Package: esqlabsR (via r-universe)

July 15, 2026

Type Package

Title ESQlabs utilities package

Version 5.7.0.9005

Maintainer Pavel Balazki <pavel.balazki@esqlabs.com>

Description The `{esqlabsR}` package facilitates and standardizes the modeling and simulation of physiologically based kinetic (PBK) and quantitative systems pharmacology/toxicology (QSP/T) models implemented in the [Open Systems Pharmacology Software](https://www.open-systems-pharmacology.org/) (OSPS).

License GPL-2

URL https://github.com/esqLABS/esqlabsR, https://esqlabs.github.io/esqlabsR/

BugReports https://github.com/esqLABS/esqlabsR/issues

Depends ospsuite (>= 12.4.2), R (>= 4.4)

Imports colorspace, cli, dplyr, fs, ggplot2, grDevices, jsonlite, lifecycle, methods, ospsuite.parameteridentification, parallel, patchwork, purrr, R6 (>= 2.5.0), readxl, scales, stringr, tidyr, tlf (>= 1.6.0), ospsuite.utils (>= 1.10.0), tools, usethis, vctrs, writexl

Suggests clipr, covr, devtools, knitr, openxlsx, quarto, reactable, rmarkdown, rlang, showtext, testthat (>= 3.0.0), vdiffr (>= 1.0.0), withr, emoji

VignetteBuilder knitr, quarto

Config/testthat/edition 3

Config/testthat/parallel true

Config/rcmdcheck/ignore-inconsequential-notes true

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

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

Config/roxygen2/version 8.0.0

Repository https://esqlabs.r-universe.dev

Date/Publication 2026-07-15 10:18:28 UTC

RemoteUrl https://github.com/esqLABS/esqlabsR

RemoteRef additional-repos-osp-universe

RemoteSha 3a5c2c6f5a9ad2d5c570de07bcceda073d0aa7a3

Contents

`addScenarioConfigurationsToExcel` *Add scenario configurations to project Excel files*

Description

Adds scenario configurations to the project's Scenarios.xlsx file and ensures that required application protocol sheets exist in the Applications.xlsx file. This function handles the Excel file operations for adding scenarios to a project.

Usage

```
addScenarioConfigurationsToExcel(
  scenarioConfigurations,
  projectConfiguration,
  appendToExisting = TRUE
)
```

Arguments

`scenarioConfigurations`
: Named list of `ScenarioConfiguration` objects to add to the project.

`projectConfiguration`
: A `ProjectConfiguration` object holding base information.

`appendToExisting`
: Logical. Whether to append new scenarios to existing ones in the scenarios file. If `FALSE`, the ENTIRE scenarios file will be overwritten with only the new scenarios. If `TRUE` (default), new scenarios will be added to existing ones.

Details

This function performs the following operations:

- Checks for duplicate scenario names if `appendToExisting` is `TRUE`.
- Creates missing application protocol sheets in Applications.xlsx by extracting parameters from PKML files (both Events and Applications parameters).
- Writes scenario configurations to the Scenarios.xlsx file with proper structure.
- Manages output paths and their IDs in the OutputPaths sheet.

The function ensures that the Excel files are properly structured with the following sheets:

- **Scenarios sheet**: Contains scenario definitions with columns for scenario name, individual/population IDs, parameter sheets, application protocol, simulation time, steady state settings, model file, and output path IDs.
- **OutputPaths sheet**: Contains output path IDs and their corresponding paths. When named vectors are used for `outputPaths` in scenario configurations, the names will be used as OutputPathId values.

- **Applications.xlsx**: Contains application protocol sheets with parameter definitions.

The function handles parameter extraction from PKML files, excluding default parameters like "Volume" and "Application rate", and only includes constant parameters.

Value

Invisibly returns the names of the added scenarios.

Examples

```
## Not run:
# Create scenario configurations from PKML
scenarios <- createScenarioConfigurationsFromPKML(
  pkmlFilePaths = "path/to/simulation.pkml",
  projectConfiguration = projectConfiguration
)

# Add scenarios to project Excel files
addScenarioConfigurationsToExcel(
  scenarioConfigurations = scenarios,
  projectConfiguration = projectConfiguration,
  appendToExisting = TRUE
)

## End(Not run)
```

`applyIndividualParameters` *Apply an individual to the simulation. For human species, only parameters that do not override formulas are applied. For other species, all parameters returned by* `createIndividual` *are applied.*

Description

Apply an individual to the simulation. For human species, only parameters that do not override formulas are applied. For other species, all parameters returned by `createIndividual` are applied.

Usage

```
applyIndividualParameters(individualCharacteristics, simulation)
```

Arguments

`individualCharacteristics`	`IndividualCharacteristics` describing an individual. Optional
`simulation`	`Simulation` loaded from the PKML file

Examples

```
## Not run:
simulation <- loadSimulation(filePath = modelPath)
humanIndividualCharacteristics <- createIndividualCharacteristics(
  species = Species$Human, population = HumanPopulation$European_ICRP_2002,
  gender = Gender$Male, weight = 70
)
applyIndividualParameters(humanIndividualCharacteristics, simulation)

## End(Not run)
```

`calculateMeanDataSet` *Calculate mean and standard deviation for the yValues of the given* `DataSet` *objects*

Description

Calculate mean and standard deviation for the yValues of the given `DataSet` objects

Usage

```
calculateMeanDataSet(
  dataSets,
  method = "arithmetic",
  lloqMode = LLOQMode$`LLOQ/2`,
  outputXunit = NULL,
  outputYunit = NULL,
  outputMolWeight = NULL
)
```

Arguments

`dataSets`	list of `DataSet` objects
`method`	method for calculating the mean and standard deviation - either `arithmetic` (default) or `geometric`
`lloqMode`	how to treat data points below LLOQ if LLOQ is given - `LLOQ/2` (default): use as given (since `DataSet` stores values below LLOQ as `LLOQ/2`), `LLOQ`: set value to LLOQ value, `ZERO`: set value to 0, `ignore`: do not use data points for mean calculation
`outputXunit`	xUnit of output data set, if `NULL` (default) xUnit of the first data set will be used
`outputYunit`	yUnit of output data set, if `NULL` (default) yUnit of the first data set will be used
`outputMolWeight`	molWeight of output data set in `g/mol` - obligatory when initial data sets have differing molWeight values

Details

Calculates mean and standard deviation of the yValues of the given `DataSet` objects per xValue. The meta data of the returned `DataSet` consists of all meta data that are equal in all initial data sets. Its LLOQ is the mean LLOQ value of all data sets which have an LLOQ set, e.g. if dataSet1 has LLOQ 1, dataSet2 has LLOQ 3 and dataSet3 has no LLOQ, then 2 is used for the returned `DataSet`. The LLOQ of the returned `DataSet` is the arithmetic mean of LLOQ values of all `DataSets`

Value

A single `DataSet` object

`col2hsv` *Returns the HSV values for a given R color name*

Description

Returns the HSV values for a given R color name

Usage

```
col2hsv(color)
```

Arguments

`color`	vector of any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa" (see rgb), or a positive integer `i` meaning `palette()[i]`.

Value

A matrix with a column for each color. The three rows of the matrix indicate hue, saturation and value and are named "h", "s", and "v" accordingly.

Examples

```
col2hsv("yellow")
```

compareSimulations *Compare two simulations*

Description

Compare two simulations

Usage

```
compareSimulations(simulation1, simulation2, compareFormulasByValue = FALSE)
```

Arguments

simulation1	First Simulation to compare
simulation2	Second Simulation to compare
compareFormulasByValue	If FALSE (default), parameters are considered not equal if the have the same value but different formulas (e.g., a constant vs. explicit formula). If TRUE, only values are compared.

Details

The function compares two simulations and returns a list of entities that differ:

- Parameters: a named list with a list of all Parameter entities that are:
- in simulation1 but not in simulation 2 (In1NotIn2)
- in simulation 2 but not in simulation 1 (I21NotIn1)
- a list Different with all parameters which values differ between the simulations. Two parameters are considered different if their formulas or values differ.

Value

Named list with following levels:

- Parameters with named lists In1NotIn2, In2NotIn1, and Different, holding the Parameter objects that are present in the first but not in the second simulation, present in the second but not in the first simulation, and present in both simulations but with different formulas and/or values, respectively.

See Also

isParametersEqual

Examples

```
## Not run:
humanSim <- loadSimulation(file.path(modelFolder, "DefaultHuman.pkml"))
ratSim <- loadSimulation(file.path(modelFolder, "DefaultRat.pkml"))
diffParams <- compareSimulationParameters(humanSim, ratSim)

## End(Not run)
```

`compareWithNA` *Compare values including* `NA`

Description

Compare values including `NA`

Usage

```
compareWithNA(v1, v2)
```

Arguments

`v1`	Value or a list of values to compare. May include `NA`.
`v2`	Value or a list of values to compare. May include `NA`.

Details

From http://www.cookbook-r.com/Manipulating_data/Comparing_vectors_or_factors_with_NA/

Value

`TRUE` wherever elements are the same, including `NA`'s,

`createDataCombinedFromExcel`
Generate DataCombined objects as defined in excel file

Description

Generate DataCombined objects as defined in excel file

Usage

```
createDataCombinedFromExcel(
  projectConfiguration,
  dataCombinedNames = NULL,
  plotGridNames = NULL,
  simulatedScenarios = NULL,
  observedData = NULL,
  stopIfNotFound = TRUE
)
```

Arguments

projectConfiguration	Object of class ProjectConfiguration that contains information about the output paths and the excel file where plots are defined.
dataCombinedNames	Names of the DataCombined objects that will be created. If a DataCombined with a given name is not defined in the Excel file, an error is thrown. Can be used together with plotGridNames.
plotGridNames	Names of the plot grid specified in the sheet plotGrids. Each data combined used by the specified plot grids will be created. Can be used together with dataCombinedNames.
simulatedScenarios	A list of simulated scenarios as returned by runScenarios()
observedData	A list of DataSet objects
stopIfNotFound	If TRUE (default), the function stops if any of the simulated results or observed data are not found. If FALSE a warning is printed.

Value

A list of DataCombined objects, or an empty list if both dataCombinedNames and plotGridNames are NULL or stopIfNotFound = TRUE and the specified DataCombined could not be created.

---

createDefaultProjectConfiguration

Create a default ProjectConfiguration

Description

Create a ProjectConfiguration based on the "ProjectConfiguration.xlsx"

Usage

```
createDefaultProjectConfiguration(
  path = file.path("ProjectConfiguration.xlsx"),
  ignoreVersionCheck = FALSE
)
```

Arguments

path	path to the ProjectConfiguration.xlsx file. Defaults to the ProjectConfiguration.xlsx file in the working directory.
ignoreVersionCheck	If TRUE, skip the version mismatch check when loading the configuration file. Use this in non-interactive contexts such as automated tests or scripts running from console where interactive user input cannot be assured. Defaults to FALSE.

Value

Object of type ProjectConfiguration

---

createEsqlabsExportConfiguration

Create an instance of ExportConfiguration *R6 class*

Description

An instance of ExportConfiguration R6 class from {tlf} package is needed for saving the plots and plot grids created using the {ospsuite} package.

The default attributes of the class are chosen to reflect the corporate standards adopted by esqLABS GmbH.

Usage

```
createEsqlabsExportConfiguration(outputFolder)
```

Arguments

outputFolder	Path to the folder where the results will be stored.

Value

An instance of ExportConfiguration R6 class.

See Also

Other create-plotting-configurations: createEsqlabsPlotConfiguration(), createEsqlabsPlotGridConfiguration()

Examples

```
myProjConfig <- ProjectConfiguration$new()
createEsqlabsExportConfiguration(myProjConfig$outputFolder)
```

createEsqlabsPlotConfiguration

Create an instance of `DefaultPlotConfiguration` *R6 class*

Description

An instance of `DefaultPlotConfiguration` R6 class from `{tlf}` package is needed for creating visualizations with the `{ospsuite}` package.

