity paths (molecules and/or parameters) for which the steady-state will be simulated. If `NULL` (default), all molecules and state variable parameters are considered. The same list is applied for all simulations.
`resultsXLSPath`	Path to the xls-file where the results will be written to. If the file does not exist, a new file is created. If no path is provided, the file will be created in the same directory where the model file is located. The name of the file will be `<SimulationFileName>_SS.xlsx`.
`steadyStateTime`	Simulation time (minutes). In `NULL` (default), the default simulation time is the start time of the last application plus three days. The simulated time must be long enough for the system to reach a steady-state. Either a single value (will be applied for all simulations), or a list of values specific for each simulation. In latter case, must have equal size as `simulations`. When providing a list, `NULL` is allowed to calculate the time based on the last application.
`ignoreIfFormula`	If `TRUE` (default), species and parameters with initial values defined by a formula are not included.
`lowerThreshold`	Numerical value (in default unit of the output). Any steady-state values with absolute value below this threshold are considered as numerical noise and replaced by 0 (i.e., values in the interval `[-lowerThreshold, lowerThreshold]`). If `lowerThreshold` is `NULL`, no cut-off is applied. Default value is 1e-15.

simulationRunOptions
: Optional instance of a SimulationRunOptions used during the simulation run.

Details

Simulates a given model to its steady-state and creates an Excel-file with the end values of molecules amounts in all containers and parameter values that have a right-hand-side (state variable parameters). The excel file contains two sheets - one for the molecules and one for the parameters.

Value

An openxlsx workbook object.

Formula *Formula*

Description

A formula of the model (Typically related to a Quantity such as a parameter)

A table formula of the model (Typically related to a Quantity such as a parameter)

Super classes

rSharp::NetObject -> DotNetWrapper -> ObjectBase -> Formula

Active bindings

isTable Is this a table formula (Read-Only)

isTableWithOffSet Is this a table formula with Offset (Read-Only)

isTableWithXArgument Is this a table formula with xArgs (typically time, or pH) (Read-Only)

isConstant Is this a constant formula (Read-Only)

isExplicit Is this an explicit formula (Read-Only)

isDistributed Is this a distributed formula (Read-Only)

dimension The dimension in which the quantity is defined (Read-Only)

formulaString Returns the formula as a string for an ExplicitFormula or NULL otherwise (Read-Only).

Methods

Public methods:

- Formula$print()
- Formula$printFormula()

Formula$print(): Print the object to the console

Usage:

`Formula$print(...)`

Arguments:

`...` Rest arguments.

`Formula$printFormula()`: Print the formula to the console without the name of the class

Usage:

```
Formula$printFormula()
```

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `ObjectBase` -> `Formula` -> `TableFormula`

Active bindings

`allPoints` Returns all points defined in the table formula for a `TableFormula` or `NULL` otherwise (Read-Only).

`useDerivedValues` Indicates whether table values should be derived during solving. the ODE system. Default value is `TRUE`

`xDimension` The dimension in which the x values are defined (Read-Only).

Methods

Public methods:

- `TableFormula$addPoints()`
- `TableFormula$removePoint()`
- `TableFormula$clearPoints()`
- `TableFormula$setPoints()`
- `TableFormula$print()`
- `TableFormula$valueAt()`
- `TableFormula$printFormula()`

`TableFormula$addPoints()`: Adds one or more points to a table

Usage:

```
TableFormula$addPoints(xValues, yValues)
```

Arguments:

`xValues` x values (single value or array) in base unit for XDimension

`yValues` y values (single value or array) in base unit for Dimension

`TableFormula$removePoint()`: Remove the point having the same x and y from the table

Usage:

```
TableFormula$removePoint(xValue, yValue)
```

Arguments:

`xValue` xValue value in base unit for XDimension

`yValue` yValue value in base unit for Dimension

`TableFormula$clearPoints()`: Remove all points from the table

Usage:

```
TableFormula$clearPoints()
```

`TableFormula$setPoints()`: Replace all points defined in the table with the new values given. This is a convenience method for calling `clearPoints` and `addPoints`

Usage:

```
TableFormula$setPoints(xValues, yValues)
```

Arguments:

`xValues` x values (single value or array) in base unit for XDimension

`yValues` y values (single value or array) in base unit for Dimension

`TableFormula$print()`: Print the object to the console

Usage:

```
TableFormula$print(...)
```

Arguments:

`...` Rest arguments.

`TableFormula$valueAt()`: Returns the y defined for the x value in base unit. If not exact match is found, value will be interpolated between two existing points If the table contains no point, 0 is returned

Usage:

```
TableFormula$valueAt(xValue)
```

Arguments:

`xValue` x value for in base unit for which the yValue should be returned

`TableFormula$printFormula()`: Print the formula to the console

Usage:

```
TableFormula$printFormula()
```

`Gender` *Default genders defined in PK-Sim*

Description

Default genders defined in PK-Sim

Usage

```
Gender
```

getAllContainerPathsIn

Retrieves the path of all containers defined in the container and all its children

Description

Retrieves the path of all containers defined in the container and all its children

Usage

```
getAllContainerPathsIn(container)
```

Arguments

container	A Container or Simulation used to find the parameters

Value

An array with one entry per container defined in the container

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Returns the path of all molecules defined in the simulation
moleculePaths <- getAllContainerPathsIn(sim)
```

getAllContainersMatching

Retrieve all sub containers of a parent container (simulation or container instance) matching the given path criteria

Description

Retrieve all sub containers of a parent container (simulation or container instance) matching the given path criteria

Usage

```
getAllContainersMatching(paths, container)
```

Arguments

paths	A vector of strings representing the paths relative to the container
container	A Container or Simulation used to find the containers

Value

A list of containers matching the path criteria. The list is empty if no containers matching were found.

See Also

loadSimulation() and getContainer() to create objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Return all `Intracellular` containers defined in all direct containers of the organism
containers <- getAllContainersMatching("Organism|*|Intracellular", sim)

# Return all `Intracellular` containers defined in all direct containers of the organism
# and the container "Interstitial" under 'Organism|Brain'
paths <- c("Organism|*|Intracellular", "Organism|Brain|Interstitial")
containers <- getAllContainersMatching(paths, sim)

# Returns all `Intracellular` containers defined in `Organism` and all its subcontainers
containers <- getAllContainersMatching("Organism|**|Intracellular", sim)
```

---

getAllMoleculePathsIn *Retrieves the paths of all molecules defined in the container and all its children*

Description

Retrieves the paths of all molecules defined in the container and all its children

Usage

```
getAllMoleculePathsIn(container)
```

Arguments

container	A Container or Simulation used to find the parameters

Value

An array with one entry per molecule defined in the container

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Returns the path of all molecules defined in the simulation
moleculePaths <- getAllMoleculePathsIn(sim)
```

---

getAllMoleculesMatching

Retrieve all molecules of a container (simulation or container instance) matching the given path criteria

---

Description

Retrieve all molecules of a container (simulation or container instance) matching the given path criteria

Usage

```
getAllMoleculesMatching(paths, container)
```

Arguments

paths	A vector of strings representing the paths relative to the container
container	A Container or Simulation used to find the molecules

Value

A list of molecules matching the path criteria. The list is empty if no molecules matching were found.

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Return all `A` molecules defined in all direct containers of the organism
molecules <- getAllMoleculesMatching("Organism|*|A", sim)

# Return all `A` molecules defined in all direct containers of the organism
# and the molecule `B`` of the container 'Liver'
paths <- c("Organism|*|A", "Organism|Liver|B")
molecules <- getAllMoleculesMatching(paths, sim)

# Returns all `A` molecules defined in `Organism` and all its subcontainers
molecules <- getAllMoleculesMatching("Organism|**|A", sim)
```

---

getAllObserverPathsIn *Retrieves the path of all observers defined in the container and all its children*

Description

Retrieves the path of all observers defined in the container and all its children

Usage

```
getAllObserverPathsIn(container)
```

Arguments

container	A Container or Simulation used to find the observers

Value

An array with one entry per observer defined in the container

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Returns the path of all quantities defined in the simulation
observerPaths <- getAllObserverPathsIn(sim)
```

getAllParameterPathsIn

Retrieves the path of all parameters defined in the container and all its children

Description

Retrieves the path of all parameters defined in the container and all its children

Usage

```
getAllParameterPathsIn(container)
```

Arguments

container	A Container or Simulation used to find the parameters

Value

An array with one entry per parameter defined in the container

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Returns the path of all quantities defined in the simulation
parameterPaths <- getAllParameterPathsIn(sim)
```

getAllParametersForSensitivityAnalysisMatching

Retrieve all parameters of the given simulation matching the given path criteria and also potential candidate for sensitivity variation

Description

Retrieve all parameters of the given simulation matching the given path criteria and also potential candidate for sensitivity variation

Usage

```
getAllParametersForSensitivityAnalysisMatching(paths, simulation)
```

Arguments

paths	A vector of strings representing the path of the parameters (potentially using wildcards)
simulation	Simulation used to find the parameters

Value

A list of parameters matching the path criteria and also candidates for a sensitivity analysis. The list is empty if no parameters matching were found.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Return all `Volume` parameters defined in all direct containers of the organism
params <- getAllParametersForSensitivityAnalysisMatching("Organism|*|Volume", sim)
```

---

getAllParametersMatching

Retrieve all parameters of a container (simulation or container instance) matching the given path criteria

---

Description

Retrieve all parameters of a container (simulation or container instance) matching the given path criteria

Usage

```
getAllParametersMatching(paths, container)
```

Arguments

paths	A vector of strings representing the paths relative to the container
container	A Container or Simulation used to find the parameters

Value

A list of parameters matching the path criteria. The list is empty if no parameters matching were found.

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Return all `Volume` parameters defined in all direct containers of the organism
params <- getAllParametersMatching("Organism|*|Volume", sim)

# Return all `Volume` parameters defined in all direct containers of the organism
# and the parameter 'Weight (tissue)' of the container 'Liver'
paths <- c("Organism|*|Volume", "Organism|Liver|Weight (tissue)")
params <- getAllParametersMatching(paths, sim)

# Returns all `Volume` parameters defined in `Organism` and all its subcontainers
params <- getAllParametersMatching("Organism|**|Volume", sim)
```

---

getAllQuantitiesMatching

Retrieve all quantities of a container (simulation or container instance) matching the given path criteria

---

Description

Retrieve all quantities of a container (simulation or container instance) matching the given path criteria

Usage

```
getAllQuantitiesMatching(paths, container)
```

Arguments

paths	A vector of strings relative to the container
container	A Container or Simulation used to find the parameters

Value

A list of quantities matching the path criteria. The list is empty if no quantity matching were found.

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Return all `Volume` quantities defined in all direct containers of the organism
quantities <- getAllQuantitiesMatching("Organism|*|Volume", sim)

# Return all `Volume` quantities defined in all direct containers of the organism
# and the parameter 'Weight (tissue)' of the container 'Liver'
paths <- c("Organism|*|Volume", "Organism|Liver|Weight (tissue)")
quantities <- getAllQuantitiesMatching(paths, sim)

# Returns all `Volume` quantities defined in `Organism` and all its subcontainers
quantities <- getAllQuantitiesMatching("Organism|**|Volume", sim)
```

---

getAllQuantityPathsIn *Retrieves the path of all quantities defined in the container and all its children*

Description

Retrieves the path of all quantities defined in the container and all its children

Usage

```
getAllQuantityPathsIn(container)
```

Arguments

container	A Container or Simulation used to find the parameters

Value

An array with one entry per quantity defined in the container

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Returns the path of all quantities defined in the simulation
quantityPaths <- getAllQuantityPathsIn(sim)
```

`getAllStateVariableParametersPaths`
Get the paths of all state variable parameters of the simulation

---

Description

Get the paths of all state variable parameters of the simulation

Usage

```
getAllStateVariableParametersPaths(simulation)
```

Arguments

`simulation`	`Simulation` object

Details

List of paths of all state variable parameters.

Value

A list of paths

`getAllStateVariablesPaths`
Get the paths of all state variable quantities of the simulation

---

Description

Get the paths of all state variable quantities of the simulation

Usage

```
getAllStateVariablesPaths(simulation)
```

Arguments

`simulation`	`Simulation` object

Details

List of paths of all molecules in all compartments and all parameters that are state variables.

Value

A list of paths

getBaseUnit *Get base unit of a dimension*

Description

Get base unit of a dimension

Usage

```
getBaseUnit(quantityOrDimension)
```

Arguments

quantityOrDimension	Instance of a quantity from which the dimension will be retrieved or name of dimension

Value

String name of the base unit.

getContainer *Retrieve a single container by path under the given container*

Description

Retrieve a single container by path under the given container

Usage

```
getContainer(path, container, stopIfNotFound = TRUE)
```

Arguments

path	A string representing the path relative to the container
container	A Container or Simulation used to find the containers
stopIfNotFound	Boolean. If TRUE (default) and no container exists for the given path, an error is thrown. If FALSE, NULL is returned.

Value

The Container with the given path. If the container for the path does not exist, an error is thrown if stopIfNotFound is TRUE (default), otherwise NULL

See Also

loadSimulation() and getContainer() to create objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
container <- getContainer("Organism|Liver", sim)
```

---

getDimensionByName *Get dimension by name*

Description

Get dimension by name

Usage

```
getDimensionByName(name)
```

Arguments

name	Name of dimension that should be retrieved

Value

Returns the an instance of the dimension with the given name if found or NULL otherwise.

Examples

```
getDimensionByName("Time")
```

---

getDimensionForUnit *Get dimension for a given unit*

Description

Get dimension for a given unit

Usage

```
getDimensionForUnit(unit)
```

Arguments

unit	Unit used to find the corresponding dimension.

Value

Returns the name of dimension that can be used to support the given unit or NULL if the dimension cannot be found.

Examples

```
getDimensionForUnit("mg")
```

---

getMolecule *Retrieve a single molecule by path in the given container*

Description

Retrieve a single molecule by path in the given container

Usage

```
getMolecule(path, container, stopIfNotFound = TRUE)
```

Arguments

path	A string representing the path relative to the container
container	A Container or Simulation used to find the molecules
stopIfNotFound	Boolean. If TRUE (default) and no molecule exist for the given path, an error is thrown. If FALSE, NULL is returned.

Value

The Molecule with the given path. If the molecule for the path does not exist, an error is thrown if stopIfNotFound is TRUE (default), otherwise NULL

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
molecule <- getMolecule("Organism|Liver|A", sim)
```

getMolWeightFor *Retrieve molecular weight for a quantity's molecule*

Description

Returns the molecular weight of the molecule for a given quantity. If no unit is provided, the value is returned in the base unit (kg/\u00b5mol).

Usage

```
getMolWeightFor(quantity, unit = NULL, stopIfNotFound = FALSE)
```

Arguments

quantity	A Quantity object.
unit	Optional. Target unit for the molecular weight. Defaults to kg/\u00b5mol.
stopIfNotFound	Logical. If TRUE, throws an error when the molecular weight cannot be retrieved. If FALSE, returns NA. Default is FALSE.

Value

The molecular weight in the specified unit or NA if not found.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
parameterPath <- "Organism|Lumen|Stomach|Dapagliflozin"
quantity <- getQuantity(parameterPath, container = sim)
getMolWeightFor(quantity, unit = "g/mol")
```

getOSPSuiteSetting *Get the value of a global ospsuite-R setting.*

Description

Get the value of a global ospsuite-R setting.

Usage

```
getOSPSuiteSetting(settingName)
```

Arguments

settingName	String name of the setting

Value

Value of the setting stored in ospsuiteEnv. If the setting does not exist, an error is thrown.

Examples

```
getOSPSuiteSetting("suiteVersion")
getOSPSuiteSetting("sensitivityAnalysisConfig")$totalSensitivityThreshold
```

---

getOutputValues *Extracting simulated values*

Description

The function receives an object of simulation results generated by running the simulation and returns time-values profiles for the chosen quantities. Results of a simulation of a single individual is treated as a population simulation with only one individual.

Usage

```
getOutputValues(
  simulationResults,
  quantitiesOrPaths = NULL,
  population = NULL,
  individualIds = NULL,
  stopIfNotFound = TRUE,
  addMetaData = TRUE
)
```

Arguments

simulationResults	Object of type `SimulationResults` produced by calling `runSimulations` on a `Simulation` object.
quantitiesOrPaths	Quantity instances (element or vector) typically retrieved using `getAllQuantitiesMatching` or quantity path (element or vector of strings) for which the results are to be returned. (optional) When providing the paths, only absolute full paths are supported (i.e., no matching with '*' possible). If quantitiesOrPaths is `NULL` (default value), returns the results for all output defined in the results.
population	population used to calculate the `simulationResults` (optional). This is used only to add the population covariates to the resulting data table.
individualIds	`numeric` IDs of individuals for which the results should be extracted. By default, all individuals from the results are considered. If the individual with the provided ID is not found, the ID is ignored.
stopIfNotFound	If `TRUE` (default) an error is thrown if no results exist for any `path`. If `FALSE`, a list of `NA` values is returned for the respective path.

addMetaData	If TRUE (default), the output is a list two sublists dataand metaData, with latter storing information about units and dimensions of the outputs. If FALSE, metaData is NULL. Setting this option to FALSE might improve the performance of the function.

Value

Returns the simulated values for the selected outputs (e.g molecules or parameters).

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Running an individual simulation
# results is an instance of `SimulationResults`
results <- runSimulations(sim)[[1]]

getOutputValues(results)
```

---

getParameter *Retrieve a single parameter by path in the given container*

Description

Retrieve a single parameter by path in the given container

Usage

```
getParameter(path, container, stopIfNotFound = TRUE)
```

Arguments

path	A string representing the path relative to the container
container	A Container or Simulation used to find the parameters
stopIfNotFound	Boolean. If TRUE (default) and no parameter exist for the given path, an error is thrown. If FALSE, NULL is returned.