The default attributes of the class are chosen to reflect the corporate standards adopted by esqLABS GmbH.

Usage

```
createEsqlabsPlotConfiguration()
```

Value

An instance of `DefaultPlotConfiguration` R6 class.

See Also

Other create-plotting-configurations: `createEsqlabsExportConfiguration()`, `createEsqlabsPlotGridConfiguration(`

Examples

```
createEsqlabsPlotConfiguration()
```

createEsqlabsPlotGridConfiguration

Create an instance of `PlotGridConfiguration` *R6 class*

Description

An instance of `PlotGridConfiguration` R6 class from `{tlf}` package is needed for creating a grid of multiple visualizations created using the `{ospsuite}` package.

The default attributes of the class are chosen to reflect the corporate standards adopted by esqLABS GmbH.

Usage

```
createEsqlabsPlotGridConfiguration()
```

Value

An instance of `PlotGridConfiguration` R6 class.

See Also

Other create-plotting-configurations: `createEsqlabsExportConfiguration()`, `createEsqlabsPlotConfiguration()`

Examples

```
createEsqlabsPlotGridConfiguration()
```

`createPITasks` *Create Parameter Identification tasks*

Description

Creates ParameterIdentification objects from PI configurations. Each PITaskConfiguration produces one ParameterIdentification object.

Usage

```
createPITasks(
  piTaskConfigurations,
  observedData = NULL,
  stopIfParameterNotFound = TRUE
)
```

Arguments

`piTaskConfigurations`	Named list of `PITaskConfiguration` objects
`observedData`	Named list of `DataSet` objects. When provided, datasets are looked up by the name in the `DataSet` column of `PIOutputMappings`. When `NULL` (default), observed data is loaded internally using the `ObservedDataSheet` column.
`stopIfParameterNotFound`	Logical. If `TRUE` (default), an error is thrown when a parameter defined in the scenario configuration cannot be found in the simulation. Set to `FALSE` to continue silently when scenario parameters are absent from the model.

Value

Named list of `ParameterIdentification` objects

Examples

```
## Not run:
projectConfiguration <- createProjectConfiguration(
  exampleProjectConfigurationPath(),
  ignoreVersionCheck = TRUE
)
```

```
piTaskConfigurations <- readPITaskConfigurationFromExcel(
  projectConfiguration = projectConfiguration
)
piTasks <- createPITasks(piTaskConfigurations)

## End(Not run)
```

---

createPlotsFromExcel *Generate plots as defined in excel file* `projectConfiguration$plotsFile`

Description

Generate plots as defined in excel file `projectConfiguration$plotsFile`

Usage

```
createPlotsFromExcel(
  plotGridNames = NULL,
  simulatedScenarios = NULL,
  observedData = NULL,
  dataCombinedList = NULL,
  projectConfiguration,
  outputFolder = NULL,
  stopIfNotFound = TRUE
)
```

Arguments

`plotGridNames`	Names of the plot grid specified in the sheet `plotGrids` for which the figures will be created. If `NULL` (default), all plot grids specified in the excel sheet will be created. If a plot grid with a given name does not exist, an error is thrown.
`simulatedScenarios`	A list of simulated scenarios as returned by `runScenarios()`. Can be `NULL` if no simulated data is required for the plots.
`observedData`	A list of `DataSet` objects. Can be `NULL` if no observed data is required for the plots.
`dataCombinedList`	A (named) list of `DataCombined` objects as input to create plots defined in the `plotGridNames` argument. Missing `DataCombined` will be created from the Excel file (default behavior). Defaults to `NULL`, in which case all `DataCombined` are created from Excel.
`projectConfiguration`	Object of class `ProjectConfiguration` that contains information about the output paths and the excel file where plots are defined.

outputFolder	Optional - path to the folder where the results will be stored. If NULL (default), projectConfiguration$outputFolder is used. Only relevant for plots specified for export in the exportConfiguration sheet.
stopIfNotFound	If TRUE (default), the function stops if any of the simulated results or observed data are not found. If FALSE a warning is printed.

Value

A list of ggplot objects

---

createProjectConfiguration

Create a ProjectConfiguration

Description

Create a ProjectConfiguration based on the "ProjectConfiguration.xlsx"

Usage

```
createProjectConfiguration(
  path = file.path("ProjectConfiguration.xlsx"),
  ignoreVersionCheck = FALSE
)
```

Arguments

path	path to the ProjectConfiguration.xlsx file. default to the ProjectConfiguration.xlsx file located in the working directory.
ignoreVersionCheck	If TRUE, skip the version mismatch check when loading the configuration file. Use this in non-interactive contexts such as automated tests or scripts running from console where interactive user input cannot be assured. Defaults to FALSE.

Value

Object of type ProjectConfiguration

createScenarioConfigurationsFromPKML
Create scenario configurations from PKML files

Description

Creates scenario configurations from PKML files by extracting available information such as applications, output paths, and simulation time settings. This function creates scenario configuration objects that can be used with the esqlabsR workflow.

Usage

```
createScenarioConfigurationsFromPKML(
  pkmlFilePaths,
  projectConfiguration,
  scenarioNames = NULL,
  individualId = NULL,
  populationId = NULL,
  applicationProtocols = NULL,
  paramSheets = NULL,
  outputPaths = NULL,
  simulationTime = NULL,
  simulationTimeUnit = NULL,
  steadyState = FALSE,
  steadyStateTime = NULL,
  steadyStateTimeUnit = NULL,
  readPopulationFromCSV = FALSE
)
```

Arguments

pkmlFilePaths	Character vector of paths to PKML files to create scenarios from. Can be a single string (recycled for all scenarios) or a vector with the same length as the number of scenarios being created (determined by the longest vector argument).
projectConfiguration	A ProjectConfiguration object holding base information.
scenarioNames	Character vector. Optional custom names for the scenarios. If NULL (default), scenario names will be extracted from the simulation names in the PKML files. If provided, must have the same length as pkmlFilePaths.
individualId	Character vector. Optional individual IDs to use for scenarios. If NULL (default), no individual will be specified. Can be a single string (recycled for all scenarios) or a vector with the same length as pkmlFilePaths.
populationId	Character vector. Optional population IDs to use for scenarios. If NULL (default), no population will be specified. If provided, sets simulation type to "Population". Can be a single string (recycled for all scenarios) or a vector with the same length as pkmlFilePaths.

`applicationProtocols`
: Character vector. Optional application protocol names to use for scenarios. If `NULL` (default), application protocols will be set to the scenario name. Can be a single string (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`.

`paramSheets`
: Character vector. Optional parameter sheet names to apply to scenarios. If `NULL` (default), no parameter sheets will be applied. Can be a single string (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`. If providing multiple sheets per scenario, separate them with commas in the string.

`outputPaths`
: Character vector or named vector. Optional output paths to use for scenarios. If `NULL` (default), output paths will be extracted from the PKML files' output selections. Can be a single string (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`. If providing multiple paths per scenario, separate them with commas in the string. Named vectors are supported where names serve as aliases for the paths, e.g., c("plasma" = "Organism|VenousBlood|Plasma|AKB-9090|Concentration in container").

`simulationTime`
: Character vector. Optional simulation time to use for scenarios as character strings containing one or multiple time intervals separated by a ';'. Each time interval is a triplet of values <StartTime, EndTime, Resolution>, where `Resolution` is the number of simulated points per time unit defined in the `simulationTimeUnit`. If `NULL` (default), simulation time will be extracted from the PKML files' output schema intervals. Can be a single string (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`.

`simulationTimeUnit`
: Character vector. Optional simulation time units. Only used when `simulationTime` is provided. If `NULL` (default), will be extracted from the PKML file's output schema intervals, or set to "min" (minutes) if not available. Can be a single string (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`.

`steadyState`
: Logical vector. Whether to simulate steady-state for each scenario. Default is `FALSE`. Can be a single logical value (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`.

`steadyStateTime`
: Numeric vector. Steady-state times. Only used when corresponding `steadyState` is `TRUE`. If `NULL` (default), no steady-state time will be set. Can be a single numeric value (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`.

`steadyStateTimeUnit`
: Character vector. Steady-state time units. Only used when `steadyState = TRUE` and `steadyStateTime` is provided. If `NULL` (default), "min" will be used. Can be a single string (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`.

`readPopulationFromCSV`
: Logical vector. Whether to read population from CSV for each scenario. Default is `FALSE`. Can be a single logical value (recycled for all scenarios) or a vector with the same length as `pkmlFilePaths`.

Details

This function extracts the following information from PKML files:

- **Applications**: Application protocol names (defaults to scenario name).
- **Output paths**: All selected outputs for the simulation from `outputSelections$allOutputs`.
- **Simulation time**: Time intervals with start time, end time, and resolution from `outputSchema$intervals`.
- **Simulation time unit**: Time unit from the output schema intervals (e.g., "h" for hours).

Vector Arguments and Recycling:

All arguments support vectorization to create scenarios with different parameter values:

- **Length 1**: The value is recycled (applied to all scenarios).
- **Length > 1**: All vector arguments must have the same length, which determines the number of scenarios.
- **Mixed lengths**: An error is thrown if vector arguments have inconsistent lengths.

The number of scenarios created is determined by the longest vector argument. All shorter vectors (including `pkmlFilePaths`) are recycled to match this length.

This allows you to efficiently create multiple scenarios in several ways:

- **Same PKML, different settings**: Use a single PKML file with vectors of different parameter values.
- **Different PKMLs, same settings**: Use multiple PKML files with single parameter values.
- **Different PKMLs, different settings**: Use vectors of both PKML files and parameter values.

The function handles duplicate scenario names by appending indices (e.g., "Scenario_1", "Scenario_2"). It creates scenario configurations but does not write them to Excel files. Use `addScenarioConfigurationsToEx` to add the scenarios to the project's Excel files.

Value

A named list of `ScenarioConfiguration` objects with the names being the scenario names.

Examples

```
## Not run:
# Create default project configuration
projectConfiguration <- createDefaultProjectConfiguration()

# Create scenarios from a single PKML file
pkmlPath <- "path/to/simulation.pkml"
scenarios <- createScenarioConfigurationsFromPKML(
  pkmlFilePaths = pkmlPath,
  projectConfiguration = projectConfiguration
)

# Add scenarios to Excel configuration
addScenarioConfigurationsToExcel(
  scenarioConfigurations = scenarios,
  projectConfiguration = projectConfiguration
)
```

```
# Create scenarios from multiple PKML files with custom names
pkmlPaths <- c("path/to/sim1.pkml", "path/to/sim2.pkml")
scenarios <- createScenarioConfigurationsFromPKML(
  pkmlFilePaths = pkmlPaths,
  projectConfiguration = projectConfiguration,
  scenarioNames = c("Scenario1", "Scenario2")
)

# Add multiple scenarios to configuration
addScenarioConfigurationsToExcel(
  scenarioConfigurations = scenarios,
  projectConfiguration = projectConfiguration
)

# Example of vector recycling - single value applied to all scenarios
scenarios <- createScenarioConfigurationsFromPKML(
  pkmlFilePaths = c("sim1.pkml", "sim2.pkml", "sim3.pkml"),
  projectConfiguration = projectConfiguration,
  individualId = "Individual_001", # Recycled to all scenarios
  steadyState = TRUE,              # Recycled to all scenarios
  steadyStateTime = 1000           # Recycled to all scenarios
)

# Example of vector arguments - different values per scenario
scenarios <- createScenarioConfigurationsFromPKML(
  pkmlFilePaths = c("pediatric.pkml", "adult.pkml", "elderly.pkml"),
  projectConfiguration = projectConfiguration,
  scenarioNames = c("Pediatric", "Adult", "Elderly"),
  individualId = c("Child_001", "Adult_001", "Elderly_001"),
  applicationProtocols = c("Pediatric_Dose", "Standard_Dose", "Reduced_Dose"),
  steadyState = c(FALSE, TRUE, TRUE),
  steadyStateTime = c(NA, 2000, 1500)
)

# Example of PKML recycling - same model, different settings
scenarios <- createScenarioConfigurationsFromPKML(
  pkmlFilePaths = "base_model.pkml",                    # Single PKML recycled
  projectConfiguration = projectConfiguration,
  scenarioNames = c("LowDose", "MediumDose", "HighDose"),
  individualId = c("Patient1", "Patient2", "Patient3"),
  applicationProtocols = c("Low_Protocol", "Med_Protocol", "High_Protocol"),
  steadyState = c(FALSE, TRUE, TRUE)                  # Different settings per scenario
)

## End(Not run)
```

---

createScenarios *Create* Scenario *objects from* ScenarioConfiguration *objects*

Description

Load simulation. Apply parameters from global XLS. Apply individual physiology. Apply individual model parameters. Set simulation outputs. Set simulation time. initializeSimulation(). Create population

Usage

```
createScenarios(
  scenarioConfigurations,
  customParams = NULL,
  stopIfParameterNotFound = TRUE
)
```

Arguments

`scenarioConfigurations`	List of `ScenarioConfiguration` objects to be simulated. See `createScenarios()` for details.
`customParams`	A list containing vectors 'paths' with the full paths to the parameters, 'values' the values of the parameters, and 'units' with the units the values are in. The values will be applied to all scenarios.
`stopIfParameterNotFound`	Boolean. If `TRUE` (default) and a custom parameter is not found, an error is thrown. If `FALSE`, non-existing parameters are ignored.

Value

Named list of `Scenario` objects.

`Distributions` *Supported distributions for sampling*

Description

Supported distributions for sampling

Usage

```
Distributions
```

enumPutList *Add a new key-value pairs to an enum, where the value is a list.*

Description

Add a new key-value pairs to an enum, where the value is a list.

Usage

```
enumPutList(key, values, enum, overwrite = FALSE)
```

Arguments

key	Key to be added
values	Values to be added
enum	enum the key-value pairs should be added to. **WARNING**: the original object is not modified!
overwrite	if TRUE and a key exists, it will be overwritten with the new value. Otherwise, an error is thrown. Default is FALSE.

Value

Enum with added key-value pair.

Examples

```
library(ospsuite.utils)
myEnum <- enum(c(a = "b"))
myEnum <- enumPut("c", "d", myEnum)
myEnum <- enumPut(c("c", "d", "g"), list(12, 2, "a"), myEnum, overwrite = TRUE)
myEnum <- enumPutList("g", list(12, 2, "a"), myEnum, overwrite = TRUE)
```

esqlabsColors *esqLABS color palette*

Description

Returns the list of colors extrapolated between the esqLABS colors blue, red, and green.

Usage

```
esqlabsColors(nrOfColors)
```

Arguments

nrOfColors	Positive integer defining the number of colors to be generated.

Details

For `nrOfColors == 1`, the esqLABS-blue is returned For `nrOfColors == 2`, the esqLABS-blue and green are returned For `nrOfColors == 3`, the esqLABS-blue, red, and green are returned For `nrOfColors > 3`, the three esqLABS colors are fixed, and the remaining colors are extrapolated from blue to red to green. If `nrOfColors` is uneven, the blue-to-red section becomes one color more than the red-to-green section. In this implementation, blue-to-green is not considered.

Value

A list of colors as HEX values.

---

esqlabsRSettingNames *Names of the settings stored in esqlabsEnv Can be used with* `getEsqlabsRSetting()`

Description

Names of the settings stored in esqlabsEnv Can be used with `getEsqlabsRSetting()`

Usage

```
esqlabsRSettingNames
```

---

exampleProjectConfigurationPath *Get the path to example ProjectConfiguration.xlsx*

Description

Get the path to example ProjectConfiguration.xlsx

Usage

```
exampleProjectConfigurationPath()
```

Value

A string representing the path to the example ProjectConfiguration.xlsx file

Examples

```
exampleProjectConfigurationPath()
```

`executeInParallel` *Parallelize the execution of a function over a list of arguments values*

Description

Parallelize the execution of a function over a list of arguments values

Usage

```
executeInParallel(
  fun,
  firstArguments,
  exports = NULL,
  ...,
  outputNames = NULL,
  nrOfCores = ospsuite::getOSPSuiteSetting("numberOfCores")
)
```

Arguments

`fun`	A function that will be called with different arguments values
`firstArguments`	A list of the values of the first argument of the function. The function will be called `n` times where `n` is the number of entries in `firstArguments`
`exports`	Names of the objects in the calling environment that the function relies on that are not passed as arguments. May be `NULL` (default).
`...`	Further arguments of the function.
`outputNames`	Optional: a list of names used for the output list. Result of each execution of `fun` will be named with the name having the same index in `outputNames` as as the argument value in `firstArguments`. If specified, `outputNames` must have the same length as `firstArguments`
`nrOfCores`	Optional: the maximal number of parallel threads. By default the value defined in `ospsuite::getOSPSuiteSetting("numberOfCores")` is used, and equals the number of logical cores minus 1.

Value

A list containing the outputs of the function `fun` iterated over the values in `firstArguments`.

See Also

`parLapply()`

ExportConfiguration *ExportConfiguration*

Description

R6 class extending tlf::ExportConfiguration with row-aware height.

Details

Inherits fields and behavior from tlf::ExportConfiguration.

Inherited fields

See tlf::ExportConfiguration for `name`, `path`, `format`, `width`, `height`, `units`, and `dpi`.

Super class

`tlf::ExportConfiguration` -> `ExportConfiguration`

Active bindings

`heightPerRow` The height of the plot dimensions for a row in a multi panel plot. The final height of the figure will be 'heightPerRow' times the number of rows. If `NULL` (default), value used in `height` is used. If not `NULL`, this value always overrides the `height` property.

Methods

Public methods:

- `ExportConfiguration$savePlot()`
- `ExportConfiguration$clone()`

`ExportConfiguration$savePlot()`: Save/Export a plot

Usage:

```
ExportConfiguration$savePlot(plotObject, fileName = NULL)
```

Arguments:

`plotObject` A `ggplot` object

`fileName` character file name of the exported plot

Returns: The file name of the exported plot

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

exportParametersToXLS *Export simulation parameters to excel*

Description

Creates an excel file with information from the passed parameters. The excel sheet will contain columns "Container Path", "Parameter Name", "Value", and "Units". The resulting file can be loaded in MoBi or in R with the function readParametersFromXLS().

Usage

```
exportParametersToXLS(parameters, paramsXLSpath, sheet = NULL, append = FALSE)
```

Arguments

parameters	A single or a list of Parameter objects
paramsXLSpath	Path to the excel file
sheet	(Optional) name of the excel sheet
append	If TRUE, appends parameters to existing file. If the file exists but the sheet doesn't exist, creates a new sheet. If FALSE (default), overwrites the existing file.

extendParameterStructure *Extend parameters structure with new entries*

Description

Extend parameters structure with new entries

Usage

```
extendParameterStructure(parameters, newParameters)
```

Arguments

parameters	A parameter structure (a list with elements paths, values, and units) or NULL. If NULL, it is treated as an empty parameter structure.
newParameters	A parameter structure (a list with elements paths, values, and units) or NULL. If NULL, it is treated as an empty parameter structure whose entries will be added to or overwrite those in parameters.

Details

This function adds new parameter entries from `newParameters` to `parameters`. If an entry with the same path is already present in `parameters`, its value and unit will be overwritten with the values from `newParameters`.

Value

Updated list of parameter paths, values, and units

`extendPopulationByUserDefinedParams` *Add user defined variability on parameters to a population.*

Description

Add user defined variability on parameters to a population.

Usage

```
extendPopulationByUserDefinedParams(
  population,
  parameterPaths,
  meanValues,
  sdValues,
  distributions = Distributions$Normal
)
```

Arguments

`population`	Object of type `Population`
`parameterPaths`	Vector of parameter path for which the variability is to be added.
`meanValues`	Vector of mean values of the parameters. Must have the same length as `parameterPaths`. The type of mean (arithmetic, geometric) depends on the selected `distribution`. The values must be in the base units of the parameters.
`sdValues`	Vector of standard deviation values of the parameters. Must have the same length as `parameterPaths`. The type of standard deviation depends on the selected `distribution`.
`distributions`	Type of distribution from which the random values will be sampled. Must have the same length as `parameterPaths`. A list of supported distributions is defined in `Distributions`. Default is `"Normal"`.

`extendPopulationFromXLS` *Add user defined variability on parameters to a population from an excel file.*

Description

Add user defined variability on parameters to a population from an excel file.

Usage

```
extendPopulationFromXLS(population, XLSpath, sheet = NULL)
```

Arguments

`population`	Object of type `Population`
`XLSpath`	Path to the excel file that stores the information of parameters. The file must have the columns "Container Path", "Parameter Name", "Mean", "SD", "Units", and "Distribution". Mean and SD values must be in the base units of the parameters.
`sheet`	Name or the index of the sheet in the excel file. If `NULL`, the first sheet in the file is used.

Details

The method reads the information from the specified excel sheet(s) and calls `extendPopulationByUserDefinedParams`

`GenderInt` *Possible gender entries as integer values*

Description

Possible gender entries as integer values

Usage

```
GenderInt
```

geomean *Calculate geometric mean of a numeric vector*

Description

Calculate geometric mean of a numeric vector

Usage

```
geomean(x, na.rm = FALSE, trim = 0)
```

Arguments

x	Numeric array to calculate geometric mean for
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds
trim	Fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint

Value

Geometric mean of x

geosd *Calculate geometric standard deviation of a numeric vector*

Description

Calculate geometric standard deviation of a numeric vector

Usage

```
geosd(x, na.rm = FALSE)
```

Arguments

x	Numeric array
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

Geometric standard deviation of x

getAllApplicationParameters *Get parameters of applications in the simulation*

Description

Get parameters of applications in the simulation

Usage

```
getAllApplicationParameters(simulation, moleculeNames = NULL)
```

Arguments

simulation	A Simulation object
moleculeNames	Names of the molecules which applications parameters will be returned. If NUll(default), applications for all molecules are returned.

Details

Every application event has a ProtocolSchemaItem container that holds parameters describing the dose, start time, infusion time etc. This function returns a list of all constant parameters located under the ProtocolSchemaItem container of applications defined for the moleculeNames.

Value

A list of Parameter objects defining the applications in the simulation.