Value

The Parameter with the given path. If the parameter for the path does not exist, an error is thrown if stopIfNotFound is TRUE (default), otherwise NULL

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
param <- getParameter("Organism|Liver|Volume", sim)
```

---

getParameterDisplayPaths *Retrieves the display path of the parameters defined by paths in the simulation*

Description

Retrieves the display path of the parameters defined by paths in the simulation

Usage

```
getParameterDisplayPaths(paths, simulation)
```

Arguments

`paths`	A single string or array of paths path relative to the `container`
`simulation`	A simulation used to find the entities

Value

a display path for each parameter in paths

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
displayPath <- getParameterDisplayPaths("Organism|Liver|Volume", sim)
```

---

getQuantity *Retrieve a single quantity by path in the given container*

Description

Retrieve a single quantity by path in the given container

Usage

```
getQuantity(path, container, stopIfNotFound = TRUE)
```

Arguments

path	A string representing the path relative to the container
container	A Container or Simulation used to find the parameters
stopIfNotFound	Boolean. If TRUE (default) and no quantity exists for the given path, an error is thrown. If FALSE, NULL is returned.

Value

The Quantity with the given path. If the quantity for the path does not exist, an error is thrown if stopIfNotFound is TRUE (default), otherwise NULL

See Also

loadSimulation(), getContainer() and getAllContainersMatching() to retrieve objects of type Container or Simulation

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
quantity <- getQuantity("Organism|Liver|Volume", sim)
```

---

getQuantityValuesByPath

Get the values of quantities in the simulation by path

Description

Get the values of quantities in the simulation by path

Usage

```
getQuantityValuesByPath(
  quantityPaths,
  simulation,
  units = NULL,
  stopIfNotFound = TRUE
)
```

Arguments

quantityPaths	A single or a list of absolute quantity paths
simulation	Simulation containing the quantities
units	A string or a list of strings defining the units of returned values. If NULL (default), values are returned in base units. If not NULL, must have the same length as quantityPaths. Single entries may be NULL.
stopIfNotFound	Boolean. If TRUE (default) and no quantity exists for the given path, an error is thrown. If FALSE, a warning is shown to the user.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
getQuantityValuesByPath(
  list("Organism|Liver|Volume", "Organism|Liver|A"),
  sim, list("ml", NULL)
)
```

---

getSimulationTree *Get simulation tree*

Description

Given a simulation file path or an instance of a simulation, traverses the simulation structure and returns a tree like structure allowing for intuitive navigation in the simulation tree.

Usage

```
getSimulationTree(simulationOrFilePath, quantityType = "Quantity")
```

Arguments

`simulationOrFilePath`	Full path of the simulation to load or instance of a simulation.
`quantityType`	A vector of strings that specify the types of the entities to be included in the tree. The types can be any combination of "Quantity", "Molecule", "Parameter" and "Observer".

Value

A list with a branched structure representing the path tree of entities in the simulation file that fall under the types specified in `quantityType`. At the end of each branch is a string called 'path' that is the path of the quantity represented by the branch.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

tree <- getSimulationTree(sim)

liver_volume_path <- tree$Organism$Liver$Volume$path
```

`getStandardMoleculeParameters`
Returns a list containing all standard global parameters defined in a `simulation` *for given* `moleculeName`*. These parameters are typically located directly under the container named after the* `moleculeName`*. For the list of standard parameters*

Description

Returns a list containing all standard global parameters defined in a `simulation` for given `moleculeName`. These parameters are typically located directly under the container named after the `moleculeName`. For the list of standard parameters

Usage

```
getStandardMoleculeParameters(moleculeName, simulation)
```

Arguments

`moleculeName`	Name of molecule (Enzyme, Transporter etc..) for which global parameters should be returned
`simulation`	Simulation to query for molecule parameters

Value

A list of all standard global parameters defined for `moleculeName` if the molecule exists in the `simulation`. Otherwise an empty list is returned

See Also

MoleculeParameter

Examples

```
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim1 <- loadSimulation(simPath)

parameters <- getStandardMoleculeParameters("CYP3A4", sim1)
```

`getSteadyState` *Get the steady-state values of species and state variable parameters.*

Description

Get the steady-state values of species and state variable parameters.

Usage

```
getSteadyState(
  simulations,
  quantitiesPaths = NULL,
  steadyStateTime = NULL,
  ignoreIfFormula = TRUE,
  lowerThreshold = 1e-15,
  simulationRunOptions = NULL
)
```

Arguments

`simulations`	`Simulation` object or a list or vector of `Simulation` objects to simulate. List or vector can be named (names must be uniques), in which case the names will reused in the output list. If not named, the output list will use simulation ids for names.
`quantitiesPaths`	List of quantity paths (molecules and/or parameters) for which the steady-state will be simulated. If `NULL` (default), all molecules and state variable parameters are considered. The same list is applied for all simulations.
`steadyStateTime`	Simulation time (minutes). In `NULL` (default), the default simulation time is the start time of the last application plus three days. The simulated time must be long enough for the system to reach a steady-state. Either a single value (will be applied for all simulations), or a list of values specific for each simulation. In latter case, must have equal size as `simulations`. When providing a list, `NULL` is allowed to calculate the time based on the last application.
`ignoreIfFormula`	If `TRUE` (default), species and parameters with initial values defined by a formula are not included.
`lowerThreshold`	Numerical value (in default unit of the output). Any steady-state values with absolute value below this threshold are considered as numerical noise and replaced by 0 (i.e., values in the interval `[-lowerThreshold, lowerThreshold]`). If `lowerThreshold` is `NULL`, no cut-off is applied. Default value is 1e-15.
`simulationRunOptions`	Optional instance of a `SimulationRunOptions` used during the simulation run.

Details

The steady-state is considered to be the last values of the molecules amounts and state variable parameters in the simulation with sufficiently long simulation time, i.e., where the rates of the processes do not (significantly) change. The steady-state is NOT analytically calculated or estimated in any other way than simulating for the given time.

Value

A named list, where the names are the IDs of the simulations and the entries are lists containing `paths` and their `values` at the end of the simulation.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
steadyState <- getSteadyState(simulations = sim)
# Set initial values for steady-state simulations
setQuantityValuesByPath(
  quantityPaths = steadyState[[sim$id]]$paths,
  values = steadyState[[sim$id]]$values, simulation = sim
)
```

---

getUnitsForDimension *Get units for a given dimension*

Description

Get units for a given dimension

Usage

```
getUnitsForDimension(dimension)
```

Arguments

`dimension`	Name of dimension for which units should be returned

Value

Returns a vector containing all units defined in the dimension

Examples

```
getUnitsForDimension("Mass")
```

hasDimension *Dimension existence*

Description

Dimension existence

Usage

```
hasDimension(dimension)
```

Arguments

dimension	String name of the dimension.

Details

Returns TRUE if the provided dimension is supported otherwise FALSE

hasUnit *Unit existence*

Description

Unit existence

Usage

```
hasUnit(unit, dimension)
```

Arguments

unit	String name of the unit
dimension	String name of the dimension.

Details

Check if the unit is valid for the dimension.

HumanPopulation *Default human population defined in PK-Sim*

Description

Default human population defined in PK-Sim

Usage

```
HumanPopulation
```

importPKAnalysesFromCSV *Loads the pK-analyses from csv file*

Description

Loads the pK-analyses from csv file

Usage

```
importPKAnalysesFromCSV(filePath, simulation)
```

Arguments

filePath	Full path of the file containing the pK-Analyses to load.
simulation	Instance of the simulation for which the pk-Analyses were calculated. This is required to verify that the file matches the simulation.

importResultsFromCSV *Imports the simulation results from one or more csv files*

Description

Imports the simulation results from one or more csv files

Usage

```
importResultsFromCSV(simulation, filePaths)
```

Arguments

simulation	Instance of a simulation used to calculate the results
filePaths	Full path of result files to import. Typically only one file is provided but a list of files is sometimes available when the simulation was parallelized and computed on different machines

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
resultPath <- system.file("extdata", "res.csv", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath)

# Run the simulation
results <- importResultsFromCSV(sim, resultPath)
```

---

importSensitivityAnalysisResultsFromCSV

Imports the simulation analysis results from one or more csv files

---

Description

Imports the simulation analysis results from one or more csv files

Usage

```
importSensitivityAnalysisResultsFromCSV(simulation, filePaths)
```

Arguments

simulation	Instance of a simulation for which the sensitivity analysis was performed
filePaths	Full path of sensitivity analysis result files to import. Typically only one file is provided but a list of files is sometimes available when the sensitivity analysis run was parallelized and computed on different machines

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
resultPath <- system.file("extdata", "sa.csv", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath)

# Run the simulation
results <- importSensitivityAnalysisResultsFromCSV(sim, resultPath)
```

IndividualCharacteristics *IndividualCharacteristics*

Description

Characteristics of an individual describing its origin

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `IndividualCharacteristics`

Active bindings

`species` Specifies the species of the individual. It should be a species available in PK-Sim (see `Species`)

`population` For a Human species, the population of interest. It should be a population available in PK-Sim (see `HumanPopulation`)

`gender` Gender of the individual. It should be defined for the species in PK-Sim (see `Gender`)

`age` Age of the individual as in instance of a `SnapshotParameter` (optional)

`gestationalAge` Gestational Age of the individual as in instance of a `SnapshotParameter` (optional)

`weight` Weight of the individual as in instance of a `SnapshotParameter` (optional)

`height` Height of the individual as in instance of a `SnapshotParameter` (optional)

`allMoleculeOntogenies` All molecule ontogenies defined for this individual characteristics.

`seed` Seed used to generate the population

Methods

Public methods:

- `IndividualCharacteristics$new()`
- `IndividualCharacteristics$print()`
- `IndividualCharacteristics$addMoleculeOntogeny()`

`IndividualCharacteristics$new()`**:** Initialize a new instance of the class

Usage:

```
IndividualCharacteristics$new()
```

Returns: A new `IndividualCharacteristics` object.

`IndividualCharacteristics$print()`**:** Print the object to the console

Usage:

```
IndividualCharacteristics$print(...)
```

Arguments:

... Rest arguments.

`IndividualCharacteristics$addMoleculeOntogeny()`: Add a molecule ontogeny `MoleculeOntogeny` to the individual characteristics

Usage:

```
IndividualCharacteristics$addMoleculeOntogeny(moleculeOntogeny)
```

Arguments:

`moleculeOntogeny` Molecule ontogeny to add

`initPKSim` *Loads the* `PKSim.R` *dll that will enable create individual and create population workflows.*

Description

Loads the `PKSim.R` dll that will enable create individual and create population workflows.

Usage

```
initPKSim()
```

Note

PKSim dlls are included in the package.

`isExplicitFormulaByPath` *Is the value defined by an explicit formula*

Description

Is the value defined by an explicit formula

Usage

```
isExplicitFormulaByPath(path, simulation, stopIfNotFound = TRUE)
```

Arguments

`path`	Path to the quantity
`simulation`	A `Simulation` object that contains the quantity
`stopIfNotFound`	Boolean. If `TRUE` (default) and no quantity exists for the given path, an error is thrown. If `FALSE`, `FALSE` is returned.

Value

TRUE if the value is an explicit formula, FALSE otherwise. Also returns FALSE if no quantity with the given path is found and stopInfNotFound is set to FALSE.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
isExplicitFormulaByPath("Organism|Liver|Volume", sim) # FALSE
```

---

isSupportedUnit *Check if unit is supported in the OSPSuite platform*

Description

Check if unit is supported in the OSPSuite platform

Usage

```
isSupportedUnit(unit)
```

Arguments

unit	String name of the unit

Details

Returns TRUE if the provided unit is supported in the OSPSuite platform, otherwise FALSE

Value

Logical: TRUE if the unit is supported, FALSE otherwise.

Examples

```
isSupportedUnit("kg")
isSupportedUnit("invalid_unit")
```

loadAgingDataFromCSV *Loads aging data (typically generated from PK-Sim) i*

Description

Loads aging data (typically generated from PK-Sim) i

Usage

```
loadAgingDataFromCSV(filePath)
```

Arguments

filePath	Full path containing an aging data table.

Examples

```
csvPath <- system.file("extdata", "aging_data.csv", package = "ospsuite")

agingData <- loadAgingDataFromCSV(csvPath)
```

loadDataImporterConfiguration *Load* DataImporterConfiguration *from XML file.*

Description

Load DataImporterConfiguration from XML file.

Usage

```
loadDataImporterConfiguration(configurationFilePath)
```

Arguments

configurationFilePath	Path to the XML file with stored configuration (e.g. created in PK-Sim or MoBi).

Value

A new DataImporterConfiguration object to be used with loadDataSetsFromExcel().

Examples

```
configurationFilePath <- system.file(
  "extdata", "dataImporterConfiguration.xml",
  package = "ospsuite"
)

importerConfiguration <- loadDataImporterConfiguration(configurationFilePath)

# Specifying which sheet to load
importerConfiguration$sheets <- "TestSheet_1"
xlsFilePath <- system.file("extdata", "CompiledDataSet.xlsx", package = "ospsuite")
dataSets <- loadDataSetsFromExcel(
  xlsFilePath = xlsFilePath,
  importerConfigurationOrPath = importerConfiguration,
  importAllSheets = FALSE
)
```

---

loadDataSetFromPKML *Loads data (typically observed data) from a PKML file and creates a* DataSet *from it. The pkml files are typically exported from PK-Sim or MoBi.*

Description

Loads data (typically observed data) from a PKML file and creates a `DataSet` from it. The pkml files are typically exported from PK-Sim or MoBi.

Usage

```
loadDataSetFromPKML(filePath)
```

Arguments

`filePath`	Full path of pkml file containing the observed data to load

Examples

```
filePath <- system.file("extdata", "ObsDataAciclovir_1.pkml", package = "ospsuite")

obsData <- loadDataSetFromPKML(filePath)
```

`loadDataSetsFromExcel` *Load data sets from excel*

Description

Load data sets from excel

Usage

```
loadDataSetsFromExcel(
  xlsFilePath,
  importerConfigurationOrPath,
  importAllSheets = FALSE,
  sheets = NULL
)
```

Arguments

`xlsFilePath`	Path to the excel file with the data
`importerConfigurationOrPath`	An object of type `DataImporterConfiguration` that is valid for the excel file or a path to a XML file with stored configuration
`importAllSheets`	**[Deprecated]** If `FALSE` (default), only sheets specified in the `importerConfiguration` or in the `sheets` parameter will be loaded. If `TRUE`, an attempt to load all sheets is performed. If any sheet does not comply with the configuration, an error is thrown. When set to `TRUE`, this parameter takes priority over the `sheets` parameter and configuration sheets. **Deprecated**: Use `sheets = NULL` instead. This parameter will be removed in version 14.
`sheets`	Character vector of sheet names to load, or `NULL` (default). An empty character vector (`character(0)`) is treated the same as `NULL`. If `NULL` and `importAllSheets` is `FALSE`, the sheets defined in the `importerConfiguration` will be used. If the configuration has no sheets defined and `sheets` is `NULL` and `importAllSheets` is `FALSE`, all sheets will be loaded. If a character vector is provided, only the specified sheets will be loaded, overriding any sheets defined in the `importerConfiguration` (unless `importAllSheets = TRUE`).

Details

Load observed data from an excel file using an importer configuration

Value

A named set of `DataSet` objects. The naming is defined by the property `importerConfiguration$namingPattern`.

Examples

```
xlsFilePath <- system.file(
  "extdata", "CompiledDataSet.xlsx",
  package = "ospsuite"
)

# When sheet is specified, it is automatically added to the configuration
importerConfiguration <- createImporterConfigurationForFile(
  xlsFilePath,
  sheet = "TestSheet_1"
)

dataSets <- loadDataSetsFromExcel(
  xlsFilePath = xlsFilePath,
  importerConfigurationOrPath = importerConfiguration
)

# Load specific sheets using the sheets parameter
dataSets <- loadDataSetsFromExcel(
  xlsFilePath = xlsFilePath,
  importerConfigurationOrPath = importerConfiguration,
  sheets = c("TestSheet_1", "TestSheet_2")
)

## Not run:
# Load all sheets by setting sheets to NULL and no sheets in configuration
importerConfiguration <- createImporterConfigurationForFile(xlsFilePath)
dataSets <- loadDataSetsFromExcel(
  xlsFilePath = xlsFilePath,
  importerConfigurationOrPath = importerConfiguration,
  sheets = NULL
)

## End(Not run)
```

---

loadPopulation *Loads a population from a csv file and returns the population.*

Description

Loads a population from a csv file and returns the population.

Usage

```
loadPopulation(csvPopulationFile)
```

Arguments

`csvPopulationFile`
: Full path of csv population file to load.

Examples

```
csvPath <- system.file("extdata", "pop.csv", package = "ospsuite")

population <- loadPopulation(csvPath)
```

`loadProjectFromSnapshot` *Load a project from a snapshot*

Description

Converts one or more snapshot files (`.json`) into project files and writes them to an output directory.

Only PK-Sim projects (`.pksim5`) are supported for now. Support for MoBi projects is planned.

Usage

```
loadProjectFromSnapshot(..., output = ".", runSimulations = FALSE)
```

Arguments

`...`	character strings, path to snapshot files (`.json`) or a directory containing snapshot files to convert.
`output`	character string, path to the output directory where to write the converted project files.
`runSimulations`	logical, whether to run the simulations during conversion (default = `FALSE`).

Examples

```
## Not run:
loadProjectFromSnapshot("path/to/snapshot.json", output = "path/to/output")

## End(Not run)
```

`loadSimulation` *Load a simulation from a pkml file*

Description

Loads a simulation from a pkml file and returns the simulation. If the passed simulation file has been loaded before, the simulation is not loaded again but a cached object is returned. This behavior can be overridden.

Usage

```
loadSimulation(
  filePath,
  loadFromCache = FALSE,
  addToCache = TRUE,
  resetIds = TRUE
)
```

Arguments

filePath	Full path of pkml simulation file to load.
loadFromCache	If TRUE, an already loaded pkml file will not be loaded again, but the simulation object will be retrieved from cache. If FALSE, a new simulation object will be created. Default value is FALSE.
addToCache	If TRUE, the loaded simulation is added to cache. If FALSE, the returned simulation only exists locally. Default is TRUE.
resetIds	If TRUE, the internal object ids in the simulation are reset to a unique value. If FALSE, the ids are kept as defined in the pkml simulation. Default is TRUE.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load sim1 for the first time
sim1 <- loadSimulation(simPath)

# sim2 will be loaded from cache and will represent the same object as sim1
sim2 <- loadSimulation(simPath, loadFromCache = TRUE)

parameter1 <- getParameter(toPathString(c("Organism", "Liver", "Volume")), sim1)
parameter2 <- getParameter(toPathString(c("Organism", "Liver", "Volume")), sim2)

# parameter1 and parameter2 belong to the same simulation object, so changing
# one of the them will also change another
setParameterValues(parameters = parameter2, values = 0)
parameter1$value == parameter2$value # TRUE

# sim3 will not be loaded from cache
sim3 <- loadSimulation(simPath, loadFromCache = FALSE)
# parameter3 belong to different simulation object than parameter1 and parameter2
parameter3 <- getParameter(toPathString(c("Organism", "Liver", "Volume")), sim3)
setParameterValues(parameters = parameter3, values = 1)
parameter2$value == parameter3$value # FALSE#'
```

messages — *List of functions and strings used to signal error messages Extends the* `messages` *list from ospsuite.utils*

Description

List of functions and strings used to signal error messages Extends the `messages` list from ospsuite.utils

Usage

```
messages
```

MoleculeOntogeny — *MoleculeOntogeny*

Description

Use when retrieving individual values using the createIndividualAlgorithm. This class is a simple pair (MoleculeName, Ontogeny) allowing the user to retrieve potential ontogeny values.