Examples

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

applicationParams <- getAllApplicationParameters(
  simulation = simulation,
  moleculeNames = "Aciclovir"
)
```

getEsqlabsRSetting *Get the value of a global esqlabsR setting.*

Description

Get the value of a global esqlabsR setting.

Usage

```
getEsqlabsRSetting(settingName)
```

Arguments

settingName	String name of the setting

Value

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

Examples

```
getEsqlabsRSetting("packageVersion")
getEsqlabsRSetting("packageName")
```

getIndexClosestToValue *Find value in an array*

Description

Find the index of the value in an array that is closest to given one. By default, no restriction is applied how big the absolute numerical distance between `value` and a value in the `array` may be. A limit can be set by the parameters `thresholdAbs` or `thresholdRel`. If no value within the `array` has the distance to `value` that is equal to or less than the threshold, the `value` is considered not present in the `array` and `NULL` is returned.

Usage

```
getIndexClosestToValue(value, array, thresholdAbs = NULL, thresholdRel = NULL)
```

Arguments

value	Numerical value
array	Numerical array
thresholdAbs	Absolute numerical distance by which the closest value in array may differ from value to be accepted. If both thresholdAbs and thresholdRel are NULL (default), no threshold is applied. If thresholdAbs is set, thresholdRel is ignored. If 0, only exact match between value and array is accepted.
thresholdRel	A fraction by which the closest value may differ from value to be accepted. **WARNING**: setting a relative threshold will result in only exact matches if value is 0!

Value

Index of a value within the array which is closest to value and the difference is within the defined threshold. If multiple entries of array have the same difference which is minimal, a vector of indices is returned. If no value is within the defined threshold, NULL is returned.

---

getMoleculeNameFromQuantity

Get the name of the molecule from a quantity

Description

Returns the name of the molecule to which the quantity object is associated. The quantity could be the amount of the molecule in a container ('Organism|VenousBlood|Plasma|Aciclovir'), a parameter of the molecule ('Aciclovir|Lipophilicity' or 'Organism|VenousBlood|Plasma|Aciclovir|Concentration'), or an observer ("Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)").

If the quantity is not associated with a molecule (e.g. 'Organism|Weight'), an error is thrown.

Usage

```
getMoleculeNameFromQuantity(quantity)
```

Arguments

quantity	A Quantity object

Value

Name of the molecule the quantity is associated with.

Examples

```
simulation <- loadSimulation(system.file("extdata", "Aciclovir.pkml", package = "ospsuite"))
quantity <- getQuantity(
  path = "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)",
  container = simulation
)
getMoleculeNameFromQuantity(quantity = quantity)
```

---

initializeSimulation *Load a simulation and apply a set of parameters.*

Description

Helper method that combines a set of common steps performed before running a simulation. This method applies individual parameters data set and additional user-defined parameters to the simulation and runs the simulation to its steady-state and applies the steady-state as new initial conditions.

Usage

```
initializeSimulation(
  simulation,
  individualCharacteristics = NULL,
  additionalParams = NULL,
  stopIfParameterNotFound = TRUE
)
```

Arguments

`simulation`	`Simulation` loaded from a PKML file
`individualCharacteristics`	Optional `IndividualCharacteristics` describing an individual.
`additionalParams`	Optional named list with lists 'paths', 'values', and 'units'.
`stopIfParameterNotFound`	Logical. If `TRUE` (default), an error is thrown if any of the `additionalParams` does not exist. If `FALSE`, non-existent parameters are ignored.

Examples

```
## Not run:
simulation <- loadSimulation(filePath = modelPath)
humanIndividualCharacteristics <- createIndividualCharacteristics(
  species = Species$Human, population = HumanPopulation$European_ICRP_2002,
  gender = Gender$Male, weight = 70
)
userParams <- readParametersFromXLS(parameterXLSPath)
initializeSimulation(simulation, humanIndividualCharacteristics, userParams)
simulationResults <- runSimulations(simulation = simulation)
```

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

`initProject` *Initialize esqlabsR Project Folders and required Files*

Description

Creates the default project folder structure with Excel file templates in the working directory.

Usage

```
initProject(destination = ".", overwrite = FALSE)
```

Arguments

`destination`	A string defining the path where to initialize the project. default to current working directory.
`overwrite`	If TRUE, overwrites existing project without asking for permission. If FALSE and a project already exists, asks user for permission to overwrite.

`isAnyCriticalErrors` *Check if validation results contain any critical errors*

Description

Check if validation results contain any critical errors

Usage

```
isAnyCriticalErrors(validationResults)
```

Arguments

`validationResults`	Output from validateAllConfigurations

Value

Logical indicating if there are critical errors

`isParametersEqual` *Check if two parameters are equal with respect to certain properties.*

Description

Check if two parameters are equal with respect to certain properties.

Usage

```
isParametersEqual(
  parameter1,
  parameter2,
  checkFormulaValues = FALSE,
  compareFormulasByValue = FALSE
)
```

Arguments

`parameter1`	First parameter to compare
`parameter2`	Second parameter to compare
`checkFormulaValues`	If TRUE, values of explicit formulas are always compared. Otherwise, the values are only compared if the formulas are overridden (isFixedValue == TRUE). FALSE by default.
`compareFormulasByValue`	If `FALSE`(default), formulas are compared by their types and string. If `TRUE`, only values are compared.

Details

The parameters are not equal if: The paths of the parameters are not equal; The types of the formulas differ (types checked: isConstant, isDistributed, isExplicit, isTable); Constant formulas have different values; Distributed formulas have different values (not checking for distribution) Explicit formulas: If formula string are not equal, OR one of the parameter values is fixed (formula is overridden), OR both parameter values are fixed and differ, OR checkFormulaValues is TRUE and the values differ (disregarding of overridden or not) Table formulas: If the number of points differ, OR any of the points differ, OR one of the parameter values is fixed (formula is overridden), OR both parameter values are fixed and differ.

Value

`TRUE` if parameters are considered equal, `FALSE` otherwise

isProjectInitialized *Check if a directory contains an esqlabsR project*

Description

Checks if a directory already contains an esqlabsR project by looking for the presence of Project-Configuration.xlsx file or Configurations folder.

Usage

```
isProjectInitialized(destination = ".")
```

Arguments

destination	A string defining the path to check for an existing project. Defaults to current working directory.

Value

TRUE if an esqlabsR project exists in the directory, FALSE otherwise.

Examples

```
## Not run:
# Check if current directory has a project
hasProject <- isProjectInitialized()

# Check if specific directory has a project
hasProject <- isProjectInitialized("path/to/project")

## End(Not run)
```

isTableFormulasEqual *Check if two table formulas are equal.*

Description

Table formulas are equal if the number of points is equal and all x-y value pairs are equal between the two formulas

Usage

```
isTableFormulasEqual(formula1, formula2)
```

Arguments

formula1	First formula to compare
formula2	Second formula to compare

Value

TRUE if the table formulas are equal, FALSE otherwise

LLOQMode *Possible entries for the* lloqMode *argument of* calculateMeans()

Description

Possible entries for the lloqMode argument of calculateMeans()

Usage

```
LLOQMode
```

---

loadObservedData *Load data from excel*

Description

Loads data sets from excel. The excel file containing the data must be located in the folder projectConfiguration$dataFolder and be named projectConfiguration$dataFile. Importer configuration file must be located in the same folder and named projectConfiguration$dataImporterConfigurationFil

Usage

```
loadObservedData(
  projectConfiguration,
  sheets = NULL,
  importerConfiguration = NULL
)
```

Arguments

projectConfiguration	Object of class ProjectConfiguration containing the necessary information.
sheets	String or a list of strings defining which sheets to load. If NULL (default), all sheets within the file are loaded. This parameter takes precedence over any sheets defined in importerConfiguration: the function always sets importerConfiguration$sheets to NULL before calling ospsuite::loadDataSetsFromExcel(), so the passed importerConfiguration object is mutated as a side effect.

importerConfiguration
: DataImporterConfiguration object used to load the data. If NULL (default), default esqlabs importer configuration as defined in projectConfiguration$dataImporterConfigurat will be used. Note: the sheets property of this object is always set to NULL by this function (see the sheets parameter description).

Value

A named list of DataSet objects, with names being the names of the data sets.

Examples

```
## Not run:
# Create default project configuration
projectConfiguration <- createProjectConfiguration()
dataSets <- loadObservedData(projectConfiguration)

## End(Not run)
```

---

loadObservedDataFromPKML

Load data from pkml

Description

Loads data sets that are exported as pkml. The files must be located in the folder projectConfiguration$dataFolder, subfolder pkml. and be named projectConfiguration$dataFile.

Usage

```
loadObservedDataFromPKML(projectConfiguration, obsDataNames = NULL)
```

Arguments

projectConfiguration
: Object of class ProjectConfiguration containing the necessary information.

obsDataNames
: String or a list of strings defining data sets to load If NULL (default), all data sets located in the folder are loaded. Must not contain the ".pkml" file extension.

Value

A named list of DataSet objects, with names being the names of the data sets.

Examples

```
## Not run:
# Create default project configuration
projectConfiguration <- createProjectConfiguration()
dataSets <- loadObservedData(projectConfiguration)

## End(Not run)
```

`loadScenarioResults` *Load simulated scenarios from csv and pkml.*

Description

Load simulated scenarios from csv and pkml.

Usage

```
loadScenarioResults(scenarioNames, resultsFolder)
```

Arguments

`scenarioNames`	Names of simulated scenarios
`resultsFolder`	Path to the folder where simulation results as csv and the corresponding simulations as pkml are located.

Details

This function requires simulation results AND the corresponding simulation files being located in the same folder (`resultsFolder`) and have the names of the scenarios.

Value

A named list, where the names are scenario names, and the values are lists with the entries `simulation` being the initialized `Simulation` object with applied parameters, `results` being `SimulatioResults` object produced by running the simulation, and `outputValues` the output values of the `SimulationResults`.

Examples

```
## Not run:
# First simulate scenarios and save the results
projectConfiguration <- esqlabsR::createProjectConfiguration()
scenarioConfigurations <- readScenarioConfigurationFromExcel(
  projectConfiguration = projectConfiguration
)
scenarios <- createScenarios(scenarioConfigurations = scenarioConfigurations)
simulatedScenariosResults <- runScenarios(
  scenarios = scenarios
)
```

```
saveResults(simulatedScenariosResults, projectConfiguration)

# Now load the results
scnarioNames <- names(scenarios)
simulatedScenariosResults <- loadScenarioResults(
  scnarioNames = scnarioNames,
  resultsFolder = pathToTheFolder
)

## End(Not run)
```

`loadSensitivityCalculation` *Load Sensitivity Calculation Results*

Description

Restores a previously saved sensitivity calculation from a directory created with `saveSensitivityCalculation()`. If no simulation object is provided, the function loads the `simulation.pkml` bundled in the directory, falling back to the simulation file path stored in the metadata for folders saved before the pkml was bundled.

Usage

```
loadSensitivityCalculation(outputDir, simulation = NULL)
```

Arguments

`outputDir`	Path to the directory containing the saved sensitivity calculation files.
`simulation`	Optional. A `Simulation` object. If not provided, the function loads the `simulation.pkml` bundled in `outputDir`, or, if absent, the simulation stored at the source path recorded in the metadata.

Value

A named list of class `SensitivityCalculation`.

Examples

```
## Not run:
# Load sensitivity analysis result from disk
sensitivityCalculation <- loadSensitivityCalculation("output/my-sensitivity")

## End(Not run)
```

pathFromClipboard *Convert Windows filepaths for R*

Description

Converts the Windows-like path (using \) from the clipboard to the form readable by R (using /).

Usage

```
pathFromClipboard(path = "clipboard")
```

Arguments

path	Path that will be converted. If "clipboard" (default), path is queried from clipboard.

Value

String representation of a file path with / as separator.

PITaskConfiguration *PITaskConfiguration*

Description

An object storing configuration for a parameter identification (PI) task. This class holds references to PI task settings defined in ParameterIdentification.xlsx.

Active bindings

projectConfiguration ProjectConfiguration that will be used. Read-only.

taskName Name of the PI task. Key for lookup in ParameterIdentification.xlsx. Read-only.

scenarioConfiguration Named list of ScenarioConfiguration objects for the PI task. Read-only.

piConfiguration Named list of PI settings: algorithm, ciMethod, printEvaluationFeedback, autoEstimateCI, simulationRunOptions, objectiveFunctionOptions, algorithmOptions, ciOptions. Read-only.

piParameters Named list of parameter configurations from PIParameters sheet. Read-only.

piOutputMappings Named list of output mapping configurations from PIOutputMappings sheet. Read-only.

Methods

Public methods:

- PITaskConfiguration$new()
- PITaskConfiguration$print()
- PITaskConfiguration$clone()

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

Usage:

```
PITaskConfiguration$new(
  taskName,
  projectConfiguration,
  scenarioConfiguration,
  piDefinitions = NULL
)
```

Arguments:

`taskName` Character. Name of the PI task (key for lookup in Excel).

`projectConfiguration` An object of class `ProjectConfiguration`.

`scenarioConfiguration` An object of class `ScenarioConfiguration` or a named list of `ScenarioConfiguration` objects.

`piDefinitions` Named list containing:

- `piConfiguration`: Named list of PI settings
- `piParameters`: Named list of PI parameter configurations
- `piOutputMappings`: Named list of PI output mapping configurations

Returns: A new `PITaskConfiguration` object.

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

Usage:

```
PITaskConfiguration$print(
  className = TRUE,
  projectConfiguration = FALSE,
  scenarioConfiguration = FALSE
)
```

Arguments:

`className` Whether to print the name of the class at the beginning. Default is TRUE.

`projectConfiguration` Whether to also print project configuration. Default is FALSE.

`scenarioConfiguration` Whether to also print scenario configurations. Default is FALSE.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`ProjectConfiguration` *ProjectConfiguration*

Description

An object storing configuration used project-wide

Active bindings

`projectConfigurationFilePath` Path to the file that serve as base path for other parameters. If NULL, then, other paths should be absolute paths.

`projectConfigurationDirPath` Path to the folder that serve as base path for other paths. If NULL, then, other paths should be absolute paths.

`modified` Logical indicating whether any configuration properties have been modified since loading from file. Read-only.

`modelFolder` Path to the folder containing pkml simulation files.

`configurationsFolder` Path to the folder containing excel files with model parameterization.

`modelParamsFile` Name of the excel file with global model parameterization. Must be located in the "configurationsFolder".

`individualsFile` Name of the excel file with individual-specific model parameterization. Must be located in the "configurationsFolder".

`populationsFile` Name of the excel file with population information. Must be located in the "configurationsFolder".

`populationsFolder` Name of the folder containing population defined through csv files. Must be located in the "configurationsFolder".

`scenariosFile` Name of the excel file with scenario definitions. Must be located in the "configurationsFolder".

`applicationsFile` Name of the excel file scenario-specific parameters such as application protocol parameters. Must be located in the "configurationsFolder".

`plotsFile` Name of the excel file with plot definitions. Must be located in the "configurationsFolder".

`parameterIdentificationFile` Name of the excel file with parameter identification definitions. Must be located in the "configurationsFolder".

`dataFolder` Path to the folder where experimental data files are located.

`dataFile` Name of the excel file with experimental data. Must be located in the "dataFolder"

`dataImporterConfigurationFile` Name of data importer configuration file in xml format used to load the data. Must be located in the "dataFolder"

`outputFolder` Path to the folder where the results should be saved relative to the "Code" folder

`esqlabsRVersion` Version of the esqlabsR package stored in the project configuration. Read-only. Initialize

Methods

Public methods:

- ProjectConfiguration$new()
- ProjectConfiguration$print()
- ProjectConfiguration$save()
- ProjectConfiguration$clone()

`ProjectConfiguration$new()`:

Usage:

```
ProjectConfiguration$new(
  projectConfigurationFilePath = character(),
  ignoreVersionCheck = FALSE
)
```

Arguments:

`projectConfigurationFilePath` A string representing the path to the project configuration file.

`ignoreVersionCheck` If `TRUE`, skip the version mismatch check when loading the configuration file. Defaults to `FALSE`. Print

`ProjectConfiguration$print()`: print prints a summary of the Project Configuration.

Usage:

```
ProjectConfiguration$print(className = TRUE)
```

Arguments:

`className` Whether to print the name of the class at the beginning. default to TRUE.

`ProjectConfiguration$save()`: Export ProjectConfiguration object to ProjectConfiguration.xlsx

Usage:

```
ProjectConfiguration$save(path)
```

Arguments:

`path` a string representing the path or file name where to save the file. Can be absolute or relative (to working directory).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`projectConfigurationStatus` *Check if Excel configuration files are in sync with JSON snapshot*

Description

Compares Excel configuration files against their JSON snapshot to determine if they are synchronized. The JSON snapshot is considered the source of truth.

Usage

```
projectConfigurationStatus(
  projectConfig = "ProjectConfiguration.xlsx",
  jsonPath = NULL,
  silent = FALSE,
  ignoreVersionCheck = TRUE
)
```

Arguments

`projectConfig`	A ProjectConfiguration object or path to ProjectConfiguration excel file. Defaults to "ProjectConfiguration.xlsx".
`jsonPath`	Path to the JSON configuration file. If NULL (default), the function will look for a JSON file with the same name as the ProjectConfiguration file but with .json extension.
`silent`	Logical indicating whether to suppress informational messages. Defaults to FALSE.
`ignoreVersionCheck`	Logical. If `TRUE`, the version check between the stored `esqlabsR` version in the configuration and the currently installed version is skipped. Defaults to `TRUE` because this function focuses on file synchronization status, not version compatibility.

Value

A list with components:

`in_sync`	Logical indicating whether all files are synchronized
`details`	A list with detailed comparison results for each file
`unsaved_changes`	Logical indicating whether the ProjectConfiguration object has unsaved modifications

See Also

Other project configuration snapshots: `restoreProjectConfiguration()`, `snapshotProjectConfiguration()`

readExcel *Read XLSX files using* readxl::read_excel *with suppressed warnings*

Description

Read XLSX files using readxl::read_excel with suppressed warnings

Usage

```
readExcel(path, sheet = NULL, ...)
```

Arguments

path	Full path of an XLS/XLSX file
sheet	Name or number of the sheet. If NULL (default), the first sheet of the file is used.
...	Any other parameters that can be passed to readxl::read_excel

Value

A tibble with the contents of the excel sheet

readIndividualCharacteristicsFromXLS *Read individual characteristics from file*

Description

Read individual characteristics from file

Usage

```
readIndividualCharacteristicsFromXLS(
  XLSpath,
  individualId,
  sheet = "IndividualBiometrics",
  nullIfNotFound = TRUE
)
```

Arguments

XLSpath	Full path to the excel file
individualId	(String) Id of the individual as stored in the IndividualId column.
sheet	Name of the sheet. If NULL (default), the first sheet of the file is used.
nullIfNotFound	Boolean. If TRUE (default), NULL is returned if no entry with the give individualId exists. Otherwise, an error is thrown.

Details

Read individual characteristics from an excel sheet and create an `IndividualCharacteristics`-object. The excel sheet must have the columns `IndividualId`, `Species`, `Population`, `Gender`, `Weight [kg]`, `Height [cm]`, `Age [year(s)]`, and `Protein Ontogenies`.

Value

An `IndividualCharacteristics` object

`readParametersFromXLS` *Read parameter values from a structured Excel file. Each excel sheet must consist of columns 'Container Path', 'Parameter Name', 'Value', and 'Units'*

Description

Read parameter values from a structured Excel file. Each excel sheet must consist of columns 'Container Path', 'Parameter Name', 'Value', and 'Units'

Usage

```
readParametersFromXLS(paramsXLSpath, sheets = NULL)
```

Arguments

`paramsXLSpath`	Path to the excel file
`sheets`	Names of the excel sheets containing the information about the parameters. Multiple sheets can be processed. If no sheets are provided, the first one in the Excel file is used.

Value

A list containing vectors `paths` with the full paths to the parameters, `values` the values of the parameters, and `units` with the units the values are in.

readPITaskConfigurationFromExcel
Read Parameter Identification configurations from Excel

Description

Read Parameter Identification configurations from Excel

Usage

```
readPITaskConfigurationFromExcel(piTaskNames = NULL, projectConfiguration)
```

Arguments

piTaskNames	Character vector. Names of the parameter identification tasks that are defined in the Excel file. If NULL (default), all tasks specified in the Excel file will be read.
projectConfiguration	A ProjectConfiguration object holding base project information.

Details

Reads PI task configuration from the Excel file defined in ProjectConfiguration and creates PITaskConfiguration objects. If a PI task that is specified in piTaskNames is not found in the Excel file, an error is thrown.

The function expects the Excel file to have a "PIOutputMappings" sheet with PITaskName, Scenarios, OutputPath, ObservedDataSheet, DataSet, Scaling, xOffset, yOffset, xFactor, yFactor, Weight columns. OutputPath accepts either a full simulation output path or an OutputPathId defined in the "OutputPaths" sheet of Scenarios.xlsx. It also expects a "PIParameters" sheet with PITaskName, Scenarios, Container Path, Parameter Name, Units, MinValue, MaxValue, StartValue, Group columns, an optional "PIConfiguration" sheet with PITaskName, Algorithm, CIMethod, PrintEvaluationFeedback, AutoEstimateCI, numberOfCores, checkForNegativeValues, ObjectiveFunctionType, ResidualWeightingMethod, RobustMethod, ScaleVar, LinScaleCV, LogScaleSD columns, an "AlgorithmOptions" sheet with PITaskName, OptionName, OptionValue columns, and a "CIOptions" sheet with PITaskName, OptionName, OptionValue columns.

Value

A named list of PITaskConfiguration objects.

Examples

```
## Not run:
projectConfiguration <- createProjectConfiguration(
  exampleProjectConfigurationPath(),
  ignoreVersionCheck = TRUE
)
piTaskConfigurations <- readPITaskConfigurationFromExcel(
```

```
  piTaskNames = "AciclovirSimple",
  projectConfiguration = projectConfiguration
)

## End(Not run)
```

`readPopulationCharacteristicsFromXLS` *Read an excel file containing information about population and create a* `PopulationCharacteristics` *object*

Description

Read an excel file containing information about population and create a `PopulationCharacteristics` object

Usage

```
readPopulationCharacteristicsFromXLS(XLSpath, populationName, sheet = NULL)
```

Arguments

`XLSpath`	Path to the excel file
`populationName`	Name of the population, as defined in the "PopulationName" column
`sheet`	Name or the index of the sheet in the excel file. If `NULL`, the first sheet in the file is used.

Value

A `PopulationCharacteristics` object based on the information in the excel file.

`readScenarioConfigurationFromExcel` *Read scenario definition(s) from Excel file*

Description

Read scenario definition(s) from Excel file

Usage

```
readScenarioConfigurationFromExcel(scenarioNames = NULL, projectConfiguration)
```

Arguments

scenarioNames	Character vector. Names of the scenarios that are defined in the Excel file. If NULL (default), all scenarios specified in the Excel file will be created.
projectConfiguration	A ProjectConfiguration object holding base information.

Details

Reads scenario definition from the Excel file defined in ProjectConfiguration and creates ScenarioConfiguration objects with new information. If a scenario that is specified in scenarioNames is not found in the Excel file, an error is thrown.

The function expects the Excel file to have a "Scenarios" sheet with the following columns: Scenario_name, IndividualId, PopulationId, ReadPopulationFromCSV, ModelParameterSheets, ApplicationProtocol, SimulationTime, SimulationTimeUnit, SteadyState, SteadyStateTime, SteadyStateTimeUnit, ModelFile, OutputPathsIds. It also expects an "OutputPaths" sheet with OutputPathId and OutputPath columns.

Value

A named list of ScenarioConfiguration objects with the names of the list being scenario names.

Examples

```
## Not run:
# Create default ProjectConfiguration
projectConfiguration <- createProjectConfiguration(ignoreVersionCheck = TRUE)
scenarioName <- "MyScenario"
# Read scenario definition from Excel
scenarioConfiguration <- readScenarioConfigurationFromExcel(
  scenarioNames = scenarioName,
  projectConfiguration
)[[scenarioName]]

## End(Not run)
```

---

removeFromList *Remove an entry from a list*

Description

Removes all occurrences of the entry from the list. If the entry is not in the list nothing is removed.

Usage

```
removeFromList(entry, listArg)
```

Arguments

entry	The entry to be removed
listArg	The list from which the entry will be removed

Value

The list without the entry. If the input is a vector, it is converted to a list.

Examples

```
myList <- list("one", "two", "one", "three")
myList <- removeFromList("one", myList)
print(myList)
```

---

restoreProjectConfiguration

Import project configuration from JSON to Excel files

Description

Creates Excel configuration files from a JSON configuration file. This allows for recreating the project configuration from version-controlled JSON.

Usage