Public fields

`molecule` Name of the molecule in the model

`ontogeny` Name of the ontogeny to use for the molecule

Methods

Public methods:

- `MoleculeOntogeny$new()`
- `MoleculeOntogeny$print()`
- `MoleculeOntogeny$printMoleculeOntogeny()`

`MoleculeOntogeny$new()`: Initialize a new instance of the class

Usage:

```
MoleculeOntogeny$new(molecule, ontogeny)
```

Arguments:

`molecule` molecule name

`ontogeny` ontogeny to use for the Molecule (one of StandardOntogeny)

Returns: A new `MoleculeOntogeny` object.

`MoleculeOntogeny$print()`: Print the object to the console

Usage:

`MoleculeOntogeny$print(...)`

Arguments:

`...` Rest arguments.

`MoleculeOntogeny$printMoleculeOntogeny()`: Print the `MoleculeOntogeny` on one line

Usage:

```
MoleculeOntogeny$printMoleculeOntogeny()
```

`MoleculeParameter` *Standard molecule parameter names typically available in an endogenous molecule (enzyme, transporter etc...) coming from PK-Sim*

Description

Standard molecule parameter names typically available in an endogenous molecule (enzyme, transporter etc...) coming from PK-Sim

Usage

```
MoleculeParameter
```

`ospDimensions` *Supported dimensions defined as a named list*

Description

Supported dimensions defined as a named list

Usage

```
ospDimensions
```

Details

ospDimensions$Mass => "Mass"

ospsuite_deprecated *Deprecated functions*

Description

Deprecated functions

Usage

```
pkAnalysesAsDataFrame(...)

populationAsDataFrame(...)
```

Arguments

...	Arguments to the deprecated function.

Details

- pkAnalysesAsDataFrame is now pkAnalysesToDataFrame.
- populationAsDataFrame is now populationToDataFrame.

ospsuiteSettingNames *Names of the settings stored in ospsuiteEnv. Can be used with* getOSPSuiteSetting()

Description

Names of the settings stored in ospsuiteEnv. Can be used with getOSPSuiteSetting()

Usage

```
ospsuiteSettingNames
```

ospUnits *Supported units defined as a named list of lists*

Description

ospUnits$Mass$kg => "kg"

Usage

```
ospUnits
```

OutputSchema *OutputSchema*

Description

Output schema associated with a given simulation

Super classes

rSharp::NetObject -> DotNetWrapper -> OutputSchema

Active bindings

`intervals` All intervals defined in the schema (Read-Only)

`timePoints` All single time points defined in the schema (Read-Only)

`endTime` Returns the end time of the simulation in kernel unit (Read-Only)

Methods

Public methods:

- `OutputSchema$clear()`
- `OutputSchema$addInterval()`
- `OutputSchema$removeInterval()`
- `OutputSchema$addTimePoints()`
- `OutputSchema$print()`

`OutputSchema$clear()`: Clears all intervals and time points

Usage:

```
OutputSchema$clear()
```

`OutputSchema$addInterval()`: Adds an interval to the schema

Usage:

```
OutputSchema$addInterval(interval)
```

Arguments:

`interval` Interval to add

`OutputSchema$removeInterval()`: Removes the interval from the schema

Usage:

```
OutputSchema$removeInterval(interval)
```

Arguments:

`interval` Interval to remove

`OutputSchema$addTimePoints()`: Adds the time points to the schema. Note that time points and intervals exists concurrently. Use time points only if you need to ensure that specific time are used.

Usage:

```
OutputSchema$addTimePoints(timePoints)
```

Arguments:

`timePoints` Time points to add to the schema

`OutputSchema$print()`: Print the object to the console

Usage:

```
OutputSchema$print(...)
```

Arguments:

`...` Rest arguments.

`OutputSelections` *OutputSelections*

Description

List of selected quantities selected as output for a given simulation

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `OutputSelections`

Active bindings

`allOutputs` Returns all outputs defined in the selection

Methods

Public methods:

- `OutputSelections$clear()`
- `OutputSelections$addQuantity()`
- `OutputSelections$removeQuantity()`
- `OutputSelections$print()`

`OutputSelections$clear()`: Removes all selected output from the selection

Usage:

```
OutputSelections$clear()
```

`OutputSelections$addQuantity()`: Adds a quantity as selected

Usage:

```
OutputSelections$addQuantity(quantity)
```

Arguments:

`quantity` Quantity to add to the selection

`OutputSelections$removeQuantity()`: Removes a quantity from the selection

Usage:

```
OutputSelections$removeQuantity(quantity)
```

Arguments:

`quantity` Quantity to remove from the selection

`OutputSelections$print()`: Print the object to the console

Usage:

```
OutputSelections$print(...)
```

Arguments:

`...` Rest arguments.

`Parameter` *Parameter*

Description

A model parameter

Details

Derived from Quantity, please see base class documentation.

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `ObjectBase` -> `Entity` -> `Quantity` -> `Parameter`

Active bindings

`isStateVariable` Returns `TRUE` is the parameter has a RHS otherwise `FALSE`. Setting the value to `FALSE` will delete the RHS Formula. Setting it to `TRUE` is not currently supported and will throw an error.

`rhsFormula` An instance of a `Formula` object representing the RHS Formula (Read-Only)

Methods

Public methods:

- `Parameter$new()`
- `Parameter$print()`

`Parameter$new()`: Initialize a new instance of the class

Usage:

```
Parameter$new(netObject)
```

Arguments:

`netObject` An `rSharp::NetObject` object.

Returns: A new `Parameter` object.

`Parameter$print()`**:** Print the object to the console

Usage:

```
Parameter$print(...)
```

Arguments:

`...` Rest arguments.

`ParameterRange` *ParameterRange*

Description

A parameter range typically used in the definition of `PopulationCharacteristics` covariates (Height, Weight etc...)

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `ParameterRange`

Active bindings

`min` Minimum value for the parameter range

`max` Maximum value for the parameter range

`unit` Unit in which the value is defined

Methods

Public methods:

- `ParameterRange$new()`
- `ParameterRange$print()`
- `ParameterRange$printValue()`
- `ParameterRange$getPrintValue()`

`ParameterRange$new()`**:** Initialize a new instance of the class

Usage:

```
ParameterRange$new(netObject = NULL, min = NULL, max = NULL, unit = NULL)
```

Arguments:

`netObject` Optional `NetObject` of `ParameterRange`. If not defined, a new instance will be created

`min` Optional minimum value for the range

`max` Optional minimum value for the range

`unit` Optional unit of the specified min and max

Returns: A new `ParameterRange` object.

`ParameterRange$print()`**:** Print the object to the console

Usage:

```
ParameterRange$print(...)
```

Arguments:

`...` Rest arguments.

`ParameterRange$printValue()`**:** Print the parameter in one line

Usage:

```
ParameterRange$printValue(caption)
```

Arguments:

`caption` Caption to display before the value of the parameter

`ParameterRange$getPrintValue()`**:** Return a string for printing the parameter in one line

Usage:

```
ParameterRange$getPrintValue()
```

Returns: A string for printing the parameter in one line

`pkAnalysesToDataFrame` *Convert the pk-Analysis to data frame*

Description

Convert the pk-Analysis to data frame

Usage

```
pkAnalysesToDataFrame(pkAnalyses)

pkAnalysesToTibble(pkAnalyses)
```

Arguments

`pkAnalyses`	pK-Analyses to convert to data frame (typically calculated using `calculatePKAnalyses` or imported from file).

PKParameter *PKParameter*

Description

Standard PK Parameters defined in the OSPSuite

Super classes

rSharp::NetObject -> DotNetWrapper -> PKParameter

Active bindings

name Name of the PK-Parameter

displayName Display Name of the PK-Parameter. If not set, Name will be used

dimension Dimension instance used by the PK-Parameter (Read-Only)

unit Unit of the PK-Parameter (Read-Only)

displayUnit Display Unit used for the PK-Parameter

Methods

Public methods:

- PKParameter$print()

PKParameter$print(): Print the object to the console

Usage:

```
PKParameter$print(...)
```

Arguments:

... Rest arguments.

---

pkParameterByName *Returns an instance of a PK-Parameter by name or NULL if the parameter by name is not found*

Description

Returns an instance of a PK-Parameter by name or NULL if the parameter by name is not found

Usage

```
pkParameterByName(name, stopIfNotFound = TRUE)
```

Arguments

name	Name of PK-Parameter to update
stopIfNotFound	Boolean. If TRUE (default) and no pk parameter exist for the given name, an error is thrown. If FALSE, NULL is returned.

Examples

```
pkParameter <- pkParameterByName(name = "t_max")
```

PKParameterSensitivity

PKParameterSensitivity

Description

Sensitivity of a PK Parameter for one output for a given parameter

Super classes

rSharp::NetObject -> DotNetWrapper -> PKParameterSensitivity

Active bindings

parameterName Unique name of parameter in sensitivity analysis

pkParameterName Name of PK Output (Cmax, Tmax etc...)

outputPath Path of underlying quantity for which pk-analyses were performed

value Value of sensitivity

Methods

Public methods:

- PKParameterSensitivity$print()

PKParameterSensitivity$print(): Print the object to the console

Usage:

```
PKParameterSensitivity$print(...)
```

Arguments:

... Rest arguments.

plotIndividualTimeProfile

Time-profile plot of individual data

Description

[Deprecated]

Usage

```
plotIndividualTimeProfile(
  dataCombined,
  defaultPlotConfiguration = NULL,
  showLegendPerDataset = FALSE
)
```

Arguments

dataCombined	A single instance of DataCombined class containing both observed and simulated datasets to be compared.
defaultPlotConfiguration	A DefaultPlotConfiguration object, which is an R6 class object that defines plot properties.
showLegendPerDataset	Logical flag to display separate legend entries for observed and simulated datasets, if available. This is experimental and may not work reliably when both observed and simulated datasets > 1. Defaults to FALSE.

See Also

Other plotting: DefaultPlotConfiguration, plotObservedVsSimulated(), plotPopulationTimeProfile(), plotResidualsVsSimulated(), plotResidualsVsTime()

Examples

```
# Create a new instance of `DefaultPlotConfiguration` class
myPlotConfiguration <- DefaultPlotConfiguration$new()
myPlotConfiguration$title <- "My Plot Title"
myPlotConfiguration$subtitle <- "My Plot Subtitle"
myPlotConfiguration$caption <- "My Sources"

# plot
plotIndividualTimeProfile(dataCombinedAciclovir, myPlotConfiguration)
```

`plotObservedVsSimulated` *Observed versus predicted/simulated scatter plot*

Description

[Deprecated]

Usage

```
plotObservedVsSimulated(
  dataCombined,
  defaultPlotConfiguration = NULL,
  foldDistance = NULL
)
```

Arguments

`dataCombined`	A single instance of `DataCombined` class containing both observed and simulated datasets to be compared.
`defaultPlotConfiguration`	A `DefaultPlotConfiguration` object, which is an `R6` class object that defines plot properties.
`foldDistance`	A vector for plotting lines at required fold distances around the identity line (`x=y`). Set to NULL (default) to only draw identity line. Set to FALSE to not draw any lines. The vector can include only fold distance values `>1`. An `x`-fold distance is defined as all simulated values within the range between `x`-fold (depicted by the upper fold range line) and `1/x`-fold (depicted by the lower fold range line) of observed values. The identity line can be interpreted as the `1`-fold range.

See Also

Other plotting: `DefaultPlotConfiguration`, `plotIndividualTimeProfile()`, `plotPopulationTimeProfile()`, `plotResidualsVsSimulated()`, `plotResidualsVsTime()`

Examples

```
# Create a new instance of `DefaultPlotConfiguration` class
myPlotConfiguration <- DefaultPlotConfiguration$new()
myPlotConfiguration$title <- "My Plot Title"
myPlotConfiguration$subtitle <- "My Plot Subtitle"
myPlotConfiguration$caption <- "My Sources"

# plot
plotObservedVsSimulated(dataCombinedAciclovir, myPlotConfiguration)
```

plotPopulationTimeProfile

Time-values profile plot for population simulations

Description

[Deprecated]

Usage

```
plotPopulationTimeProfile(
  dataCombined,
  defaultPlotConfiguration = NULL,
  aggregation = "quantiles",
  quantiles = c(0.05, 0.5, 0.95),
  showLegendPerDataset = FALSE,
  ...
)
```

Arguments

dataCombined	A single instance of DataCombined class containing both observed and simulated datasets to be compared.
defaultPlotConfiguration	A DefaultPlotConfiguration object, which is an R6 class object that defines plot properties.
aggregation	The type of the aggregation of individual data. One of quantiles (Default), arithmetic or geometric (full list in ospsuite::DataAggregationMethods). Will replace yValues by the median, arithmetic or geometric average and add a set of upper and lower bounds (yValuesLower and yValuesHigher).
quantiles	A numerical vector with quantile values (Default: c(0.05, 0.50, 0.95)) to be plotted. Ignored if aggregation is not quantiles.
showLegendPerDataset	Logical flag to display separate legend entries for observed and simulated datasets, if available. This is experimental and may not work reliably when both observed and simulated datasets > 1. Defaults to FALSE.
...	additionnal arguments to pass to .extractAggregatedSimulatedData()

Details

The simulated values will be aggregated across individuals for each time point.

For aggregation = quantiles (default), the quantile values defined in the argument quantiles will be used. In the profile plot, the middle value will be used to draw a line, while the lower and upper values will be used as the lower und upper ranges. For aggregation = arithmetic, arithmetic mean with arithmetic standard deviation (SD) will be plotted. Use the optional parameter

nsd to change the number of SD to plot above and below the mean. For aggregation = geometric, geometric mean with geometric standard deviation (SD) will be plotted. Use the optional parameter nsd to change the number of SD to plot above and below the mean.

See Also

Other plotting: DefaultPlotConfiguration, plotIndividualTimeProfile(), plotObservedVsSimulated(), plotResidualsVsSimulated(), plotResidualsVsTime()

Examples

```
simFilePath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
sim <- loadSimulation(simFilePath)

populationResults <- importResultsFromCSV(
  simulation = sim,
  filePaths = system.file("extdata", "SimResults.csv", package = "ospsuite")
)

# Create a new instance of `DataCombined` class
myDataComb <- DataCombined$new()
myDataComb$addSimulationResults(populationResults)


# plot
plotPopulationTimeProfile(myDataComb)

# plot with other quantiles
plotPopulationTimeProfile(myDataComb, quantiles = c(0.1, 0.5, 0.9))

# plot with arithmetic mean
plotPopulationTimeProfile(myDataComb,
  aggregation = "arithmetic"
)
```

---

plotPredictedVsObserved

Plot Predicted vs Observed Values

Description

Plots predicted vs observed data, grouped by "group".

This function visualizes the relationship between predicted and observed values, allowing for easy identification of discrepancies.