```
restoreProjectConfiguration(
  jsonPath = "ProjectConfiguration.json",
  outputDir = NULL,
  silent = FALSE
)
```

Arguments

jsonPath	Path to the JSON configuration file. Defaults to "ProjectConfiguration.json".
outputDir	Directory where the Excel files will be created. If NULL (default), the Excel files will be created in the same directory as the source JSON file.
silent	Logical. If TRUE, suppresses informational messages. Defaults to FALSE.

Value

A ProjectConfiguration object initialized with the regenerated ProjectConfiguration.xlsx

See Also

Other project configuration snapshots: projectConfigurationStatus(), snapshotProjectConfiguration()

runPI *Run Parameter Identification tasks*

Description

Executes parameter identification for all PI tasks. Handles failures gracefully - continues with other tasks if one fails.

Usage

```
runPI(piTasks)
```

Arguments

piTasks	Named list of ParameterIdentification objects usually created using createPITasks

Value

Named list of PI results. Each result contains:

- task: original ParameterIdentification object
- result: PIResult object from task$run(), or NULL if failed
- error: Error message string if failed, absent on success

Examples

```
## Not run:
# After creating PI tasks
piResults <- runPI(piTasks)

## End(Not run)
```

runScenarios *Run a set of scenarios.*

Description

Run a set of scenarios.

Usage

```
runScenarios(scenarios, simulationRunOptions = NULL)
```

Arguments

scenarios	List of Scenario objects to be simulated.
simulationRunOptions	Object of type SimulationRunOptions that will be passed to simulation runs. If NULL, default options are used.

Details

If simulation of a scenario fails, a warning is produced, and the outputValues for this scenario is NULL.

Value

A named list, where the names are scenario names, and the values are lists with the entries simulation being the initialized Simulation object with applied parameters, results being SimulatioResults object produced by running the simulation, outputValues the output values of the SimulationResults, and population the Population object if the scenario is a population simulation.

---

sampleRandomValue *Sample a random value from a distribution*

Description

Sample a random value from a distribution

Usage

```
sampleRandomValue(distribution, mean, sd, n)
```

Arguments

distribution	The type of the distribution the random variable is to be sampled from. See Distributions for the list of supported entries.
mean	Mean value of the random variable
sd	Standard deviation of the random variable
n	Size of the sample

Value

Numerical vector of size n with randomly sampled values

saveScenarioResults *Save results of scenario simulations to csv.*

Description

Save results of scenario simulations to csv.

Usage

```
saveScenarioResults(
  simulatedScenariosResults,
  projectConfiguration,
  outputFolder = NULL,
  saveSimulationsToPKML = TRUE
)
```

Arguments

simulatedScenariosResults	Named list with simulation, results, outputValues, and population as produced by runScenarios().
projectConfiguration	An instance of ProjectConfiguration
outputFolder	Optional - path to the folder where the results will be stored. If NULL (default), a sub-folder in ProjectConfiguration$outputFolder/SimulationResults/<DateSuffix>.
saveSimulationsToPKML	If TRUE (default), simulations corresponding to the results are saved to PKML along with the results.

Details

For each scenario, a separate csv file will be created. If the scenario is a population simulation, a population is stored along with the results with the file name suffix _population. Results can be read with the loadScenarioResults() function.

Value

outputFolder or the created output folder path, if no outputFolder was provided.

Examples

```
## Not run:
projectConfiguration <- esqlabsR::createProjectConfiguration()
scenarioConfigurations <- readScenarioConfigurationFromExcel(
  projectConfiguration = projectConfiguration
)
scenarios <- createScenarios(scenarioConfigurations = scenarioConfigurations)
```

```
simulatedScenariosResults <- runScenarios(
  scenarios = scenarios
)
saveScenarioResults(simulatedScenariosResults, projectConfiguration)

## End(Not run)
```

---

saveSensitivityCalculation
Save Sensitivity Calculation Results

Description

Saves the results of a sensitivity analysis to a specified directory, including metadata and simulation output required for restoring or sharing the analysis. The underlying simulation is written to `simulation.pkml` in the same directory so the saved calculation is self-contained and can be reloaded without access to the original simulation file.

Usage

```
saveSensitivityCalculation(
  sensitivityCalculation,
  outputDir,
  overwrite = FALSE
)
```

Arguments

`sensitivityCalculation`	A named list of class `SensitivityCalculation` as returned by `sensitivityCalculation()`, containing `simulationResults`, `outputPaths`, `parameterPaths`, and `pkData`.
`outputDir`	A character string specifying the path to the directory where the results should be saved.
`overwrite`	Logical. If `TRUE`, an existing directory at `outputDir` will be deleted and replaced. Default is `FALSE`.

Value

Invisibly returns `NULL`. Results are saved to disk in the specified folder.

Examples

```
## Not run:
sensitivityCalculation <- sensitivityCalculation(
  simulation = mySim,
  outputPaths = "Organism|PeripheralVenousBlood|Drug|Plasma",
  parameterPaths = c("Drug|Lipophilicity", "Application|Dose")
)
```

```
saveSensitivityCalculation(
  sensitivityCalculation,
  outputDir = "output/my-sensitivity",
  overwrite = TRUE
)

## End(Not run)
```

`Scenario` *Scenario*

Description

Simulation scenario

Active bindings

`scenarioConfiguration` `scenarioConfiguration` used for creation of this scenario. Read-only.

`finalCustomParams` Custom parameters to be used for the simulation. Read-only.

`simulation` Simulation object created from the `ScenarioConfiguration`. Read-only

`population` Population object in case the scenario is a population simulation. Read-only.

`scenarioType` Type of the scenario - individual or population. Read-only

Methods

Public methods:

- `Scenario$new()`
- `Scenario$print()`

`Scenario$new()`**:** Custom parameters to be used for the simulation. The final custom parameters are a combination of parametrization through the excel files and the custom parameters specified by the user through the `customParams` argument of the `Scenario` constructor.

Simulation object. Read-only.

Initialize a new instance of the class. Initializes the scenario from `ScenarioConfiguration` object.

Usage:

```
Scenario$new(
  scenarioConfiguration,
  customParams = NULL,
  stopIfParameterNotFound = TRUE
)
```

Arguments:

`scenarioConfiguration` An object of class `ScenarioConfiguration`.

`customParams` Custom parameters to be used for the simulation. A list containing vectors 'paths' with the full paths to the parameters, 'values' the values of the parameters, and 'units' with the units the values are in.

`stopIfParameterNotFound` Logical. If `TRUE` (default), an error is thrown if any of the custom defined parameter does not exist. If `FALSE`, non-existent parameters are ignored.

Returns: A new `Scenario` object.

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

Usage:

```
Scenario$print(...)
```

Arguments:

`...` Rest arguments.

`ScenarioConfiguration` *ScenarioConfiguration*

Description

An object storing configuration of a specific scenario

Public fields

`scenarioName` Name of the simulated scenario

`modelFile` Name of the simulation to be loaded (must include the extension ".pkml"). Must be located in the "modelFolder".

`applicationProtocol` Name of the application protocol to be applied. Defined in the excel file "Applications.xlsx"

`individualId` Id of the individual as specified in "Individuals.xlsx". If `NULL` (default), the individual as defined in the simulation file will be simulated.

`populationId` Id of the population as specified in "Populations.xlsx", sheet "Demographics". If `ScenarioConfguration$simulationType` is `population`, a population will be created a the scenario will be simulated as a population simulation.

`outputPaths` a character vector or named vector of output paths for which the results will be calculated. If `NULL` (default), outputs as defined in the simulation are used. Can be a named vector where names serve as aliases for the paths, e.g., c("plasma" = "Organism|VenousBlood|Plasma|AKB-9090|Concentration in container").

Active bindings

`simulateSteadyState` Boolean representing whether the simulation will be brought to a steady-state first

`readPopulationFromCSV` Boolean representing whether the a new population will be created (value is `FALSE`) or an existing population will be imported from a csv.

`simulationTime` Specified simulation time intervals. If `NULL` (default), simulation time as defined in the `Simulation` object will be used. Accepted are multiple time intervals separated by a ';'. Each time interval is a triplet of values <StartTime, EndTime, Resolution>, where `Resolution` is the number of simulated points per time unit defined in the column `TimeUnit`.

`simulationTimeUnit` Unit of the simulation time intervals.

`steadyStateTime` Time in minutes to simulate if simulating steady-state. May be `NULL`.

`paramSheets` A named list. Names of the sheets from the parameters-excel file that will be applied to the simulation

`simulationType` Type of the simulation - "Individual" or "Population". If "Population", population characteristics are created based on information stored in `populationsFile`. Default is "Individual"

`projectConfiguration` `ProjectConfiguration` that will be used in scenarios. Read-only

Methods

Public methods:

- `ScenarioConfiguration$new()`
- `ScenarioConfiguration$addParamSheets()`
- `ScenarioConfiguration$removeParamSheets()`
- `ScenarioConfiguration$print()`
- `ScenarioConfiguration$clone()`

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

Usage:

```
ScenarioConfiguration$new(projectConfiguration)
```

Arguments:

`projectConfiguration` An object of class `ProjectConfiguration`.

Returns: A new `ScenarioConfiguration` object.

`ScenarioConfiguration$addParamSheets()`: Add the names of sheets in the parameters excel-file that will be applied to the simulation

Usage:

```
ScenarioConfiguration$addParamSheets(sheetNames)
```

Arguments:

`sheetNames` A name or a list of names of the excel sheet

`ScenarioConfiguration$removeParamSheets()`: Remove the names of sheets in the parameters excel-file from the list of sheets `paramSheets`

Usage:

```
ScenarioConfiguration$removeParamSheets(sheetNames = NULL)
```

Arguments:

`sheetNames` A name or a list of names of the excel sheet. If `NULL` (default), all sheets are removed.

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

Usage:

```
ScenarioConfiguration$print(className = TRUE, projectConfiguration = FALSE)
```

Arguments:

`className` Whether to print the name of the class at the beginning. default to TRUE.

`projectConfiguration` Whether to also print project configuration. default to TRUE.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`sensitivityCalculation` *Carry out and visualize sensitivity analysis (with OSPSuite)*

Description

Carry out and visualize sensitivity analysis (with OSPSuite)

Usage

```
sensitivityCalculation(
  simulation,
  outputPaths,
  parameterPaths,
  variationRange = c(seq(0.1, 1, by = 0.1), seq(2, 10, by = 1)),
  variationType = c("relative", "absolute"),
  pkParameters = c("C_max", "t_max", "AUC_inf"),
  customOutputFunctions = NULL,
  saOutputFilePath = NULL,
  simulationRunOptions = NULL
)
```

Arguments

`simulation`	An object of type `Simulation`.
`outputPaths`	Path (or a vector of paths) to the output(s) for which the sensitivity will be analyzed.
`parameterPaths`	A single or a vector of the parameter path(s) to be varied. Can also be a named vector, where the names are user-defined labels. These names will be stored and used in downstream plotting functions (e.g., as legend labels) if provided.

`variationRange` Optional numeric vector or list defining the scaling of the parameters. The same variation range is applied to all specified parameters unless a list is provided, in which case the length of the list must match the length of `parameterPaths`, allowing individual variation for each parameter. If not specified, the following vector will be used: c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

`variationType` A string specifying whether the values in `variationRange` are applied as `"absolute"` or `"relative"` scaling. When set to `"absolute"`, the values are interpreted as absolute parameter values. When set to `"relative"`, the values are interpreted as scaling factors relative to the initial parameter values. Default is `"relative"`. When `"absolute"`, every parameter must have a non-zero initial value; an error is thrown otherwise.

`pkParameters` A vector of names of PK parameters for which the sensitivities will be calculated. For a full set of available standard PK parameters, run `names(ospsuite::StandardPKParameter)`. By default, the following parameters are considered: `"C_max"`, `"t_max"`, `"AUC_inf"`. If `NULL`, all available PK-parameters (including the user-defined) will be calculated.

`customOutputFunctions`
A named list with custom function(s) for output calculations. User-defined functions should have either 'x', 'y', or both 'x' and 'y' as parameters which correspond to x-Dimension (time) or y-Dimension values from simulation results. The output of the function is a single numerical value for each output and parameter path, which is then included in the returned dataframe of PK parameters.

`saOutputFilePath`
Path to excel file in which PK-parameter data should be saved. If a file already exists, it will be overwritten. Default is `NULL`, meaning the data will not be saved to a spreadsheet.

`simulationRunOptions`
Optional instance of a `SimulationRunOptions` used during the simulation run

Value

A list containing following objects:

- `SimulationResults`
- specified output paths
- specified parameter paths
- A data frame of PK parameters

See Also

Other sensitivity-calculation: `sensitivitySpiderPlot()`, `sensitivityTimeProfiles()`, `sensitivityTornadoPlot()`

Examples