Usage

```
plotPredictedVsObserved(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  yUnit = NULL,
  xyScale = "log",
  predictedAxis = "y",
  comparisonLineVector = ospsuite.plots::getFoldDistanceList(folds = c(2)),
  showLegendPerDataset = "all",
  ...
)
```

Arguments

plotData	An object of class `DataCombined` or a `data.table`. If a `data.table`, it must include the following: • `xValues`: Numeric time points. • `yValues`: Observed or simulated values (numeric). • `group`: Grouping variable (factor or character). • `name`: Name for the dataset (factor or character). • `xUnit`: Unit of the x-axis values (character). • `yUnit`: Unit of the y-axis values (character). • `dataType`: Specifies data type—either `observed` or `simulated`. • Optional: – `yErrorType`: Type of y error, relative to `ospsuite::DataErrorType`. – `yErrorValues`: Numeric error values. – `yMin`, `yMax`: Custom ranges for y-axis instead of error types. – `IndividualId`: Used for aggregation of simulated population data.
metaData	A list containing metadata for the plot. If NULL, a default list is constructed from the data. Expected structure includes information about dimensions and units for both x and y axes.
mapping	A ggplot2 aesthetic mapping object. Default is `ggplot2::aes()`. This is added or replaces the default mapping constructed by the data.
yUnit	A character string specifying the target unit for the x and y-axis. If `NULL` (default), the most frequent unit in the data is used. For available units, see `ospsuite::ospUnits`.
xyScale	A character string specifying the scale for the x and y-axis. Default is `log`.
predictedAxis	A character string specifying which axis to use for predicted values. Options are `"x"` (predicted on x-axis, observed on y-axis) or `"y"` (default, predicted on y-axis, observed on x-axis).
comparisonLineVector	A named list generated by the function `ospsuite.plots::getFoldDistanceList`. This list contains fold distances, where each entry represents a fold and its reciprocal. The identity fold (1) will be included if specified in `getFoldDistanceList`.

showLegendPerDataset

Controls display of separate legend entries for individual datasets. One of:

- "none" : No per-dataset differentiation. Only group-level legend.
- "all" (default) or "observed": Differentiate observed datasets via different shapes (using the name column).

User-provided mapping will override internal settings.

... Arguments passed on to ospsuite.plots::plotYVsX

geomPointAttributes A list with arguments which are passed on to the call ggplot2::geom_point

geomErrorbarAttributes A list with arguments which are passed on to the call geom_errorbar_osp

geomGuestLineAttributes A list of arguments passed to ggplot2::geom_function to display guest criteria.

geomComparisonLineAttributes A list of arguments passed to ggplot2::hline or ggplot2::abline to display comparison lines.

geomLLOQAttributes A list with arguments which are passed on to the call ggplot2::geom_hline

groupAesthetics A character vector of aesthetic names used for grouping data points when calculating comparison statistics. Data will be grouped by combinations of these aesthetics before computing counts and proportions within comparison lines. Common grouping aesthetics include "colour", "fill", "shape".

addRegression A boolean that activates the insertion of a regression line.

addGuestLimits A boolean that activates the insertion of guest limits.

deltaGuest Numeric value parameter for the Guest function.

labelGuestCriteria Label used in the legend for guest criteria (default: "guest criteria").

xScaleArgs list of arguments passed to ggplot2::scale_x_continuous() or ggplot2::scale_x_log10()

yScaleArgs list of arguments passed to ggplot2::scale_y_continuous() or ggplot2::scale_y_log10()

lloqOnBothAxes A boolean; if TRUE, LLOQ lines are drawn on both axes. If FALSE (default), the LLOQ line is drawn for the observed-data axis only. (geom_vline when observedDataDirection = "x", geom_hline when observedDataDirection = "y").

Value

A ggplot2 plot object representing predicted vs observed values, including aesthetics for the x and y axes, or NULL if the data contains no plottable entries.

See Also

Other plot functions based on ospsuite.plots: plotQuantileQuantilePlot(), plotResidualsAsHistogram(), plotResidualsVsCovariate(), plotTimeProfile()

Examples

```
# Generate a predicted vs observed plot
plotPredictedVsObserved(dataCombinedAciclovir)

# Generate an observed vs predicted plot (swap axes)
plotPredictedVsObserved(dataCombinedAciclovir, predictedAxis = "x")

# Show individual dataset names in legend
plotPredictedVsObserved(dataCombinedAciclovir, showLegendPerDataset = "observed")
```

---

plotQuantileQuantilePlot

Plot Quantile-Quantile Plot

Description

Plots a Quantile-Quantile plot, grouped by "group".

This function visualizes the distribution of predicted vs observed values using a Q-Q plot.

Usage

```
plotQuantileQuantilePlot(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  xUnit = NULL,
  yUnit = NULL,
  residualScale = "log",
  ...
)
```

Arguments

plotData	An object of class `DataCombined` or a `data.table`. If a `data.table`, it must include the following: • `xValues`: Numeric time points. • `yValues`: Observed or simulated values (numeric). • `group`: Grouping variable (factor or character). • `name`: Name for the dataset (factor or character). • `xUnit`: Unit of the x-axis values (character). • `yUnit`: Unit of the y-axis values (character). • `dataType`: Specifies data type—either `observed` or `simulated`. • Optional: – `yErrorType`: Type of y error, relative to `ospsuite::DataErrorType`.

- `yErrorValues`: Numeric error values.
- `yMin`, `yMax`: Custom ranges for y-axis instead of error types.
- `IndividualId`: Used for aggregation of simulated population data.

`metaData` A list containing metadata for the plot. If NULL, a default list is constructed from the data. Expected structure includes information about dimensions and units for both x and y axes.

`mapping` A ggplot2 aesthetic mapping object. Default is `ggplot2::aes()`. This is added or replaces the default mapping constructed by the data.

`xUnit` A character string specifying the target unit for the x-axis. If `NULL` (default), the most frequent unit in the data is used. For available units, see `ospsuite::ospUnits`.

`yUnit` A character string specifying the target unit for the simulated and observed y-values used for residual calculation. If `NULL` (default), the most frequent unit in the data is used. For available units, see `ospsuite::ospUnits`.

`residualScale` Either "linear", "log", or "ratio" method for computing residuals. Default is `log`.

`...` Arguments passed on to `ospsuite.plots::plotQQ`

`xScaleArgs` list of arguments passed to `ggplot2::scale_x_continuous()` or `ggplot2::scale_x_log10()`

`yScaleArgs` list of arguments passed to `ggplot2::scale_y_continuous()` or `ggplot2::scale_y_log10()`

`geomQQAttributes` A list of arguments passed to `ggplot2::stat_qq()`.

`geomQQLineAttributes` A list of arguments passed to `ggplot2::stat_qq_line()`.

`groupAesthetics` A character vector of aesthetic names used for grouping data points in the Q-Q plot. Common options include `"colour"`, `"fill"`, `"shape"`, `"linetype"`, and `"size"`.

Value

A `ggplot2` plot object representing the Q-Q plot, or `NULL` if the data contains no plottable entries.

See Also

Other plot functions based on ospsuite.plots: `plotPredictedVsObserved()`, `plotResidualsAsHistogram()`, `plotResidualsVsCovariate()`, `plotTimeProfile()`

Examples

```
# Generate a Q-Q plot with default settings
plotQuantileQuantilePlot(dataCombinedAciclovir)

# Generate a Q-Q plot with linear scale
plotQuantileQuantilePlot(dataCombinedAciclovir, residualScale = "linear")
```

`plotResidualsAsHistogram` *Plot Residuals Histogram*

Description

Plots residuals as a histogram, grouped by "group".

This function generates a histogram of the residuals, providing a visual representation of their distribution.

Usage

```
plotResidualsAsHistogram(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  yUnit = NULL,
  distribution = "normal",
  residualScale = "log",
  ...
)
```

Arguments

`plotData`	An object of class `DataCombined` or a `data.table`. If a `data.table`, it must include the following: • `xValues`: Numeric time points. • `yValues`: Observed or simulated values (numeric). • `group`: Grouping variable (factor or character). • `name`: Name for the dataset (factor or character). • `xUnit`: Unit of the x-axis values (character). • `yUnit`: Unit of the y-axis values (character). • `dataType`: Specifies data type—either `observed` or `simulated`. • Optional: – `yErrorType`: Type of y error, relative to `ospsuite::DataErrorType`. – `yErrorValues`: Numeric error values. – `yMin`, `yMax`: Custom ranges for y-axis instead of error types. – `IndividualId`: Used for aggregation of simulated population data.
`metaData`	A list containing metadata for the plot. If NULL, a default list is constructed from the data. Expected structure includes information about dimensions and units for both x and y axes.
`mapping`	A ggplot2 aesthetic mapping object. Default is `ggplot2::aes()`. This is added or replaces the default mapping constructed by the data.

yUnit	A character string specifying the target unit for the simulated and observed y-values used for residual calculation. If NULL (default), the most frequent unit in the data is used. For available units, see ospsuite::ospUnits.
distribution	parameter passed to ospsuite.plots::plotHistogram.
residualScale	Either "linear", "log", or "ratio" method for computing residuals. Default is log.
...	Arguments passed on to ospsuite.plots::plotHistogram asBarPlot A logical indicating if geom_histogram should be used (for continuous data) or geom_bar (for categorical data). If TRUE, the variables distribution, meanFunction, xScale, and xScaleArgs are ignored. geomHistAttributes A list of arguments passed to ggplot2::geom_histogram (or geom_bar if asBarPlot = TRUE). plotAsFrequency A logical indicating if the histogram displays frequency on the y-axis. xScale either 'linear' then ggplot2::scale_x_continuous() or 'log' then ggplot2::scale_x_log10() is used xScaleArgs list of arguments passed to ggplot2::scale_x_continuous() or ggplot2::scale_x_log10() yScale either 'linear' then ggplot2::scale_y_continuous() or 'log' then ggplot2::scale_y_log10() is used yScaleArgs list of arguments passed to ggplot2::scale_y_continuous() or ggplot2::scale_y_log10() meanFunction Function selection for the display of a vertical line. Options: 'none', 'mean', 'geomean', 'median', 'auto' (default). 'auto' selects 'mean' for normal distribution, 'geomean' for lognormal, 'median' for other distributions, and 'none' when no distribution fit.

Value

A ggplot2 plot object representing the histogram of residuals, or NULL if the data contains no plottable entries.

See Also

Other plot functions based on ospsuite.plots: plotPredictedVsObserved(), plotQuantileQuantilePlot(), plotResidualsVsCovariate(), plotTimeProfile()

Examples

```
# Generate a histogram of residuals with default settings
plotResidualsAsHistogram(dataCombinedAciclovir)

# Generate a histogram with linear scale
plotResidualsAsHistogram(dataCombinedAciclovir, residualScale = "linear")
```

plotResidualsVsCovariate
Plot Residuals vs Covariate

Description

Plots residuals vs a covariate (time, observed, or predicted values), grouped by "group".

This function visualizes the residuals against time, observed, or predicted (simulated) values, helping to assess model performance.

Usage

```
plotResidualsVsCovariate(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  xUnit = NULL,
  yUnit = NULL,
  residualScale = "log",
  xAxis = "observed",
  showLegendPerDataset = "all",
  ...
)
```

Arguments

plotData	An object of class `DataCombined` or a `data.table`. If a `data.table`, it must include the following: • `xValues`: Numeric time points. • `yValues`: Observed or simulated values (numeric). • `group`: Grouping variable (factor or character). • `name`: Name for the dataset (factor or character). • `xUnit`: Unit of the x-axis values (character). • `yUnit`: Unit of the y-axis values (character). • `dataType`: Specifies data type—either `observed` or `simulated`. • Optional: – `yErrorType`: Type of y error, relative to `ospsuite::DataErrorType`. – `yErrorValues`: Numeric error values. – `yMin`, `yMax`: Custom ranges for y-axis instead of error types. – `IndividualId`: Used for aggregation of simulated population data.
metaData	A list containing metadata for the plot. If NULL, a default list is constructed from the data. Expected structure includes information about dimensions and units for both x and y axes.

`mapping` A ggplot2 aesthetic mapping object. Default is `ggplot2::aes()`. This is added or replaces the default mapping constructed by the data.

`xUnit` A character string specifying the target unit for the time values. (only relevant if `xAxis = "time"`) If `NULL` (default), the most frequent unit in the data is used. For available units, see `ospsuite::ospUnits`.

`yUnit` A character string specifying the target unit for the simulated and observed y-values used for residual calculation and (if `xAxis != "time"`) displayed on the x-Axis. If `NULL` (default), the most frequent unit in the data is used. For available units, see `ospsuite::ospUnits`.

`residualScale` Either "linear", "log", or "ratio" method for computing residuals. Default is `log`.

`xAxis` A character string specifying what to display on the x-axis. Options are `"time"` (time points from xValues), `"observed"` (observed values, default), or `"predicted"` (predicted/simulated values).

`showLegendPerDataset`

Controls display of separate legend entries for individual datasets. One of:

- `"none"`: No per-dataset differentiation. Only group-level legend.
- `"all"` (default) or `"observed"`: Differentiate observed datasets via different shapes (using the `name` column).

User-provided `mapping` will override internal settings.

`...` Arguments passed on to ospsuite.plots::plotYVsX, ospsuite.plots::plotResVsCov

`geomPointAttributes` A `list` with arguments which are passed on to the call `ggplot2::geom_point`

`geomErrorbarAttributes` A `list` with arguments which are passed on to the call `geom_errorbar_osp`

`geomComparisonLineAttributes` A `list` of arguments passed to `ggplot2::hline` or `ggplot2::abline` to display comparison lines.

`geomLLOQAttributes` A `list` with arguments which are passed on to the call `ggplot2::geom_hline`

`groupAesthetics` A character vector of aesthetic names used for grouping data points when calculating comparison statistics. Data will be grouped by combinations of these aesthetics before computing counts and proportions within comparison lines. Common grouping aesthetics include `"colour"`, `"fill"`, `"shape"`.

`addRegression` A boolean that activates the insertion of a regression line.

`xScale` either 'linear' then `ggplot2::scale_x_continuous()` or 'log' then `ggplot2::scale_x_log10()` is used

`xScaleArgs` list of arguments passed to `ggplot2::scale_x_continuous()` or `ggplot2::scale_x_log10()`

`yScale` either 'linear' then `ggplot2::scale_y_continuous()` or 'log' then `ggplot2::scale_y_log10()` is used

`yScaleArgs` list of arguments passed to `ggplot2::scale_y_continuous()` or `ggplot2::scale_y_log10()`

`comparisonLineVector` A vector defining the comparison lines.

Details

Residual Calculation:

Residuals are calculated by pairing observed and simulated data at matching time points within the same group. Only datasets that can be paired (i.e., have corresponding observed and simulated values) are included in the residual plot. The function automatically removes unpaired datasets with a warning, converts units to ensure consistent comparisons, and computes residuals as the difference between observed and predicted values.

Residual Scales:

The `residualScale` parameter controls how residuals are displayed:

- `linear`: Absolute residuals (Observed - Predicted). Values centered around zero indicate good model fit. Useful for normally distributed errors.
- `log`: Log-transformed residuals, calculated as log(Observed / Predicted). Values centered around zero indicate good fit. Preferred for log-normally distributed data or when errors are proportional to magnitude.
- `ratio`: Ratio of observed to predicted (Observed / Predicted). Values centered around 1.0 indicate good fit. Useful for understanding relative prediction error.

Value

A `ggplot2` plot object representing residuals vs time, observed, or predicted values, or `NULL` if the data contains no plottable entries.

See Also

Other plot functions based on ospsuite.plots: `plotPredictedVsObserved()`, `plotQuantileQuantilePlot()`, `plotResidualsAsHistogram()`, `plotTimeProfile()`

Examples

```
# Generate a residuals vs observed plot for the provided data
plotResidualsVsCovariate(
  dataCombinedAciclovir,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`,
  xAxis = "time",
  residualScale = 'linear'
)

# Generate a residuals vs predicted plot
plotResidualsVsCovariate(dataCombinedAciclovir, xAxis = "predicted")

# Generate a residuals vs time plot
plotResidualsVsCovariate(dataCombinedAciclovir, xAxis = "time")

# Show individual dataset names in legend
plotResidualsVsCovariate(dataCombinedAciclovir, showLegendPerDataset = "observed")
```

plotResidualsVsSimulated *Residuals versus time scatter plot*

Description

[Deprecated]

Usage

```
plotResidualsVsSimulated(
  dataCombined,
  defaultPlotConfiguration = NULL,
  scaling = "lin"
)
```

Arguments

dataCombined	A single instance of DataCombined class containing both observed and simulated datasets to be compared.
defaultPlotConfiguration	A DefaultPlotConfiguration object, which is an R6 class object that defines plot properties.
scaling	A character of length one specifying the scale type for residual. can be lin or log.

See Also

Other plotting: DefaultPlotConfiguration, plotIndividualTimeProfile(), plotObservedVsSimulated(), plotPopulationTimeProfile(), plotResidualsVsTime()

Examples

```
# Create a new instance of `DefaultPlotConfiguration` class
myPlotConfiguration <- DefaultPlotConfiguration$new()
myPlotConfiguration$title <- "My Plot Title"
myPlotConfiguration$subtitle <- "My Plot Subtitle"
myPlotConfiguration$caption <- "My Sources"

# plot
plotResidualsVsSimulated(dataCombinedAciclovir,
  scaling = "log",
  defaultPlotConfiguration = myPlotConfiguration
)
```

plotResidualsVsTime *Residuals versus time scatter plot*

Description

[Deprecated]

Usage

```
plotResidualsVsTime(
  dataCombined,
  defaultPlotConfiguration = NULL,
  scaling = "lin"
)
```

Arguments

dataCombined	A single instance of DataCombined class containing both observed and simulated datasets to be compared.
defaultPlotConfiguration	A DefaultPlotConfiguration object, which is an R6 class object that defines plot properties.
scaling	A character of length one specifying the scale type for residual. can be lin or log.

See Also

Other plotting: DefaultPlotConfiguration, plotIndividualTimeProfile(), plotObservedVsSimulated(), plotPopulationTimeProfile(), plotResidualsVsSimulated()

Examples

```
# Create a new instance of `DefaultPlotConfiguration` class
myPlotConfiguration <- DefaultPlotConfiguration$new()
myPlotConfiguration$title <- "My Plot Title"
myPlotConfiguration$subtitle <- "My Plot Subtitle"
myPlotConfiguration$caption <- "My Sources"

# plot
plotResidualsVsTime(
dataCombinedAciclovir,
scaling = "lin",
defaultPlotConfiguration = myPlotConfiguration
)
```

`plotTimeProfile` *Create Time Profile Plot*

Description

Creates a time profile plot for given data.