```
## Not run:
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
simulation <- loadSimulation(simPath)
```

```
outputPaths <- "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)"
parameterPaths <- c(
  "Aciclovir|Lipophilicity",
  "Events|IV 250mg 10min|Application_1|ProtocolSchemaItem|Dose",
 "Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Glomerular Filtration-GFR-Aciclovir|GFR fraction"
)

# extract the results into a list of dataframes
sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = parameterPaths
)

# Calculate sensitivity for a user-defined function that computes the
# averate of the simulated y-values
customOutputFunctions <- list(
  "Average" = function(y) mean(y)
)
sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = parameterPaths,
  customOutputFunctions = customOutputFunctions
)

## End(Not run)
```

`sensitivitySpiderPlot` *Sensitivity Spider Plot for Pharmacokinetic Parameters*

Description

Creates spider plots for sensitivity calculation. A spider plot is a visualization technique that displays the sensitivity of a model output to changes in model parameters. Each plot displays the sensitivity of a set of PK parameters for a single output path to changes in model parameters. The x-axis represents the value of model parameters (absolute or percent from default value), and the y-axis represents the sensitivity of the output to changes in the model parameter.

Usage

```
sensitivitySpiderPlot(
  sensitivityCalculation,
  outputPaths = NULL,
  parameterPaths = NULL,
  pkParameters = NULL,
  xAxisScale = NULL,
  yAxisScale = NULL,
```

```
  xAxisType = "percent",
  yAxisType = "percent",
  yAxisFacetScales = "fixed",
  defaultPlotConfiguration = NULL
)
```

Arguments

`sensitivityCalculation`
: The `SensitivityCalculation` object returned by `sensitivityCalculation()`.

`outputPaths`, `parameterPaths`, `pkParameters`
: A single or a vector of the output path(s), parameter path(s), and PK parameters to be displayed, respectively. If `NULL`, all included paths and parameters present in the supplied `SensitivityCalculation` object will be displayed in the visualization. A separate plot will be generated for each output path. Each plot will contain a spider plot panel for each PK parameter, and the sensitivities for each parameter will be displayed as lines.

`xAxisScale`
: Character string, either "log" (logarithmic scale) or "lin" (linear scale), to set the x-axis scale. Default is "log".

`yAxisScale`
: Character string, either "log" or "lin", sets the y-axis scale similarly to `xAxisScale`. Default is "lin".

`xAxisType`
: Character string, either "percent" (percentage change) or "absolute" (absolute values) for PK parameter values, for x-axis data normalization. Default is "percent".

`yAxisType`
: Character string, either "percent" (percentage change) or "absolute" (absolute values) for PK parameter values, for y-axis data normalization. Default is "percent".

`yAxisFacetScales`
: Character string, either "fixed" or "free", determines the scaling across y-axes of different facets. Default is "fixed". If "fixed", all facetes within one plot will have the same range, which allows for easier comparison between different PK parameters. If "free", each facet will have its own range, which allows for better visualization of the single PK parameters sensitivity.

`defaultPlotConfiguration`
: An object of class `DefaultPlotConfiguration` used to customize plot aesthetics. Plot-specific settings provided directly to the function, such as `xAxisScale`, will take precedence over any modifications in `defaultPlotConfiguration`.

  Supported parameters include:

  - `legendPosition`: Position of the legend on the plot.
  - `legendTitle`: Title displayed for the legend.
  - `linesAlpha`: Alpha transparency for line elements.
  - `linesColor`: Color of the line elements.
  - `linesSize`: Thickness of the line elements.
  - `pointsShape`: Shape of the point elements.
  - `pointsSize`: Size of the point elements.
  - `subtitle`: Subtitle text for the plot.

- `title`: Main title text for the plot.
- `titleSize`: Font size of the plot title.
- `xAxisScale`: Scale type for the x-axis (`"log"` or `"lin"`).
- `xLabel`: Label text for the x-axis.
- `xValuesLimits`: Numeric vector specifying the limits for x-values.
- `yAxisLimits`: Numeric vector specifying the limits for y-values.
- `yAxisScale`: Scale type for the y-axis (`"log"` or `"lin"`).
- `yAxisTicks`: Number of ticks on the y-axis.
- `yLabel`: Label text for the y-axis.
- `yValuesLimits`: Numeric vector specifying the limits for y-values.

Default values are set to provide a standardized look, but each parameter can be tailored to fit specific visual needs. Modifying these parameters will directly affect the aesthetics of the output plots.

Value

A `patchwork` object containing the combined ggplot objects if a single output path is specified, or a list of `patchwork` objects for multiple output paths.

See Also

Other sensitivity-calculation: `sensitivityCalculation()`, `sensitivityTimeProfiles()`, `sensitivityTornadoPlot()`

Examples

```
## Not run:
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
simulation <- loadSimulation(simPath)
outputPaths <- "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)"
parameterPaths <- c(
  "Aciclovir|Lipophilicity",
  "Events|IV 250mg 10min|Application_1|ProtocolSchemaItem|Dose",
 "Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Glomerular Filtration-GFR-Aciclovir|GFR fraction"
)

results <- sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = parameterPaths
)

# Print plots with default settings
sensitivitySpiderPlot(results)

# Print plots with absolute y-axis values
sensitivitySpiderPlot(results, yAxisType = "absolute", yAxisFacetScales = "free")

# Print plots with custom configuration settings
myPlotConfiguration <- createEsqlabsPlotConfiguration()
myPlotConfiguration$pointsShape <- 22
```

```
myPlotConfiguration$subtitle <- "Custom settings"
sensitivitySpiderPlot(results, defaultPlotConfiguration = myPlotConfiguration)

# Use named parameter paths to customize legend labels
namedParameterPaths <- c(
  "Lipophilicity" = "Aciclovir|Lipophilicity",
  "Dose" = "Events|IV 250mg 10min|Application_1|ProtocolSchemaItem|Dose",
 "GFR fraction" = "Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Glomerular Filtration-GFR-Aciclovir|GFR fraction
)

resultsNamed <- sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = namedParameterPaths
)

sensitivitySpiderPlot(resultsNamed)

## End(Not run)
```

`sensitivityTimeProfiles` *Time Profile plots for Sensitivity Analysis*

Description

Creates time profiles for selected outputs generated in a sensitivity analysis. This function plots time profiles for each specified output path, illustrating the dynamics of model outputs to parameter variations.

Usage

```
sensitivityTimeProfiles(
  sensitivityCalculation,
  outputPaths = NULL,
  parameterPaths = NULL,
  xAxisScale = NULL,
  yAxisScale = NULL,
  xUnits = NULL,
  yUnits = NULL,
  observedData = NULL,
  defaultPlotConfiguration = NULL
)
```

Arguments

`sensitivityCalculation`
: The `SensitivityCalculation` object returned by `sensitivityCalculation()`.

outputPaths, parameterPaths
: A single or a vector of the output path(s) to be plotted for parameter path(s) which impact is analyzed, respectively. If NULL, all included paths and parameters present in the supplied SensitivityCalculation object will be displayed in the visualization. A separate plot will be generated for each output path, and a separate curve will be generated for each parameter variation. A separate panel is created for each varied parameter.

xAxisScale
: Character string, either "log" (logarithmic scale) or "lin" (linear scale), to set the x-axis scale. Default is "lin".

yAxisScale
: Character string, either "log" or "lin", sets the y-axis scale similarly to xAxisScale. Default is "log".

xUnits, yUnits
: Units for the x-axis and y-axis, respectively. Can be provided as a single string (e.g., "nmol/l") or a list of strings. A single string or a list of length one will be applied to all outputPaths if conversion is possible. If a list of multiple units is provided, the units list should correspond to the outputPaths, and units conversion will be applied accordingly. If NULL, default units from the simulation results will be used.

observedData
: Optional. A DataSet or a list of DataSet objects containing observed data. If provided, observed data will be plotted together with the simulated data based on OutputPath dimension for direct comparison within the visualizations. Observed data will only be added to plots with matching y-dimensions.

defaultPlotConfiguration
: An object of class DefaultPlotConfiguration used to customize plot aesthetics. Plot-specific settings provided directly to the function, such as xAxisScale, will take precedence over any modifications in defaultPlotConfiguration. If not provided, default settings are applied.

  Supported parameters for defaultPlotConfiguration include:

  - legendPosition: Specifies the position of the plot legend.
  - legendTitle: Sets the title displayed for the legend.
  - linesAlpha: Alpha transparency for the line elements.
  - linesColor: Color of the line elements.
  - linesSize: Thickness of the line elements.
  - pointsShape: Shape of the point elements for observed data.
  - title: Main title text for the plot.
  - titleSize: Font size of the plot title.
  - xAxisScale: Scale type for the x-axis ("log" or "lin").
  - xLabel: Label text for the x-axis.
  - yAxisScale: Scale type for the y-axis ("log" or "lin").
  - yLabel: Label text for the y-axis.

Value

A patchwork object containing the combined ggplot objects if a single output path is specified, or a list of patchwork objects for multiple output paths.

See Also

Other sensitivity-calculation: sensitivityCalculation(), sensitivitySpiderPlot(), sensitivityTornadoPlot()

Examples

```
## Not run:
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
simulation <- loadSimulation(simPath)
outputPaths <- "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)"
parameterPaths <- c(
  "Aciclovir|Lipophilicity",
  "Events|IV 250mg 10min|Application_1|ProtocolSchemaItem|Dose",
 "Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Glomerular Filtration-GFR-Aciclovir|GFR fraction"
)

results <- sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = parameterPaths
)

# Print plots with default settings
sensitivityTimeProfiles(results)

# Print plots with linear y-axis values
sensitivityTimeProfiles(results, yAxisScale = "lin")

# Print plots with custom configuration settings
myPlotConfiguration <- createEsqlabsPlotConfiguration()
myPlotConfiguration$linesColor <- c("#4D8076", "#C34A36")
myPlotConfiguration$subtitle <- "Custom settings"
sensitivityTimeProfiles(results, defaultPlotConfiguration = myPlotConfiguration)

# Use named parameter paths to customize facet labels
namedParameterPaths <- c(
  "Lipophilicity" = "Aciclovir|Lipophilicity",
  "Dose" = "Events|IV 250mg 10min|Application_1|ProtocolSchemaItem|Dose",
 "GFR fraction" = "Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Glomerular Filtration-GFR-Aciclovir|GFR fractior
)

resultsNamed <- sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = namedParameterPaths
)

sensitivitySpiderPlot(resultsNamed)

## End(Not run)
```

`sensitivityTornadoPlot` *Tornado Plot for Sensitivity Analysis*

Description

Generates tornado plots to visualize the results of sensitivity analysis. Each plot shows the effect of modifying parameters by a specific scaling factor (`parameterFactor`) and its reciprocal on specific model outputs. This visualization helps to assess the impact of parameter changes on the results, highlighting the model's sensitivity to these parameters.

Usage