This function generates a time profile plot using ggplot2, where the data is grouped by a column named "group".

Usage

```
plotTimeProfile(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  observedMapping = NULL,
  xUnit = NULL,
  yUnit = NULL,
  y2Unit = NULL,
  aggregation = "quantiles",
  quantiles =
   ospsuite.plots::getOspsuite.plots.option(ospsuite.plots::OptionKeys$defaultPercentiles),
  nsd = 1,
  showLegendPerDataset = "all",
  ...
)
```

Arguments

`plotData` An object of class `DataCombined` or a `data.table`. If a `data.table`, it must include the following:

- `xValues`: Numeric time points.
- `yValues`: Observed or simulated values (numeric).
- `group`: Grouping variable (factor or character).
- `name`: Name for the dataset (factor or character).
- `xUnit`: Unit of the x-axis values (character).
- `yUnit`: Unit of the y-axis values (character).
- `dataType`: Specifies data type—either `observed` or `simulated`.
- Optional:
  - `yErrorType`: Type of y error, relative to `ospsuite::DataErrorType`.
  - `yErrorValues`: Numeric error values.
  - `yMin`, `yMax`: Custom ranges for y-axis instead of error types.
  - `IndividualId`: Used for aggregation of simulated population data.

`metaData` A list containing metadata for the plot. If NULL, a default list is constructed from the data. Expected structure includes information about dimensions and units for both x and y axes.

`mapping` A ggplot2 aesthetic mapping object. Default is `ggplot2::aes()`. This is added or replaces the default mapping constructed by the data.

`observedMapping`
A ggplot2 aesthetic mapping for observed data. Default is NULL. Then a copy of mapping without line typical aesthetics like linetype and linewidth is used.

`xUnit` A character string specifying the target unit for the x-axis. If `NULL` (default), the most frequent unit in the data is used. For available units, see `ospsuite::ospUnits`.

`yUnit` A character string specifying the target unit for the primary y-axis. If `NULL` (default), the most frequent unit in the data is used. For available units, see `ospsuite::ospUnits`.

`y2Unit` A character string specifying the target unit for the secondary y-axis (only applicable when data contains two y-dimensions). If `NULL` (default), the most frequent unit in the data is used.

`aggregation` The type of the aggregation of simulated data. One of `quantiles` (Default), `arithmetic` or `geometric` (full list in `ospsuite::DataAggregationMethods`). Will replace `yValues` by the median, arithmetic or geometric average and add a set of upper and lower bounds (`yMin` and `yMax`). It is only applied if the simulated data represents a population.

`quantiles` A numerical vector with quantile values (Default: `c(0.05, 0.50, 0.95)`) to be plotted. Ignored if `aggregation` is not `quantiles`.

`nsd` Optional parameter specifying the number of standard deviations to plot above and below the mean (used for error bars when aggregation is "arithmetic" or "geometric"). Ignored if `aggregation` is `quantiles`.

`showLegendPerDataset`
Controls display of separate legend entries for individual datasets. One of:

- `"none"`: No per-dataset differentiation. Only group-level legend.
- `"all"` (default): Differentiate both observed (via `shape`) and simulated (via `linetype`).
- `"observed"`: Differentiate only observed data via different shapes.
- `"simulated"`: Differentiate only simulated data via different line types.

User-provided `mapping` and `observedMapping` will override internal settings. A warning is issued if the override removes per-dataset differentiation.

`...` Arguments passed on to `ospsuite.plots::plotTimeProfile`

`xScale` either 'linear' then `ggplot2::scale_x_continuous()` or 'log' then `ggplot2::scale_x_log10()` is used

`xScaleArgs` list of arguments passed to `ggplot2::scale_x_continuous()` or `ggplot2::scale_x_log10()`

`yScale` either 'linear' then `ggplot2::scale_y_continuous()` or 'log' then `ggplot2::scale_y_log10()` is used

`yScaleArgs` list of arguments passed to `ggplot2::scale_y_continuous()` or `ggplot2::scale_y_log10()`

y2Scale either 'linear' the secondary axis is displayed linear, or 'log' secondary axis is displayed with log scale

y2ScaleArgs list of arguments passed to ggplot2::sec_axis(), trans, break are set by code

plotObject An optional ggplot object on which to add the plot layers

geomLineAttributes A list with arguments which are passed on to the call ggplot2::geom_line

geomRibbonAttributes A list with arguments which are passed on to the call ggplot2::geom_ribbon

geomPointAttributes A list with arguments which are passed on to the call ggplot2::geom_point

geomErrorbarAttributes A list with arguments which are passed on to the call geom_errorbar_osp

geomLLOQAttributes A list with arguments which are passed on to the call ggplot2::geom_hline

groupAesthetics vector of aesthetics, which are used for columns mapped with groupby,

Details

Automatic Unit Conversion:

When using a DataCombined object, the function automatically converts mixed units to a common unit. The target unit is determined by the most frequently occurring unit in the observed data (or simulated data if no observed data exists). Concentration dimensions (Concentration (mass) and Concentration (molar)) are treated as compatible and can be converted between each other if molecular weight is available.

Mixed Error Types:

The function automatically handles data containing different error type specifications:

- If all data uses the same error type (ArithmeticStdDev or GeometricStdDev), it is passed directly to the plotting function.
- If data contains **mixed error types**, they are automatically converted to yMin/yMax bounds:
  - ArithmeticStdDev: yMin = yValues - yErrorValues, yMax = yValues + yErrorValues
  - GeometricStdDev: yMin = yValues / yErrorValues, yMax = yValues * yErrorValues
- For custom error types (not ArithmeticStdDev or GeometricStdDev), provide error bounds directly in yMin and yMax columns.

Value

A ggplot2 plot object representing the time profile, or NULL if the data contains no plottable entries.

See Also

Other plot functions based on ospsuite.plots: plotPredictedVsObserved(), plotQuantileQuantilePlot(), plotResidualsAsHistogram(), plotResidualsVsCovariate()

Examples

```
# Generate a time profile plot for the provided data
plotTimeProfile(dataCombinedAciclovir,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`mg/l`)

# Show individual dataset names for observed data only
plotTimeProfile(dataCombinedAciclovir, showLegendPerDataset = "observed")

# Show individual dataset names for simulated data only
plotTimeProfile(dataCombinedAciclovir, showLegendPerDataset = "simulated")

# Show individual dataset names for both observed and simulated
plotTimeProfile(dataCombinedAciclovir, showLegendPerDataset = "all")
```

`Population` *Population*

Description

List of individuals used in a population simulation

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `Population`

Active bindings

`count` the number of individual in the population

`allCovariateNames` the names of all covariates defined in the population

`allParameterPaths` the paths of all parameters defined in the population

`allIndividualIds` Ids of individuals defined in the population

Methods

Public methods:

- `Population$has()`
- `Population$setParameterValues()`
- `Population$getParameterValues()`
- `Population$getCovariateValues()`
- `Population$getCovariateValue()`
- `Population$getParameterValuesForIndividual()`
- `Population$remove()`
- `Population$print()`

`Population$has()`: Returns `TRUE` if the population has variability defined for `parameterOrPath` otherwise `FALSE`

Usage:

```
Population$has(parameterOrPath)
```

Arguments:

`parameterOrPath` Parameter instance of parameter path

`Population$setParameterValues()`: Updates or adds the variability values in the population for `parameterOrPath`.

Usage:

```
Population$setParameterValues(parameterOrPath, values)
```

Arguments:

`parameterOrPath` Parameter instance of parameter path. If an entry already exists for this parameter by path, its values be overwritten, otherwise it will be created.

`values` double vector containing the value to set for the `parameterOrPath`

`Population$getParameterValues()`: Returns the variability values defined in the population for `parameterOrPath`

Usage:

```
Population$getParameterValues(parameterOrPath)
```

Arguments:

`parameterOrPath` Parameter instance of parameter path

`Population$getCovariateValues()`: Returns the values defined in the population for the covariate named `covariateName`

Usage:

```
Population$getCovariateValues(covariateName)
```

Arguments:

`covariateName` Name of covariate for which values should be retrieved

`Population$getCovariateValue()`: Returns the values defined in the population for the covariate named `covariateName` and individual with id `individualId`

Usage:

```
Population$getCovariateValue(covariateName, individualId)
```

Arguments:

`covariateName` Name of covariate for which values should be retrieved

`individualId` Id of individual for which the value for covariate `covariateName` should be retrieved

`Population$getParameterValuesForIndividual()`: Returns all values defined in the population the individual with id `individualId`

Usage:

```
Population$getParameterValuesForIndividual(individualId)
```

Arguments:

`individualId` Id of individual for which all values should be returned

`Population$remove()`: Removes the value of a parameter by path

Usage:

```
Population$remove(parameterPath)
```

Arguments:

`parameterPath` Path of the parameter values to remove

`Population$print()`: Print the object to the console

Usage:

```
Population$print(...)
```

Arguments:

`...` Rest arguments.

`PopulationCharacteristics` *PopulationCharacteristics*

Description

Characteristics of a population used for population creation

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `PopulationCharacteristics`

Active bindings

`numberOfIndividuals` Number of individuals in the population

`proportionOfFemales` Proportion of female in the population

`species` Specifies the species of the individual. It should be a species available in PK-Sim (see `Species`)

`population` For a Human species, the population of interest. It should be a population available in PK-Sim (see `HumanPopulation`)

`age` Age range of the population as in instance of a `ParameterRange` (optional)

`gestationalAge` Gestational Age range of the population as in instance of a `ParameterRange` (optional)

`weight` Weight range of the population as in instance of a `ParameterRange` (optional)

`height` Height range of the population as in instance of a `ParameterRange` (optional)

`BMI` BMI range of the population as in instance of a `ParameterRange` (optional)

`allMoleculeOntogenies` All molecule ontogenies defined for this population characteristics.

`seed` Seed used to generate the population

Methods

Public methods:

- PopulationCharacteristics$new()
- PopulationCharacteristics$print()
- PopulationCharacteristics$addMoleculeOntogeny()

PopulationCharacteristics$new(): Initialize a new instance of the class

Usage:

```
PopulationCharacteristics$new()
```

Returns: A new PopulationCharacteristics object.

PopulationCharacteristics$print(): Print the object to the console

Usage:

```
PopulationCharacteristics$print(...)
```

Arguments:

... Rest arguments.

PopulationCharacteristics$addMoleculeOntogeny(): Add a molecule ontogeny MoleculeOntogeny to the individual characteristics

Usage:

```
PopulationCharacteristics$addMoleculeOntogeny(moleculeOntogeny)
```

Arguments:

moleculeOntogeny Molecule ontogeny to add

---

populationFromDataFrame

Creates a Population *object from a data.frame*

Description

Creates a Population object from a data.frame

Usage

```
populationFromDataFrame(dataFrame)
```

Arguments

dataFrame	A data.frame containing population data. Each column represents a parameter path or covariate, with one row per individual. If no IndividualId column is present, one will be automatically generated with 0-based sequential IDs.

Value

A Population object

Examples

```
csvPath <- system.file("extdata", "pop.csv", package = "ospsuite")

population <- loadPopulation(csvPath)
df <- populationToDataFrame(population)
populationFromDf <- populationFromDataFrame(df)
```

---

populationToDataFrame *Creates a data.frame containing one column for each parameter defined in the population*

Description

Creates a data.frame containing one column for each parameter defined in the population

Usage

```
populationToDataFrame(population)

populationToTibble(population)
```

Arguments

population	Population to convert to data frame (typically imported from file using loadPopulation)

Examples

```
csvPath <- system.file("extdata", "pop.csv", package = "ospsuite")

population <- loadPopulation(csvPath)
df <- populationToDataFrame(population)
```

---

potentialVariableParameterPathsFor *Returns an array of parameter path with one entry for each parameter that is used in the simulation and can potentially be used for sensitivity analysis*

Description

Returns an array of parameter path with one entry for each parameter that is used in the simulation and can potentially be used for sensitivity analysis

Usage

```
potentialVariableParameterPathsFor(simulation)
```

Arguments

simulation	Instance of a simulation for which variable parameters should be retrieved

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath)

parameterPaths <- potentialVariableParameterPathsFor(sim)
```

Quantity *Quantity*

Description

A quantity of the model (with unit, value) such as a Parameter or an Amount

Super classes

rSharp::NetObject -> DotNetWrapper -> ObjectBase -> Entity -> Quantity

Active bindings

value The value of the quantity in unit

unit The base unit in which the quantity value is defined (Read-Only)

displayUnit The unit in which the quantity value is usually displayed (Read-Only)

dimension The dimension in which the quantity is defined (Read-Only)

allUnits the list of all supported units (Read-Only)

quantityType The type of the quantity (Read-Only)

formula An instance of a Formula object used by this quantity (Read-Only)

isTable Returns TRUE if the formula used by this quantity is a table formula otherwise FALSE

isConstant Returns TRUE if the formula used by this quantity is a constant formula otherwise FALSE

isFormula Returns TRUE if the formula used by this quantity is an explicit formula (e.g an equation) otherwise FALSE

isDistributed Returns TRUE if the quantity represents a quantity with an underlying distribution otherwise FALSE

formulaString Returns the equation of the formula for a quantity using an explicit formula (e.g. isFormula == TRUE) or NULL for a quantity that does not use an explicit formula.

isFixedValue Returns TRUE if the formula was overridden by a constant value, otherwise FALSE

valueOrigin The value origin of the quantity (Read-Only)

Methods

Public methods:

- Quantity$new()
- Quantity$print()
- Quantity$printValue()
- Quantity$printQuantityValue()
- Quantity$getPrintValue()
- Quantity$setValue()
- Quantity$hasUnit()
- Quantity$reset()

`Quantity$new()`: Initialize a new instance of the class

Usage:

```
Quantity$new(netObject)
```

Arguments:

`netObject` A `NetObject` object with the pointer to the .NET `Quantity`

Returns: A new `Quantity` object.

`Quantity$print()`: Print the object to the console

Usage:

```
Quantity$print(...)
```

Arguments:

`...` Rest arguments.

`Quantity$printValue()`: Print the name of the quantity and its value

Usage:

```
Quantity$printValue()
```

`Quantity$printQuantityValue()`: Print the value (in scientific notation with 2 digits when needed) and unit of the quantity

Usage:

```
Quantity$printQuantityValue(caption)
```

Arguments:

`caption` Text to prepend to the value

`Quantity$getPrintValue()`: Return a string for printing the value (in scientific notation with 2 digits when needed) and unit of the quantity

Usage:

```
Quantity$getPrintValue()
```

Returns: A string for printing the quantity in one line

`Quantity$setValue()`: Convert value from unit to the base unit and sets the value in base unit.

Usage:

```
Quantity$setValue(value, unit = NULL)
```

Arguments:

`value` Value to set. If unit is null, we assume that the value is in base unit

`unit` Optional unit in which the value is given.

`Quantity$hasUnit()`: Returns `TRUE` if the quantity supports the given unit otherwise `FALSE`. For the list of supported units, use `allUnits`

Usage:

```
Quantity$hasUnit(unit)
```

Arguments:

`unit` Unit to check

`Quantity$reset()`: Ensures that the quantity uses the value computed by its formula. It is a shortcut for `self$isFixedValue <- false`.

Usage:

```
Quantity$reset()
```

`QuantityPKParameter` *QuantityPKParameter*

Description

pK-Parameter values for all individuals of a simulation (1 or more) calculated for a specific quantity with path `quantityPath`

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `QuantityPKParameter`

Active bindings

`values` All values for `quantityPath` and `name`

`quantityPath` The path of the quantity for which the values were calculated

`name` The name of the pK-Parameter (AUC, Cmax, Tmax etc...)

`unit` Base unit in which the pk parameter was calculated

`dimension` Dimension in which the pk parameter was calculated

Methods

Public methods:

- QuantityPKParameter$new()
- QuantityPKParameter$print()

QuantityPKParameter$new(): Initialize a new instance of the class

Usage:

```
QuantityPKParameter$new(netObject)
```

Arguments:

`netObject` An `rSharp::NetObject` object.

Returns: A new `QuantityPKParameter` object.

QuantityPKParameter$print(): Print the object to the console

Usage:

```
QuantityPKParameter$print(...)
```

Arguments:

`...` Rest arguments.

`QuantitySelection` *QuantitySelection*

Description

List of quantities selected as output for a given simulation

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `QuantitySelection`

Active bindings

`path` Path of selected quantity

`quantityType` Type of selected quantity

Methods

Public methods:

- QuantitySelection$print()

QuantitySelection$print(): Print the object to the console

Usage:

```
QuantitySelection$print(...)
```

Arguments:

`...` Rest arguments.

`removeAllUserDefinedPKParameters` *Removes all User-Defined PK-Parameters that may have been added to the system*

Description

Removes all User-Defined PK-Parameters that may have been added to the system

Usage

```
removeAllUserDefinedPKParameters()
```

`removeSimulationFromCache` *Removes a simulation from simulations cache.*

Description

Removes a simulation from simulations cache.

Usage

```
removeSimulationFromCache(simulation)
```

Arguments

`simulation`	Simulation to be removed from the cache

Value

TRUE if the simulation was cached and could be removed from cache. FALSE otherwise, usually indicating that the specific simulation was not cached.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim1 <- loadSimulation(simPath)
sim2 <- loadSimulation(simPath, loadFromCache = FALSE, addToCache = FALSE)

removeSimulationFromCache(sim1) # returns TRUE
removeSimulationFromCache(sim2) # returns FALSE
```

resetSimulationCache *Clears cache of loaded simulations*

Description

Clears cache of loaded simulations

Usage

```
resetSimulationCache()
```

runSensitivityAnalysis *Runs a sensitivity analysis*

Description

Runs a sensitivity analysis

Usage

```
runSensitivityAnalysis(
  sensitivityAnalysis,
  sensitivityAnalysisRunOptions = NULL
)
```

Arguments

sensitivityAnalysis	Instance of a `SensitivityAnalysis` to run
sensitivityAnalysisRunOptions	Optional instance of a `SensitivityAnalysisRunOptions` used during the sensitivity analysis run

Value

SimulationResults (one entry per Individual)

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Create a new sensitivity object for the simulation
sensitivity <- SensitivityAnalysis$new(sim)

# Runs the sensitivity analysis
results <- runSensitivityAnalysis(sensitivity)
```

`runSimulationBatches` *Run simulation batches*

Description

Run simulation batches

Usage

```
runSimulationBatches(
  simulationBatches,
  simulationRunOptions = NULL,
  silentMode = FALSE,
  stopIfFails = FALSE
)
```

Arguments

`simulationBatches`	List of `SimulationBatch` objects with added parameter and initial values
`simulationRunOptions`	Optional instance of a `SimulationRunOptions` used during the simulation run.
`silentMode`	If `TRUE`, no warnings are displayed if a simulation fails. Default is `FALSE`. Has no effect if `stopIfFails` is `TRUE`.
`stopIfFails`	Whether to stop the execution if one of the simulations failed. Default is `FALSE`.

Details

Runs a set of simulation batches. The simulation batches must be populated with sets of parameter and start values with `SimulationBatch$addRunValues()` prior to running. After the run, the list of parameter and start values is cleared.

Value

Nested list of `SimulationResults` objects. The first level of the fist are the IDs of the SimulationBatches, containing a list of `SimulationResults` for each set of parameter/initial values. If a simulation with a parameter/initial values set fails, the result for this run is `NULL`

Examples

```
## Not run:
sim1 <- loadSimulation("sim1", loadFromCache = TRUE)
sim2 <- loadSimulation("sim2", loadFromCache = TRUE)
parameters <- c("Organism|Liver|Volume", "R1|k1")
molecules <- "Organism|Liver|A"
# Create two simulation batches.
simulationBatch1 <- createSimulationBatch(
  simulation = sim1,
```

```
  parametersOrPaths = parameters,
  moleculesOrPaths = molecules
)
simulationBatch2 <- createSimulationBatch(
  simulation = sim2,
  parametersOrPaths = parameters,
  moleculesOrPaths = molecules
)
# Ids of run values
ids <- c()
ids[[1]] <- simulationBatch1$addRunValues(parameterValues = c(1, 2), initialValues = 1)
ids[[2]] <- simulationBatch1$addRunValues(parameterValues = c(1.6, 2.4), initialValues = 3)
ids[[3]] <- simulationBatch2$addRunValues(parameterValues = c(4, 2), initialValues = 4)
ids[[4]] <- simulationBatch2$addRunValues(parameterValues = c(2.6, 4.4), initialValues = 5)
res <- runSimulationBatches(simulationBatches = list(simulationBatch1, simulationBatch2))

## End(Not run)
```

`runSimulations` *Runs multiple simulations concurrently.*

Description

Runs multiple simulations concurrently.

Usage

```
runSimulations(
  simulations,
  population = NULL,
  agingData = NULL,
  simulationRunOptions = NULL,
  silentMode = FALSE,
  stopIfFails = FALSE
)
```

Arguments

`simulations`	One `Simulation` or a list or vector of `Simulation` objects to simulate. List or vector can be named (names must be uniques), in which case the names will reused in the `simulationResults` output list. If not named, the output list will use simulation ids for names.
`population`	Optional instance of a `Population` to use for the simulation. Only allowed when simulating one simulation. Alternatively, you can also pass the result of `createPopulation` directly. In this case, the population will be extracted.
`agingData`	Optional instance of `AgingData` to use for the simulation. This is only used with a population simulation

simulationRunOptions
: Optional instance of a SimulationRunOptions used during the simulation run

silentMode
: If TRUE, no warnings are displayed if a simulation fails. Default is FALSE. Has no effect if stopIfFails is TRUE.

stopIfFails
: Whether to stop the execution if one of the simulations failed. Default is FALSE.

Details

For multiple simulations, only individual simulations are possible. For single simulation, either individual or population simulations can be performed.

Value

A named list of SimulationResults objects with names being the IDs of the respective simulations. If a simulation fails, the result for this simulation is NULL

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Running an individual simulation
# Results is a list with one object `SimulationResults`
results <- runSimulations(sim)

# Creating custom simulation run options

simRunOptions <- SimulationRunOptions$new()
simRunOptions$numberOfCores <- 3
simRunOptions$showProgress <- TRUE

# Running a population simulation
popPath <- system.file("extdata", "pop.csv", package = "ospsuite")
population <- loadPopulation(popPath)
results <- runSimulations(sim, population, simulationRunOptions = simRunOptions)[[1]]

# Running multiple simulations in parallel
sim2 <- loadSimulation(simPath)

# Results is a list of `SimulationResults`
results <- runSimulations(list(sim, sim2))
```

---

runSimulationsFromSnapshot

Run Simulations From Snapshot Files

Description

Run Simulations From Snapshot Files

Usage

```
runSimulationsFromSnapshot(
  ...,
  output = ".",
  RunForAllOutputs = FALSE,
  exportCSV = TRUE,
  exportPKML = FALSE,
  exportJSON = FALSE,
  exportXML = FALSE
)
```

Arguments

...	character strings, path to snapshot files or a directory containing snapshot files
output	character string, path to the output directory where to write simulation results
RunForAllOutputs	logical, whether to run the simulation for all outputs or only OutputSelections (default = FALSE)
exportCSV	logical, whether to export the results as csv (default = TRUE)
exportPKML	logical, whether to export the results as pkml (default = FALSE)
exportJSON	logical, whether to export simulation results as json (default = FALSE)
exportXML	logical, whether to export the results as xml (default = FALSE)

Examples

```
## Not run:
runSimulationsFromSnapshot("path/to/my_snapshot.json", csv = TRUE, pkml = TRUE)

## End(Not run)
```

---

saveDataSetToPKML *Save the* DataSet *to pkml*

Description

Save the DataSet to pkml

Usage

```
saveDataSetToPKML(dataSet, filePath)
```

Arguments

dataSet	The DataSet object
filePath	Path where the pkml file will be created

Details

Save the DataSet to a pkml file that can be loaded by MoBi

Examples

```
## Not run:
dataSet <- DataSet$new(name = "NewDataSet")
dataSet$setValues(xValues = c(1, 2, 3, 4, 5), yValues = c(10, 20, 30, 40, 50))
dataSet$saveToPKML(filePath = "../ObsData.pkml")

## End(Not run)
```

---

saveSimulation *Saves a simulation to pkml file*

Description

Saves a simulation to pkml file

Usage

```
saveSimulation(simulation, filePath)
```

Arguments

simulation	Instance of a simulation to save.
filePath	Full path of where the simulation will be saved.

---

scaleParameterValues *Scale current values of parameters using a factor*

Description

Scale current values of parameters using a factor

Usage

```
scaleParameterValues(parameters, factor)
```

Arguments

parameters	A single or a list of Parameter
factor	A numeric value that will be used to scale all parameters

See Also

getParameter() and getAllParametersMatching() to create objects of type Parameter

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
param <- getParameter("Organism|Liver|Volume", sim)
scaleParameterValues(param, 1)
params <- getAllParametersMatching("Organism|**|Volume", sim)
scaleParameterValues(params, 1.5)
```

SensitivityAnalysis *SensitivityAnalysis*

Description

Supports Sensitivity Analysis workflow to assess the impact of input parameters on the simulation outputs

Super classes

rSharp::NetObject -> DotNetWrapper -> SensitivityAnalysis

Active bindings

`simulation` Reference to the `Simulation` used to calculate or import the sensitivity analysis results (Read-Only).

`numberOfSteps` Number of steps used for the variation of each parameter in one direction from the reference value. The parameter is varied in both positive and negative directions, so the total number of variations per parameter is `2 * numberOfSteps`. For example, `numberOfSteps = 2` with `variationRange = 0.1` tests the parameter at four points: 90%, 95%, 105%, and 110% of the reference value. Default value can be retrieved with `getOSPSuiteSetting("sensitivityAnalysisConfig")$n`

`variationRange` Relative variation range applied to each parameter. This defines the total range of variation from the reference value. For example, `variationRange = 0.1` means ±10% variation. Combined with `numberOfSteps = 2`, the parameter would be tested at 90%, 95%, 105%, and 110% of its reference value (i.e., the variation range is divided into `numberOfSteps` equal intervals in each direction). Default value can be retrieved with `getOSPSuiteSetting("sensitivityAnalysisConfig`

`parameterPaths` List of parameters to use for sensitivity calculation. If empty, the sensitivity will be performed automatically on all constant parameters that are really in use in the simulation. Constant parameter means all parameters with a constant value or a formula parameter with a value that was overridden by the user

Methods

Public methods:

- SensitivityAnalysis$new()
- SensitivityAnalysis$addParameterPaths()
- SensitivityAnalysis$clearParameterPaths()
- SensitivityAnalysis$print()

`SensitivityAnalysis$new()`: Initialize a new instance of the class

Usage:

```
SensitivityAnalysis$new(
  simulation,
  parameterPaths = NULL,
  numberOfSteps = ospsuiteEnv$sensitivityAnalysisConfig$numberOfSteps,
  variationRange = ospsuiteEnv$sensitivityAnalysisConfig$variationRange
)
```

Arguments:

`simulation` Simulation for which a sensitivity analysis should be performed

`parameterPaths` Vector of parameter paths to use for sensitivity calculation (optional). If undefined, the sensitivity will be performed automatically on all constant parameters of the simulation. Constant parameter means all parameters with a constant value or a formula parameter with a value that was overridden by the user

`numberOfSteps` Number of steps used for the variation of each parameter in one direction from the reference value (optional, default specified in `getOSPSuiteSetting("sensitivityAnalysisConfig")`). The parameter is varied in both positive and negative directions, so the total number of variations per parameter is `2 * numberOfSteps`. For example, with `numberOfSteps = 2` and `variationRange = 0.1`, each parameter will be tested at four points: 90% (refValue * 0.9), 95% (refValue * 0.95), 105% (refValue * 1.05), and 110% (refValue * 1.1) of its reference value. The total number of simulations is `2 * numberOfSteps * number_of_parameters`.

`variationRange` Relative variation range applied to each parameter (optional, default specified in `getOSPSuiteSetting("sensitivityAnalysisConfig")`). This defines the total range of variation. For example, `variationRange = 0.1` means ±10% variation. The variation range is divided into `numberOfSteps` equal intervals in each direction (positive and negative).

Returns: A new `SensitivityAnalysis` object.

`SensitivityAnalysis$addParameterPaths()`: Adds the parameterPaths to the list of parameter path to vary in the sensitivity analysis

Usage:

```
SensitivityAnalysis$addParameterPaths(parameterPaths)
```

Arguments:

`parameterPaths` Parameter paths to add (single or multiple values) If no parameters were specified during creating of a `SensitivityAnalysis` (all constant parameters are considered), calling `addParameterPaths` will make only the manually added parameters being varied.

`SensitivityAnalysis$clearParameterPaths()`: Removes all parameter paths defined in the Sensitivity Analysis

Usage:

```
SensitivityAnalysis$clearParameterPaths()
```

`SensitivityAnalysis$print()`: Print the object to the console

Usage:

```
SensitivityAnalysis$print(...)
```

Arguments:

`...` Rest arguments.

`SensitivityAnalysisResults` *SensitivityAnalysisResults*

Description

Results of a sensitivity analysis run (either individual or population simulation).

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SensitivityAnalysisResults`

Active bindings

`simulation` Reference to the `Simulation` used to calculate or import the sensitivity analysis results (Read-Only).

`count` the number of pk parameter sensitivity entries

`allPKParameterNames` Returns the name of all PK-Parameters available in this results. This will be a subset of all potential PK-Parameters available in the system.

`allQuantityPaths` Returns the path of all outputs available in this results.

Methods

Public methods:

- `SensitivityAnalysisResults$new()`
- `SensitivityAnalysisResults$allPKParameterSensitivitiesFor()`
- `SensitivityAnalysisResults$pkParameterSensitivityValueFor()`
- `SensitivityAnalysisResults$print()`

`SensitivityAnalysisResults$new()`: Initialize a new instance of the class

Usage:

```
SensitivityAnalysisResults$new(netObject, simulation)
```

Arguments:

`netObject` A `NetObject`.

`simulation` Reference to the simulation object used to calculated the results.

Returns: A new `SensitivityAnalysisResults` object.

`SensitivityAnalysisResults$allPKParameterSensitivitiesFor()`: Returns the `PKParameterSensitivity` for a given `pkParameter` and output participating to a total sensitivity greater or equal to `totalSensitivityThreshold`.

Usage:

```
SensitivityAnalysisResults$allPKParameterSensitivitiesFor(
  pkParameterName,
  outputPath,
  totalSensitivityThreshold =
    ospsuiteEnv$sensitivityAnalysisConfig$totalSensitivityThreshold
)
```

Arguments:

`pkParameterName` Name of `pkParameter` for which sensitivity should be retrieved.

`outputPath` Path of the output for which the sensitivity should be retrieved

`totalSensitivityThreshold` Threshold used to filter out the most sensitive parameter. A threshold of `0.9` means that only parameter participating to a total of `90` percent of the sensitivity would be returned. A value of `1` would return the sensitivity for all parameters. For a detailed explanation of how this threshold is used, see OSPS documentation. The default value can be retrieved with `getOSPSuiteSetting("sensitivityAnalysisConfig")$totalSensitivityThresho` and can be changed by setting `ospsuiteEnv$sensitivityAnalysisConfig$totalSensitivityThreshold <- newValue`.

`SensitivityAnalysisResults$pkParameterSensitivityValueFor()`: Returns the sensitivity value for a given `pkParameter`, output and model parameter (either by path or by name). If the sensitivity result does not exist, returns `NaN`.

Usage:

```
SensitivityAnalysisResults$pkParameterSensitivityValueFor(
  pkParameterName,
  outputPath,
  parameterName = NULL,
  parameterPath = NULL
)
```

Arguments:

`pkParameterName` Name of `pkParameter` for which sensitivity should be retrieved.

`outputPath` Path of the output for which the sensitivity should be retrieved.

`parameterName` Name of the sensitivity parameter for which the sensitivity should be retrieved.

`parameterPath` Path of the sensitivity parameter for which the sensitivity should be retrieved. Wildcards (*) not accepted.

`SensitivityAnalysisResults$print()`: Print the object to the console

Usage:

```
SensitivityAnalysisResults$print(...)
```

Arguments:

`...` Rest arguments.

SensitivityAnalysisRunOptions
SensitivityAnalysisRunOptions

Description

Options to be passed to the sensitivity analysis engine

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SensitivityAnalysisRunOptions`

Active bindings

`numberOfCores` (Maximal) number of cores to be used. Per default set to `getOSPSuiteSetting("numberOfCores")`.

`showProgress` Specifies whether a progress bar should be shown during sensitivity analysis. If `TRUE`, a progress bar is shown in the console, indicating the progress of the sensitivity analysis calculations. Default is `getOSPSuiteSetting("showProgress")`.

Methods

Public methods:

- `SensitivityAnalysisRunOptions$new()`
- `SensitivityAnalysisRunOptions$print()`

`SensitivityAnalysisRunOptions$new()`: Initialize a new instance of the class

Usage:

```
SensitivityAnalysisRunOptions$new(numberOfCores = NULL, showProgress = NULL)
```

Arguments:

`numberOfCores` Number of cores to use for the simulation. Default value is `getOSPSuiteSetting("numberOfCores")`

`showProgress` Should a progress bar be displayed during sensitivity analysis. If `TRUE`, a progress bar is shown in the console, indicating the progress of the sensitivity analysis calculations. Default value is `getOSPSuiteSetting("showProgress")`

Returns: A new `SensitivityAnalysisRunOptions` object.

`SensitivityAnalysisRunOptions$print()`: Print the object to the console

Usage:

```
SensitivityAnalysisRunOptions$print(...)
```

Arguments:

`...` Rest arguments.

setMoleculeInitialValues

Set molecule start values

Description

Set molecule start values

Usage

```
setMoleculeInitialValues(molecules, values, units = NULL)
```

Arguments

molecules	A single or a list of Molecule
values	A numeric value that should be assigned to the molecule start value or a vector of numeric values, if the start value of more than one molecule should be changed. Must have the same length as molecules. Alternatively, the value can be a unique number. In that case, the same value will be set for all molecules
units	A string or a list of strings defining the units of the values. If NULL (default), values are assumed to be in base units. If not NULL, must have the same length as quantities.

See Also

getMolecule() and getAllMoleculesMatching() to retrieve objects of type Molecule

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
molecule <- getMolecule("Organism|Liver|A", sim)
setMoleculeInitialValues(molecule, 1)
molecules <- getAllMoleculesMatching("Organism|**|A", sim)
setMoleculeInitialValues(molecules, c(2, 3), units = c("pmol", "mmol"))
```

setMoleculeScaleDivisors

Set molecule scale divisors

Description

Set molecule scale divisors

Usage

```
setMoleculeScaleDivisors(molecules, values)
```

Arguments

molecules	A single or a list of Molecule
values	A numeric value that should be assigned to the molecule scale factor or a vector of numeric values, if the scale factor of more than one molecule should be changed. Must have the same length as molecules

See Also

getMolecule() and getAllMoleculesMatching() to retrieve objects of type Molecule

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
molecule <- getMolecule("Organism|Liver|A", sim)
setMoleculeScaleDivisors(molecule, 0.001)
molecules <- getAllMoleculesMatching("Organism|**|A", sim)
setMoleculeScaleDivisors(molecules, c(0.002, 0.003))
```

---

setMoleculeValuesByPath

Set molecule start values in the simulation by path

Description

Set molecule start values in the simulation by path

Usage

```
setMoleculeValuesByPath(
  moleculePaths,
  values,
  simulation,
  units = NULL,
  stopIfNotFound = TRUE
)
```

Arguments

moleculePaths	A single or a list of molecule paths
values	A numeric value that should be assigned to the molecule start value or a vector of numeric values, if the start value of more than one molecule should be changed. Must have the same length as moleculePaths
simulation	Simulation containing the quantities
units	A string or a list of strings defining the units of the values. If NULL (default), values are assumed to be in base units. If not NULL, must have the same length as quantityPaths.

stopIfNotFound	Boolean. If TRUE (default) and no quantity exists for the given path, an error is thrown. If FALSE, a warning is shown to the user.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
setMoleculeValuesByPath("Organism|Liver|A", 1, sim)

setMoleculeValuesByPath(
  c("Organism|Liver|A", "Organism|Liver|B"),
  c(2, 3),
  sim,
  units = c("\u00b5mol", "mmol")
)
```

---

setOutputInterval *Clears the output interval from the simulation and adds a new one.*

Description

Clears the output interval from the simulation and adds a new one.