```
sensitivityTornadoPlot(
  sensitivityCalculation,
  outputPaths = NULL,
  parameterPaths = NULL,
  pkParameters = NULL,
  parameterFactor = 0.1,
  xAxisZoomRange = NULL,
  defaultPlotConfiguration = NULL
)
```

Arguments

`sensitivityCalculation`
: The `SensitivityCalculation` object returned by `sensitivityCalculation()`.

`outputPaths`, `parameterPaths`, `pkParameters`
: A single or a vector of the output path(s), parameter path(s), and PK parameters to be displayed, respectively. If `NULL`, all included paths and parameters present in the supplied `SensitivityCalculation` object will be displayed in the visualization. A separate plot will be generated for each output path. Each plot will contain a tornado plot panel for each PK parameter, and the sensitivities for each parameter will be displayed as lines.

`parameterFactor`
: Numeric; the scaling factor used to adjust parameters during sensitivity analysis used in the tornado plot. Both the `parameterFactor` and its reciprocal (`1/parameterFactor`) must be included in the `variationRange` specified in the `sensitivityCalculation`. Default is 0.1.

`xAxisZoomRange`
: Numeric vector of length 2; specifies the x-axis limits to zoom into. This does not remove any data but constrains the visible plotting range for improved readability.

`defaultPlotConfiguration`
: An object of class `DefaultPlotConfiguration` used to customize plot aesthetics.

  Supported parameters include:

- `legendPosition`: Position of the legend on the plot.
- `legendTitle`: Title displayed for the legend.
- `linesColor`: Color of the bar elements.
- `subtitle`: Subtitle text for the plot.
- `title`: Main title text for the plot.
- `titleSize`: Font size of the plot title.
- `xLabel`: Label text for the x-axis.
- `yLabel`: Label text for the y-axis.

Default values are set to provide a standardized look, but each parameter can be tailored to fit specific visual needs. Modifying these parameters will directly affect the aesthetics of the output plots.

Value

A `patchwork` object containing the combined ggplot objects if a single output path is specified, or a list of `patchwork` objects for multiple output paths.

See Also

Other sensitivity-calculation: `sensitivityCalculation()`, `sensitivitySpiderPlot()`, `sensitivityTimeProfiles()`

Examples

```
## Not run:
simPath <- system.file("extdata", "Aciclovir.pkml", package = "ospsuite")
simulation <- loadSimulation(simPath)
outputPaths <- "Organism|PeripheralVenousBlood|Aciclovir|Plasma (Peripheral Venous Blood)"
parameterPaths <- c(
  "Aciclovir|Lipophilicity",
  "Events|IV 250mg 10min|Application_1|ProtocolSchemaItem|Dose",
 "Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Glomerular Filtration-GFR-Aciclovir|GFR fraction"
)

results <- sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = parameterPaths,
  variationRange = c(seq(0.1, 1, by = 0.1), seq(2, 10, by = 1)),
)

# Print plots with default settings
sensitivityTornadoPlot(results)

# Print plots with specific parameter scaling factor
sensitivityTornadoPlot(results, parameterFactor = 0.5)

# Print plots with custom configuration settings
myPlotConfiguration <- createEsqlabsPlotConfiguration()
myPlotConfiguration$legendPosition <- "bottom"
myPlotConfiguration$subtitle <- "Custom settings"
```

```
sensitivityTornadoPlot(results, defaultPlotConfiguration = myPlotConfiguration)

# Use named parameter paths to customize axis labels
namedParameterPaths <- c(
  "Lipophilicity" = "Aciclovir|Lipophilicity",
  "Dose" = "Events|IV 250mg 10min|Application_1|ProtocolSchemaItem|Dose",
 "GFR fraction" = "Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Glomerular Filtration-GFR-Aciclovir|GFR fraction
)

resultsNamed <- sensitivityCalculation(
  simulation = simulation,
  outputPaths = outputPaths,
  parameterPaths = namedParameterPaths
)

sensitivityTornadoPlot(resultsNamed)

## End(Not run)
```

`setApplications` *Set an application protocol in a* `Simulation` *from the Excel file*

Description

Set an application protocol in a `Simulation` from the Excel file

Usage

```
setApplications(simulation, scenarioConfiguration)
```

Arguments

`simulation`	A `Simulation` object that will be modified.
`scenarioConfiguration`	A `ScenarioConfiguration` object holding the name of the application protocol.

Details

Sets the parameter values describing the application protocol defined in the scenario configuration by reading from the Applications.xlsx file. The function looks for a sheet named after the application protocol and applies all parameter values found in that sheet to the simulation.

Deprecation

This function is deprecated. Use `setParametersFromXLS` instead for better parameter handling and more flexibility.

`setParameterValuesByPathWithCondition` *Set the values of parameters in the simulation by path, if the* `condition` *is true.*

Description

Set the values of parameters in the simulation by path, if the `condition` is true.

Usage

```
setParameterValuesByPathWithCondition(
  parameterPaths,
  values,
  simulation,
  condition = function(path) {
     TRUE
 },
  units = NULL
)
```

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 used to retrieve parameter instances from given paths.
`condition`	A function that receives a parameter path as an argument and returns `TRUE` of `FALSE`
`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 `parameterPaths`.

Examples

```
simPath <- system.file("extdata", "simple.pkml", package = "ospsuite")
sim <- loadSimulation(simPath)
condition <- function(path) {
  ospsuite::isExplicitFormulaByPath(
    path = path,
    simulation = sim
  )
}
setParameterValuesByPathWithCondition(
  c("Organism|Liver|Volume", "Organism|Volume"),
  c(2, 3),
```

```
  sim,
  condition
)
```

`snapshotProjectConfiguration` *Export project configuration Excel files to JSON*

Description

Exports all Excel configuration files in the Configurations folder of an esqlabsR project to a single JSON file. This allows for easier version control and programmatic manipulation.

Usage

```
snapshotProjectConfiguration(
  projectConfig = "ProjectConfiguration.xlsx",
  outputDir = NULL,
  ignoreVersionCheck = TRUE,
  ...
)
```

Arguments

`projectConfig`	A ProjectConfiguration object or path to ProjectConfiguration excel file. Defaults to "ProjectConfiguration.xlsx".
`outputDir`	Directory where the JSON file will be saved. If NULL (default), the JSON file will be created in the same directory as the source Excel file.
`ignoreVersionCheck`	Logical indicating whether to ignore version mismatch checks when creating the ProjectConfiguration from a file path. Defaults to TRUE.
`...`	Additional arguments.

Value

Invisibly returns the exported configuration data structure

See Also

Other project configuration snapshots: `projectConfigurationStatus()`, `restoreProjectConfiguration()`

sourceAll *Source all .R files located in a specific folder*

Description

Source all .R files located in a specific folder

Usage

```
sourceAll(folderPath, recursive = FALSE)
```

Arguments

folderPath	Path to the folder where .R files are located
recursive	If TRUE, the contents of the sub-folders are also sourced, otherwise only the files located directly in the directory are considered. Default is FALSE.

stringToNum *Convert string to numeric*

Description

Convert string to numeric

Usage

```
stringToNum(string, lloqMode = LLOQMode$`LLOQ/2`, uloqMode = ULOQMode$ULOQ)
```

Arguments

string	A string or a list of strings to be converted to numeric values
lloqMode	How to treat entries below LLOQ, i.e., of a form "<2": LLOQ/2 (default): return the number divided by 2, LLOQ: return the numerical value, ZERO: return 0, ignore: return NA
uloqMode	How to treat entries above ULOQ, i.e., of a form ">2": ULOQ: return the numerical value, ignore: return NA

Details

Tries to convert each string to a numeric with `as.numeric()`. If any conversion fails and returns NA, the value is tested for being a LLOQ- or a ULOQ value, i.e., of a form "<2" or ">2", respectively. If this is a case, the returned value is defined by the parameters lloqMode and uloqMode. In any other case where the string cannot be converted to a numeric, NA is returned.

Value

A numeric value or a list of numeric values

ULOQMode *Possible modes to treat values above the upper limit of quantification.*

Description

Possible modes to treat values above the upper limit of quantification.

Usage

```
ULOQMode
```

validateAllConfigurations *Validate all configuration files in a project*

Description

Validate all configuration files in a project

Usage

```
validateAllConfigurations(projectConfiguration, ignoreVersionCheck = TRUE)
```

Arguments

projectConfiguration	ProjectConfiguration object or path to ProjectConfiguration.xlsx
ignoreVersionCheck	Logical; if TRUE, skip project configuration version mismatch checks when creating a ProjectConfiguration from a path. Defaults to TRUE

Value

Named list of validationResult objects

validationResult *validationResult*

Description

R6 class for storing validation results from Excel configuration files

Public fields

`data` Successfully validated/processed data

`critical_errors` List of critical errors (blocking issues)

`warnings` List of warnings (non-blocking issues)

Methods

Public methods:

- `validationResult$new()`
- `validationResult$add_critical_error()`
- `validationResult$add_warning()`
- `validationResult$set_data()`
- `validationResult$is_valid()`
- `validationResult$has_critical_errors()`
- `validationResult$get_formatted_messages()`
- `validationResult$get_summary()`
- `validationResult$clone()`

`validationResult$new()`: Initialize a new ValidationResult

Usage:

```
validationResult$new()
```

`validationResult$add_critical_error()`: Add a critical error

Usage:

```
validationResult$add_critical_error(category, message, details = NULL)
```

Arguments:

`category` Error category (e.g., "Structure", "Missing Fields", "Uniqueness")

`message` Error message

`details` Optional list with additional details (sheet, row, column)

`validationResult$add_warning()`: Add a warning

Usage:

```
validationResult$add_warning(category, message, details = NULL)
```

Arguments:

`category` Warning category (e.g., "Data", "Structure")

`message` Warning message

`details` Optional list with additional details (sheet, row, column)

`validationResult$set_data()`: Set validated data

Usage:

```
validationResult$set_data(data)
```

Arguments:

`data` The validated/processed data to store

`validationResult$is_valid()`: Check if validation passed (no critical errors)

Usage:

```
validationResult$is_valid()
```

`validationResult$has_critical_errors()`: Check if there are critical errors

Usage:

```
validationResult$has_critical_errors()
```

`validationResult$get_formatted_messages()`: Get formatted messages for display

Usage:

```
validationResult$get_formatted_messages()
```

`validationResult$get_summary()`: Get validation summary

Usage:

```
validationResult$get_summary()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`validationSummary` *Get summary of all validation results*

Description

Get summary of all validation results

Usage

```
validationSummary(validationResults)
```

Arguments

validationResults
: Output from validateAllConfigurations

Value

List with summary statistics

---

writeIndividualToXLS *Create a parameter set describing an individual and write it to the Excel file*

Description

Create a parameter set describing an individual and write it to the Excel file

Usage

```
writeIndividualToXLS(individualCharacteristics, outputXLSPath)
```

Arguments

individualCharacteristics
: An IndividualCharacteristics object describing the individual. See createIndividualCharacters for more information.

outputXLSPath
: Path to the Excel file the parameter set will be written to

Value

Path to the created Excel file

See Also

createIndividualCharacteristics crateIndividual

Examples

```
## Not run:
simulation <- loadSimulation(pathToPKML)
humanIndividualCharacteristics <- createIndividualCharacteristics(
  species = Species$Human, population = HumanPopulation$European_ICRP_2002,
  gender = Gender$Male, weight = 70
)
writeIndividualToXLS(humanIndividualCharacteristics, pathToExcelFile)

## End(Not run)
```

writeParameterStructureToXLS *Write parameter structure to excel that can be loaded in MoBi*

Description

Write parameter structure to excel that can be loaded in MoBi

Usage

```
writeParameterStructureToXLS(
  parameterStructure,
  paramsXLSpath,
  sheet = NULL,
  append = FALSE
)
```

Arguments

parameterStructure	A list containing vectors `paths` with the full paths to the parameters, `values` the values of the parameters, and `units` with the units the values are in.
paramsXLSpath	Path to the excel file
sheet	(Optional) name of the excel sheet
append	If TRUE, the existing excel file/sheet will be appended with the new parameter structure. If FALSE (default), the existing file will be overwritten.

Examples

```
## Not run:
params <- list(
  paths = c("Container1|Path1", "Container|Second|Third|Path2"),
  values = c(1, 2),
  units = c("", "µmol")
)

writeParameterStructureToXLS(params, "test.xlsx")

## End(Not run)
```

Index