Usage

```
setOutputInterval(
  simulation,
  startTime,
  endTime,
  resolution,
  intervalName = NULL
)
```

Arguments

simulation	Simulation for which a new interval should be created
startTime	Start time of the interval in min
endTime	End time of the interval in min
resolution	resolution in points/min
intervalName	Optional Name of interval. If not specified, a unique name will be assigned.

Value

Returns the interval created.

Note

This is essentially a shortcut for clearOutputIntervals followed by addOutputInterval

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")

# Load the simulation
sim <- loadSimulation(simPath, addToCache = FALSE, loadFromCache = FALSE)

# Adds a new interval starting at 1h and ending at 10h with a resolution of 10 points per hour
setOutputInterval(sim, 1 * 60, 10 * 60, 1 / 6)
```

---

setOutputs *Set outputs of a simulation*

Description

Sets the quantities as output of the `simulation`. The quantities can either be specified using explicit instances or using paths. This function clears the output selection before adding the new quantities. See `addOutputs` for adding quantities without clearing the output selection. See `clearOutputs` for clearing the output selection without adding new quantities.

Usage

```
setOutputs(quantitiesOrPaths, simulation, stopIfNotFound = TRUE)
```

Arguments

`quantitiesOrPaths`	Quantity instances (element or vector) (typically retrieved using `getAllQuantitiesMatching`) or quantity path (element or vector) to add.
`simulation`	Instance of a simulation for which output selection should be updated.
`stopIfNotFound`	Boolean. If `TRUE` (default) and no quantity exists for the given path, an error is thrown. If `FALSE`, `NULL` is returned.

---

setParameterValues *Set values of parameters*

Description

Set values of parameters

Usage

```
setParameterValues(parameters, values, units = NULL)
```

Arguments

parameters	A single or a list of Parameter
values	A numeric value that should be assigned to the parameter or a vector of numeric values, if the value of more than one parameter should be changed. Must have the same length as 'parameters'. Alternatively, the value can be a unique number. In that case, the same value will be set in all parameters
units	A string or a list of strings defining the units of the values. If NULL (default), values are assumed to be in base units. If not NULL, must have the same length as quantities.

See Also

getParameter() and getAllParametersMatching() to create objects of type Parameter

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
param <- getParameter("Organism|Liver|Volume", sim)
setParameterValues(param, 1)
params <- getAllParametersMatching("Organism|**|Volume", sim)
setParameterValues(params, c(2, 3, 4), units = c("ml", "l", "ml"))
```

---

setParameterValuesByPath

Set the values of parameters in the simulation by path

Description

Set the values of parameters in the simulation by path

Usage

```
setParameterValuesByPath(
  parameterPaths,
  values,
  simulation,
  units = NULL,
  stopIfNotFound = TRUE
)
```

Arguments

parameterPaths	A single or a list of parameter path
values	A numeric value that should be assigned to the parameters or a vector of numeric values, if the value of more than one parameter should be changed. Must have the same length as 'parameterPaths'

simulation	Simulation uses to retrieve parameter instances from given paths.
units	A string or a list of strings defining the units of the values. If NULL (default), values are assumed to be in base units. If not NULL, must have the same length as quantityPaths.
stopIfNotFound	Boolean. If TRUE (default) and no parameter exists for the given path, an error is thrown. If FALSE, a warning is shown to the user

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
setParameterValuesByPath("Organism|Liver|Volume", 1, sim)

setParameterValuesByPath(c("Organism|Liver|Volume", "Organism|Volume"), c(2, 3), sim)
```

---

setQuantityValuesByPath

Set the values of quantities in the simulation by path

Description

Set the values of quantities in the simulation by path

Usage

```
setQuantityValuesByPath(
  quantityPaths,
  values,
  simulation,
  units = NULL,
  stopIfNotFound = TRUE
)
```

Arguments

quantityPaths	A single or a list of absolute quantity paths
values	A numeric value that should be assigned to the quantities or a vector of numeric values, if the value of more than one quantity should be changed. Must have the same length as 'quantityPaths'.
simulation	Simulation containing the quantities
units	A string or a list of strings defining the units of the values. If NULL (default), values are assumed to be in base units. If not NULL, must have the same length as quantityPaths.
stopIfNotFound	Boolean. If TRUE (default) and no quantity exists for the given path, an error is thrown. If FALSE, a warning is shown to the user.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
setQuantityValuesByPath("Organism|Liver|Volume", 1, sim)

setParameterValuesByPath(list("Organism|Liver|Volume", "Organism|Liver|A"), c(2, 3), sim)
```

Simulation *Simulation*

Description

An OSPSuite simulation

Super classes

rSharp::NetObject -> DotNetWrapper -> ObjectBase -> Simulation

Active bindings

`root` Root container of the simulation (read-only)

`path` Path of the root container of the simulation (read-only)

`solver` SimulationSolver object for the simulation (read-only)

`outputSchema` outputSchema object for the simulation (read-only)

`outputSelections` outputSelections object for the simulation (read-only)

`sourceFile` Path to the file the simulation was loaded from (read-only)

`name` Name of the simulation

Methods

Public methods:

- Simulation$new()
- Simulation$allEndogenousStationaryMoleculeNames()
- Simulation$allXenobioticFloatingMoleculeNames()
- Simulation$allStationaryMoleculeNames()
- Simulation$allFloatingMoleculeNames()
- Simulation$molWeightFor()
- Simulation$allApplicationsFor()
- Simulation$print()

`Simulation$new()`: Initialize a new instance of the class

Usage:

```
Simulation$new(netObject, sourceFile = NULL)
```

Arguments:

`netObject` Reference to `NetObject` .NET simulation object

`sourceFile` (Optional) File used to load the simulation

Returns: A new `Simulation` object.

`Simulation$allEndogenousStationaryMoleculeNames()`: Returns the name of all endogenous stationary molecules defined in the simulation. (e.g. with the flag IsStationary = TRUE) This is a typically a molecule that is individual specific such as en Enzyme, Protein, Transporter, FcRn etc.

Usage:

```
Simulation$allEndogenousStationaryMoleculeNames()
```

`Simulation$allXenobioticFloatingMoleculeNames()`: Returns the name of all xenobiotic floating molecules defined in the simulation. (e.g. with the flag IsStationary = FALSE) This is typically a molecule that is being explicitly simulated such as Compound, Inhibitor, DrugComplex.

Usage:

```
Simulation$allXenobioticFloatingMoleculeNames()
```

`Simulation$allStationaryMoleculeNames()`: Returns the name of all stationary molecules defined in the simulation. (e.g. with the flag IsStationary = TRUE)

Usage:

```
Simulation$allStationaryMoleculeNames()
```

`Simulation$allFloatingMoleculeNames()`: Returns the name of all floating molecules defined in the simulation. (e.g. with the flag IsStationary = FALSE)

Usage:

```
Simulation$allFloatingMoleculeNames()
```

`Simulation$molWeightFor()`: Returns the mol weight value (in core unit) associated to the quantity with given path or NA if not found

Usage:

```
Simulation$molWeightFor(quantityPath)
```

Arguments:

`quantityPath` Path of quantity used to retrieve the molecular weight

`Simulation$allApplicationsFor()`: Returns the applications ordered by start time associated to the quantity with path `quantityPath` or an empty list if not found

Usage:

```
Simulation$allApplicationsFor(quantityPath)
```

Arguments:

`quantityPath` Path of quantity used to retrieve the applications (e.g. applications resulting in this quantity being applied)

`Simulation$print()`: Print the object to the console

Usage:

```
Simulation$print(...)
```

Arguments:

`...` Rest arguments.

`SimulationBatch` *SimulationBatch*

Description

An optimized simulation with faster loading. The corresponding .NET class is "OSPSuite.R.Services.ConcurrentRunSimulati

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SimulationBatch`

Active bindings

`simulation` Underlying simulation used for the batch run. Read only.

`runValuesIds` Ids of the run values that will be executed on next run

`id` The id of the .NET wrapped object. (read-only)

Methods

Public methods:

- `SimulationBatch$new()`
- `SimulationBatch$addRunValues()`
- `SimulationBatch$getVariableParameters()`
- `SimulationBatch$getVariableMolecules()`
- `SimulationBatch$print()`

`SimulationBatch$new()`**:** Initialize a new instance of the class

Usage:

```
SimulationBatch$new(netObject, simulation)
```

Arguments:

`netObject` An `rSharp::NetObject` object.

`simulation` Simulation used in the batch run

Returns: A new `SimulationBatch` object.

`SimulationBatch$addRunValues()`**:** Add a set of parameter and start values for next execution.

Usage:

```
SimulationBatch$addRunValues(parameterValues = NULL, initialValues = NULL)
```

Arguments:

parameterValues Vector of parameter values to set in the simulation (default is NULL)

initialValues Vector of initial values to set in the simulation (default is NULL)

Details: Intended for the use with runSimulationBatches. The simulation batch is executed with the sets of parameter and initial values that have been scheduled. The set of run values is cleared after successful run.

Returns: Id of the values set that can be used to get the correct result from runSimulationBatches.

Examples:

```
sim1 <- loadSimulation("sim1", loadFromCache = TRUE)
sim2 <- loadSimulation("sim2", loadFromCache = TRUE)
parameters <- c("Organism|Liver|Volume", "R1|k1")
molecules <- "Organism|Liver|A"
# Create two simulation batches.
simulationBatch1 <- createSimulationBatch(simulation = sim1,
parametersOrPaths = parameters,
moleculesOrPaths = molecules)
simulationBatch2 <- createSimulationBatch(simulation = sim2,
parametersOrPaths = parameters,
moleculesOrPaths = molecules)
#Ids of run values
ids <- c()
ids[[1]] <- simulationBatch1$addRunValues(parameterValues = c(1, 2), initialValues = 1)
ids[[2]] <- simulationBatch1$addRunValues(parameterValues = c(1.6, 2.4), initialValues = 3)
ids[[3]] <- simulationBatch2$addRunValues(parameterValues = c(4, 2), initialValues = 4)
ids[[4]] <- simulationBatch2$addRunValues(parameterValues = c(2.6, 4.4), initialValues = 5)
res <- runSimulationBatches(simulationBatches = list(simulationBatch1, simulationBatch2))
```

SimulationBatch$getVariableParameters(): Returns a list of parameter paths that are variable in this batch.

Usage:

```
SimulationBatch$getVariableParameters()
```

Details: The order of parameters is the same as the order of parameter values added with $addRunValues() method.

Returns: List of parameter paths, or NULL if no parameter is variable.

SimulationBatch$getVariableMolecules(): Returns a list of molecules paths that are variable in this batch

Usage:

```
SimulationBatch$getVariableMolecules()
```

Details: The order of molecules is the same as the order of molecule start values added with $addRunValues() method.

Returns: List of parameter paths, or NULL if no molecule is variable.

SimulationBatch$print(): Print the object to the console

Usage:

```
SimulationBatch$print(...)
```

Arguments:

... Additional arguments.

Examples

```
## ------------------------------------------------
## Method `SimulationBatch$addRunValues()`
## ------------------------------------------------

## Not run:
sim1 <- loadSimulation("sim1", loadFromCache = TRUE)
sim2 <- loadSimulation("sim2", loadFromCache = TRUE)
parameters <- c("Organism|Liver|Volume", "R1|k1")
molecules <- "Organism|Liver|A"
# Create two simulation batches.
simulationBatch1 <- createSimulationBatch(simulation = sim1,
parametersOrPaths = parameters,
moleculesOrPaths = molecules)
simulationBatch2 <- createSimulationBatch(simulation = sim2,
parametersOrPaths = parameters,
moleculesOrPaths = molecules)
#Ids of run values
ids <- c()
ids[[1]] <- simulationBatch1$addRunValues(parameterValues = c(1, 2), initialValues = 1)
ids[[2]] <- simulationBatch1$addRunValues(parameterValues = c(1.6, 2.4), initialValues = 3)
ids[[3]] <- simulationBatch2$addRunValues(parameterValues = c(4, 2), initialValues = 4)
ids[[4]] <- simulationBatch2$addRunValues(parameterValues = c(2.6, 4.4), initialValues = 5)
res <- runSimulationBatches(simulationBatches = list(simulationBatch1, simulationBatch2))

## End(Not run)
```

SimulationBatchOptions

SimulationBatchOptions

Description

Options to be passed to the `SimulationBatch`.

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SimulationBatchOptions`

Active bindings

`variableParameters` Vector of absolute parameter paths to be varied in a simulation batch

`variableMolecules` Vector of absolute molecule paths to be varied in a simulation batch

Methods

Public methods:

- `SimulationBatchOptions$new()`

- SimulationBatchOptions$print()

SimulationBatchOptions$new(): Initialize a new instance of the class

Usage:

```
SimulationBatchOptions$new(variableParameters = NULL, variableMolecules = NULL)
```

Arguments:

variableParameters Vector of absolute parameter paths to be varied in a simulation batch

variableMolecules Vector of absolute molecule paths to be varied in a simulation batch

Returns: A new SimulationBatchOptions object.

SimulationBatchOptions$print(): Print the object to the console

Usage:

```
SimulationBatchOptions$print(...)
```

Arguments:

... Rest arguments.

SimulationBatchRunValues *SimulationBatchRunValues*

Description

Options to be passed to the SimulationBatch run

Super classes

rSharp::NetObject -> DotNetWrapper -> SimulationBatchRunValues

Active bindings

parameterValues Vector of parameter values used in a batch run

initialValues Vector of initial values used in a batch run

id Internal id of the batch run value

Methods

Public methods:

- SimulationBatchRunValues$new()
- SimulationBatchRunValues$print()

SimulationBatchRunValues$new(): Initialize a new instance of the class

Usage:

```
SimulationBatchRunValues$new(parameterValues = NULL, initialValues = NULL)
```

Arguments:

`parameterValues` Vector of parameter values

`initialValues` Vector of molecule initial values

Returns: A new `SimulationBatchRunValues` object.

`SimulationBatchRunValues$print()`: Print the object to the console

Usage:

```
SimulationBatchRunValues$print(...)
```

Arguments:

`...` Rest arguments.

`SimulationPKAnalyses` *SimulationPKAnalyses*

Description

pK-Analyses of a simulation (either individual or population simulation).

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SimulationPKAnalyses`

Active bindings

`simulation` Reference to the `Simulation` used to calculate or import the PK-Analyses (Read-Only)

`allPKParameterNames` Returns the name of all pk parameters for which a value is available

`allQuantityPaths` Returns the path of all quantities for which pk parameters were calculated

Methods

Public methods:

- `SimulationPKAnalyses$new()`
- `SimulationPKAnalyses$allPKParametersFor()`
- `SimulationPKAnalyses$pKParameterFor()`
- `SimulationPKAnalyses$print()`

`SimulationPKAnalyses$new()`: Initialize a new instance of the class

Usage:

```
SimulationPKAnalyses$new(netObject, simulation)
```

Arguments:

`netObject` A `NetObject`

`simulation` Simulation for which the pkParameters were calculated

Returns: A new `SimulationPKAnalyses` object.

`SimulationPKAnalyses$allPKParametersFor()`: Returns all QuantityPKParameter defined for a given path

Usage:

```
SimulationPKAnalyses$allPKParametersFor(quantityPath)
```

Arguments:

`quantityPath` Path for which pkParameters should be retrieved

`SimulationPKAnalyses$pKParameterFor()`: The pK Parameter defined for the given path and name

Usage:

```
SimulationPKAnalyses$pKParameterFor(quantityPath, pkParameter)
```

Arguments:

`quantityPath` Path for which the pkParameter named `pkParameter` should be retrieved

`pkParameter` Name of the pkParameter to retrieve

`SimulationPKAnalyses$print()`: Print the object to the console

Usage:

```
SimulationPKAnalyses$print(...)
```

Arguments:

`...` Rest arguments.

`SimulationResults` *SimulationResults*

Description

Results of a simulation run (either individual or population simulation)

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SimulationResults`

Active bindings

`count` the number of individual results (`Count==1` generally means that we are dealing with an individual simulation results).

`simulation` Reference to the `Simulation` used to calculate or import the results (Read-Only).

`timeValues` Vector of simulated time output values

`allQuantityPaths` List of all paths for which results are defined.

`allIndividualIds` List of Ids of all individuals that have been simulated

Methods

Public methods:

- SimulationResults$new()
- SimulationResults$hasResultsForIndividual()
- SimulationResults$getValuesByPath()
- SimulationResults$resultsForIndividual()
- SimulationResults$print()

`SimulationResults$new()`: Initialize a new instance of the class

Usage:

```
SimulationResults$new(netObject, simulation)
```

Arguments:

`netObject` An `rSharp::NetObject` object.

`simulation` Reference to the simulation object used to calculated the results

Returns: A new `SimulationResults` object.

`SimulationResults$hasResultsForIndividual()`: Returns `TRUE` if results are available for the individual with id `individualId` otherwise `FALSE`

Usage:

```
SimulationResults$hasResultsForIndividual(individualId)
```

Arguments:

`individualId` Id of the individual

`SimulationResults$getValuesByPath()`: Returns `TRUE` if results are available for the individual with id `individualId` otherwise `FALSE`

Usage:

```
SimulationResults$getValuesByPath(path, individualIds, stopIfNotFound = TRUE)
```

Arguments:

`path` Path for which values should be retrieved

`individualIds` One or more individual ids for which values should be returned

`stopIfNotFound` If `TRUE` (default) an error is thrown if no values could be found for the `path`/ If `FALSE`, a list of `NA` values is returned

`SimulationResults$resultsForIndividual()`: Returns all available results for the individual with id `individualId`

Usage:

```
SimulationResults$resultsForIndividual(individualId)
```

Arguments:

`individualId` Id for which the results should be returned

`SimulationResults$print()`: Print the object to the console

Usage:

```
SimulationResults$print(...)
```

Arguments:

`...` Rest arguments.

simulationResultsToDataFrame
Converts a SimulationResults *objects to a data.frame*

Description

Converts a SimulationResults objects to a data.frame

Usage

```
simulationResultsToDataFrame(
  simulationResults,
  quantitiesOrPaths = NULL,
  population = NULL,
  individualIds = NULL
)

simulationResultsToTibble(
  simulationResults,
  quantitiesOrPaths = NULL,
  population = NULL,
  individualIds = NULL
)
```

Arguments

simulationResults	Object of type SimulationResults produced by calling runSimulations on a Simulation object.
quantitiesOrPaths	Quantity instances (element or vector) typically retrieved using getAllQuantitiesMatching or quantity path (element or vector of strings) for which the results are to be returned. (optional) When providing the paths, only absolute full paths are supported (i.e., no matching with '*' possible). If quantitiesOrPaths is NULL (default value), returns the results for all output defined in the results.
population	population used to calculate the simulationResults (optional). This is used only to add the population covariates to the resulting data table.
individualIds	numeric IDs of individuals for which the results should be extracted. By default, all individuals from the results are considered. If the individual with the provided ID is not found, the ID is ignored.

Value

SimulationResults object as data.frame with columns IndividualId, Time, paths, simulationValues, unit, dimension, TimeUnit.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

# Running an individual simulation
# results is an instance of `SimulationResults`
results <- runSimulations(sim)[[1]]

# convert to a dataframe
simulationResultsToDataFrame(results)
```

`SimulationRunOptions` *SimulationRunOptions*

Description

Options to be passed to the simulation engine

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SimulationRunOptions`

Active bindings

`numberOfCores` (Maximal) number of cores to be used. This is only relevant when simulating a population simulation. Default is `getOSPSuiteSetting("numberOfCores")`.

`checkForNegativeValues` Specifies whether negative values check is on or off. Default is `TRUE`

`showProgress` Specifies whether a progress bar should be shown during population simulations. If `TRUE`, a progress bar is shown in the console, indicating the number of already executed simulations from the total population size. The progress bar does not indicate the progress of a single simulation. This option only applies to population simulations and has no effect on individual simulations. Default is `getOSPSuiteSetting("showProgress")`

Methods

Public methods:

- `SimulationRunOptions$new()`
- `SimulationRunOptions$print()`

`SimulationRunOptions$new()`: Initialize a new instance of the class

Usage:

```
SimulationRunOptions$new(
  numberOfCores = NULL,
  checkForNegativeValues = NULL,
  showProgress = NULL
)
```

Arguments:

`numberOfCores` Number of cores to use for the simulation. Default value is `getOSPSuiteSetting("numberOfCores")`

`checkForNegativeValues` Should the solver check for negative values. Default is `TRUE`

`showProgress` Should a progress bar be displayed during population simulations. If `TRUE`, a progress bar is shown in the console, indicating the number of already executed simulations from the total population size. The progress bar does not indicate the progress of a single simulation. This option only applies to population simulations and has no effect on individual simulations. Default value is `getOSPSuiteSetting("showProgress")`

Returns: A new `SimulationRunOptions` object.

`SimulationRunOptions$print()`**:** Print the object to the console

Usage:

```
SimulationRunOptions$print(...)
```

Arguments:

`...` Rest arguments.

`SimulationSettings` *SimulationSettings*

Description

Settings associated with a given simulation

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SimulationSettings`

Active bindings

`solver` Container containing all solver parameters for the simulation (read-only)

`outputSelections` All selected quantities (species, observers, parameters) that will be part of the simulated results

`outputSchema` OutputSchema object containing the output intervals used to generate simulation data

Methods

Public methods:

- `SimulationSettings$print()`

`SimulationSettings$print()`**:** Print the object to the console

Usage:

```
SimulationSettings$print(...)
```

Arguments:

`...` Rest arguments.

SnapshotParameter *SnapshotParameter*

Description

A parameter typically used in the definition of IndividualCharacteristics covariates (Height, Weight etc...)

Super classes

rSharp::NetObject -> DotNetWrapper -> SnapshotParameter

Active bindings

value Parameter value

unit Unit in which the value is defined

Methods

Public methods:

- SnapshotParameter$new()
- SnapshotParameter$print()
- SnapshotParameter$printValue()
- SnapshotParameter$getPrintValue()

SnapshotParameter$new(): Initialize a new instance of the class

Usage:

```
SnapshotParameter$new(netObject = NULL, value = NULL, unit = NULL)
```

Arguments:

netObject Optional NetObject. If not defined, a new instance will be created

value Optional value of the parameter.

unit Optional unit of the value specified.

Returns: A new SnapshotParameter object.

SnapshotParameter$print(): Print the object to the console

Usage:

```
SnapshotParameter$print(...)
```

Arguments:

... Rest arguments.

SnapshotParameter$printValue(): Print the parameter in one line

Usage:

```
SnapshotParameter$printValue(caption)
```

Arguments:

`caption` Caption to display before the value of the parameter

`SnapshotParameter$getPrintValue()`: Return a string for printing the parameter in one line

Usage:

```
SnapshotParameter$getPrintValue()
```

Returns: A string for printing the parameter in one line

`SolverSettings` *SolverSettings*

Description

Solver settings associated with a given simulation

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `SolverSettings`

Active bindings

`useJacobian` Use of Jacobian matrix during calculations

`h0` Initial time step size

`hMin` Minimum absolute value of step size allowed

`hMax` Maximum absolute value of step size allowed

`mxStep` Maximum number of internal steps to be taken by the solver in its attempt to reach tout

`relTol` Relative tolerance of unknowns

`absTol` Absolute tolerance of unknowns

Methods

Public methods:

- `SolverSettings$print()`

`SolverSettings$print()`: Print the object to the console

Usage:

```
SolverSettings$print(...)
```

Arguments:

`...` Rest arguments.

Species *Default species defined in PK-Sim*

Description

Default species defined in PK-Sim

Usage

```
Species
```

splitPopulationFile *Loads a population from the* csvPopulationFile *and split the loaded population according to* numberOfCores.

Description

Loads a population from the csvPopulationFile and split the loaded population according to numberOfCores.

Usage

```
splitPopulationFile(
  csvPopulationFile,
  numberOfCores,
  outputFolder,
  outputFileName
)
```

Arguments

csvPopulationFile	Full path of csv population file to split.
numberOfCores	Number of cores used for parallelization computing. The population will be split across all cores.
outputFolder	Folder where all split files will be created
outputFileName	File names will be constructed using this parameter concatenated with the core index.

Value

A string vector containing the full path of the population files created. Note that there might be less files than cores

Examples

```
csvPath <- system.file("extdata", "pop.csv", package = "ospsuite")

# Split the population in up to 3 files, saved in the temp folder
splitFiles <- splitPopulationFile(csvPath, 3, tempdir(), "PopFile")
```

StandardContainer *Standard containers typically available in a PBPK simulation*

Description

Standard containers typically available in a PBPK simulation

Usage

```
StandardContainer
```

StandardOntogeny *List of ontogeny supported in PK-Sim*

Description

List of ontogeny supported in PK-Sim

Usage

```
StandardOntogeny
```

StandardPath *Standard parameter paths typically available in a PBPK simulation*

Description

Standard parameter paths typically available in a PBPK simulation

Usage

```
StandardPath
```

`StandardPKParameter` *Standard PK-Parameters types defined in OSPSuite This is only used to defined how a user defined PK Parameter should be calculated*

Description

Standard PK-Parameters types defined in OSPSuite This is only used to defined how a user defined PK Parameter should be calculated

Usage

```
StandardPKParameter
```

`toBaseUnit` *Converts a value given in a specified unit into the base unit of a quantity*

Description

Converts a value given in a specified unit into the base unit of a quantity

Usage

```
toBaseUnit(
  quantityOrDimension,
  values,
  unit,
  molWeight = NULL,
  molWeightUnit = NULL
)
```

Arguments

`quantityOrDimension`	Instance of a quantity from which the dimension will be retrieved or name of dimension
`values`	Value in unit (single or vector)
`unit`	Unit of value
`molWeight`	Optional molecule weight to use when converting, for example, from molar to mass amount or concentration. If `molWeightUnit` is not specified, `molWeight` is assumed to be in kg/\u00b5mol
`molWeightUnit`	Unit of the molecular weight value. If `NULL` (default), kg/\u00b5mol is assumed.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
par <- getParameter("Organism|Liver|Volume", sim)

# Converts the value in unit (1000 ml) to the base unit (l) => 1
valueInBaseUnit <- toBaseUnit(par, 1000, "ml")

valuesInBaseUnit <- toBaseUnit(par, c(1000, 2000, 3000), "ml")
```

`toDisplayUnit` *Convert base unit to display unit*

Description

Converts a value given in base unit of a quantity into the display unit of a quantity

Usage

```
toDisplayUnit(quantity, values, unit = NULL)
```

Arguments

`quantity`	Instance of a quantity from which the base unit will be retrieved
`values`	Value (single or vector). If `unit` is not specified, values are assumed to be in base unit
`unit`	Optional Name of the unit to convert from. If `NULL` (default), the values are assumed to be in base unit.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
par <- getParameter("Organism|Liver|Volume", sim)

# Converts the value in base unit (1L) to display unit
valueInMl <- toDisplayUnit(par, 1)

valuesInDisplayUnit <- toDisplayUnit(par, c(1, 5, 5))

# Converts the value in ml to display unit
valueInDisplayUnit <- toDisplayUnit(par, 1000, "ml")
```

toPathArray *Convert a path defined as string to a path array*

Description

Convert a path defined as string to a path array

Usage

```
toPathArray(path)
```

Arguments

path	A string representation of a path, with path entries separated by 'I'

Value

An array containing one element for each path entry.

Examples

```
toPathArray("Organism|Organ|Liver")
```

toPathString *Convert a path array to a path as string with entries separated by 'I'*

Description

Convert a path array to a path as string with entries separated by 'I'

Usage

```
toPathString(...)
```

Arguments

...	Path entries to concatenate into a path string.

Value

A string built using each entry of the pathArray.

Examples

```
toPathString(c("Organism", "Organ", "Liver"))
```

toUnit *Converts values from one unit to another*

Description

Converts values from one unit to another

Usage

```
toUnit(
  quantityOrDimension,
  values,
  targetUnit,
  sourceUnit = NULL,
  molWeight = NULL,
  molWeightUnit = NULL
)
```

Arguments

quantityOrDimension	Instance of a quantity from which the dimension will be retrieved or name of dimension
values	Values to convert (single or vector). If `sourceUnit` is not specified, `values` are in the base unit of the dimension
targetUnit	Unit to convert to
sourceUnit	Optional Name of the unit to convert from. If `NULL` (default), the values are assumed to be in base unit.
molWeight	Optional molecular weight to use when converting, for example, from molar to mass amount or concentration. If `molWeightUnit` is not specified, `molWeight` is assumed to be in kg/\u00b5mol
molWeightUnit	Optional Unit of the molecular weight value. If `NULL` (default), kg/\u00b5mol is assumed.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
par <- getParameter("Organism|Liver|Volume", sim)

# Converts the value in base unit (1L) to ml => 1000
valueInMl <- toUnit(par, 1, "ml")

valuesInMl <- toUnit(par, c(1, 5, 5), "ml")

# Converts a numerical value in from mmol/l to mg/dl
valuesInMgDl <- toUnit(ospDimensions$`Concentration (molar)`, 5,
```

```
  targetUnit = "mmol/l",
  sourceUnit = "mg/dl", molWeight = 180, molWeightUnit = "g/mol"
)
```

`uniqueEntities` *Extract Unique Elements of type 'Entity'*

Description

Extract Unique Elements of type 'Entity'

Usage

```
uniqueEntities(entities, compareBy = CompareBy$id)
```

Arguments

`entities`	List of objects of type 'Entity'
`compareBy`	A string defining the property that is compared by. Can take values 'id', 'name', and 'path'. Default is 'id'.

Value

List of entities that are unique for the property defined by the argument 'compareBy'.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)

parameters <- c(
  getParameter(toPathString(c("Organism", "Liver", "Volume")), sim),
  getParameter(toPathString(c("Organism", "Liver", "Volume")), sim),
  getParameter(toPathString(c("Organism", "TableParameter")), sim)
)

# Return a list containing the two parameters 'Volume' and 'Weight (tissue)'
uniqueEntities(parameters, CompareBy$id)
```

`updatePKParameter` *Updates some properties of a PK-Parameter (displayName and displayUnit)*

Description

Updates some properties of a PK-Parameter (displayName and displayUnit)

Usage

```
updatePKParameter(name, displayName = NULL, displayUnit = NULL)
```

Arguments

`name`	Name of PK-Parameter to update
`displayName`	Optional display name
`displayUnit`	Optional display unit. Note that the unit should be defined in unit of the dimension

Examples

```
updatePKParameter("t_max", "MyTmax", "min")
```

`UserDefinedPKParameter` *UserDefinedPKParameter*

Description

Definition of a user defined PKParameter that can be calculated on top of the standard PK Parameters

Super classes

`rSharp::NetObject` -> `DotNetWrapper` -> `PKParameter` -> `UserDefinedPKParameter`

Active bindings

`startTime` Start time in minutes for the calculation of the PK-Parameter. If not specified, the time will start at the first time point of the simulation (optional)

`startTimeOffset` Offset in minutes to apply to the start time or to the start time of the application identified by `startApplicationIndex`. (0 by default).

`endTime` End time in minutes for the calculation of the PK-Parameter. If not specified, the time will end at the last time point of the simulation (optional)

`endTimeOffset` Offset in minutes to apply to the end time or to the start time of the application identified by `endApplicationIndex`. (0 by default).

`startApplicationIndex` 1-based Index of the application to use to determine the start time for the calculation of the PK-Parameter. If not specified, the time will start at the first time point of the simulation (optional)

`endApplicationIndex` 1-based Index of the application to use to determine the end time for the calculation of the PK-Parameter. If not specified, the time will end at the last time point of the simulation (optional)

`normalizationFactor` Factor to use to normalized the calculated PK-Parameter. (typically Drug-Mass, Dose, DosePerBodyWeight). It is the responsibility of the caller to ensure that the value is in the correct unit. (optional)

`concentrationThreshold` Used in conjunction with the `threshold` parameter type. If defined, the time at which this concentration was reached will be calculated

`standardPKParameter` Based parameter to use to perform the PK-Analysis calculation. See `StandardPKParameter` enum for all possible pk parameters

Methods

Public methods:

- `UserDefinedPKParameter$print()`

`UserDefinedPKParameter$print()`: Print the object to the console

Usage:

```
UserDefinedPKParameter$print(...)
```

Arguments:

`...` Rest arguments.

`validateDimension` *Validate dimension*

Description

Validate dimension

Usage

```
validateDimension(dimension)
```

Arguments

`dimension`	String name of the dimension.

Details

Check if the provided dimension is supported. If not, throw an error

validateIsNamedList *Validate that an object is a named list*

Description

Validate that an object is a named list

Usage

```
validateIsNamedList(x, varName)
```

Arguments

x	Object to validate as a named list.
varName	Name of the variable being validated (used in error message).

Value

invisible TRUE if the validation passed, otherwise an error is thrown.

Examples

```
validateIsNamedList(list(a = 1, b = 2), "myVar") # passes
# validateIsNamedList(list(1, 2), "myVar") # throws an error
```

validateUnit *Validate unit*

Description

Validate unit

Usage

```
validateUnit(unit, dimension)
```

Arguments

unit	String name of the unit
dimension	String name of the dimension.

Details

Check if the unit is valid for the dimension. If not, throw an error

ValuePoint *ValuePoint*

Description

An entry (x, y) in a table formula

Super classes

rSharp::NetObject -> DotNetWrapper -> ValuePoint

Active bindings

x The x value of the point.

y the y value of the point.

restartSolver Indicates whether the solver should be restarted when this point is reached. Default is FALSE

Methods

Public methods:

- ValuePoint$print()

ValuePoint$print(): Print the object to the console

Usage:

```
ValuePoint$print(...)
```

Arguments:

... Rest arguments.

Index