Package: ospsuite.reportingframework (via r-universe)

July 15, 2026

Type Package

Title Provides a framework for report generation with the ospsuite

Version 1.0.1

Description The OSP Suite Reporting Framework, which is based on the esqlabsR package, is a robust tool designed to systematically define simulation scenarios, their inputs and outputs, and standardize data import and matching processes. These functionalities allow for easy and automatable model execution and plot generation, ultimately facilitating the generation of customizable reports for OSP physiologically-based pharmacokinetic (PBPK) models. As a result, this framework significantly enhances reporting efficiency, enabling researchers to create clear, tailored reports that effectively communicate complex modeling results.

URL https://www.open-systems-pharmacology.org/OSPSuite.ReportingFramework/

License GPL-2 | file LICENSE

Language en-US

Encoding UTF-8

LazyData true

Imports data.table, checkmate, utils, knitr, rmarkdown, fs, ospsuite, ospsuite.utils, ggplot2, ggsci, scales, tidyr, rlang, openxlsx, dplyr, stats, foreach, boot, doParallel, jsonlite

Depends esqlabsR, ospsuite.plots, R (>= 2.10)

Suggests rstudioapi, styler, cowplot, spelling, vdiffr, withr, devtools, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

RoxygenNote 7.3.1

Remotes ospsuite.utils=Open-Systems-Pharmacology/OSPSuite.RUtils, ospsuite.plots=Open-Systems-Pharmacology/OSPSuite.Plots, ospsuite=Open-Systems-Pharmacology/OSPSuite-R, esqlabsR=esqLABS/esqlabsR

Config/pak/sysreqs cmake dotnet-runtime-8.0 git make libgit2-dev libicu-dev libjpeg-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libx11-dev zlib1g-dev

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

Date/Publication 2025-10-09 06:15:07 UTC

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

RemoteRef v1.0.1

RemoteSha d5b92de0e943d10fd6a18e441b4e4870854e5348

Contents

`addBiometricsToConfig` *Add biometrics information to config*

Description

Add biometrics information to config

Usage

```
addBiometricsToConfig(projectConfiguration, dataDT, overwrite = FALSE)
```

Arguments

projectConfiguration	Object of class 'ProjectConfiguration' containing information on paths and file names
dataDT	A 'data.table' with observed data.
overwrite	If TRUE, existing rows will be overwritten.

See Also

Other observed data processing: aggregateObservedDataGroups(), convertDataCombinedToDataTable(), convertDataTableToDataCombined(), readObservedDataByDictionary(), updateDataGroupId(), updateOutputPathId(), validateObservedData()

---

addDefaultConfigForDistributionsVsDemographics

Add Default Configuration for for distribution vs demographics plots

---

Description

This function adds a default configuration sheet for distribution vs demographics plots to the plot configuration table. It can either create a new sheet or overwrite an existing one based on the specified parameters.

Usage

```
addDefaultConfigForDistributionsVsDemographics(
  projectConfiguration,
  pkParameterDT = NULL,
  sheetName = "DistributionVsRange",
  overwrite = FALSE
)
```

Arguments

projectConfiguration	A ProjectConfiguration class object containing configuration details, including: - 'plotsFile': A string representing the file path to the Excel workbook containing the plot configurations.
pkParameterDT	Optional. A data object containing pkParameter.
sheetName	A character string specifying the name of the sheet in the plot configuration table. Default is 'DistributionVsRange'.
overwrite	A boolean indicating whether existing configurations should be overwritten. Default is FALSE.

Details

The function retrieves scenario definitions, output path IDs, and data groups from the project configuration. It checks if the specified sheet already exists and whether to overwrite it. If not, it creates a new header and fills in the default configuration values for the histograms.

Additionally, the function performs a validity check to ensure that it is not executed during a context where helper functions are prohibited (validRun). If such a context is detected, an error is raised to prevent execution.

Value

NULL This function updates the Excel workbook in place and does not return a value.

See Also

Other plot configuration helper function: `addDefaultConfigForHistograms()`, `addDefaultConfigForPKBoxwhsikerPlot`
`addDefaultConfigForPKForestPlots()`, `addDefaultConfigForTimeProfilePlots()`

Other functions to generate histograms: `plotHistograms()`, `validateHistogramsConfig()`

`addDefaultConfigForHistograms`

Add Default Configuration for Histograms

Description

This function adds a default configuration sheet for histograms to the plot configuration table. It can either create a new sheet or overwrite an existing one based on the specified parameters.

Usage

```
addDefaultConfigForHistograms(
  projectConfiguration,
  pkParameterDT = NULL,
  sheetName = "Histograms",
  overwrite = FALSE
)
```

Arguments

`projectConfiguration`	A ProjectConfiguration class object containing configuration details, including: - 'plotsFile': A string representing the file path to the Excel workbook containing the plot configurations.
`pkParameterDT`	Optional. A data object containing pkParameter.
`sheetName`	A character string specifying the name of the sheet in the plot configuration table. Default is "Histograms".
`overwrite`	A boolean indicating whether existing configurations should be overwritten. Default is FALSE.

Details

The function retrieves scenario definitions, output path IDs, and data groups from the project configuration. It checks if the specified sheet already exists and whether to overwrite it. If not, it creates a new header and fills in the default configuration values for the histograms.

Additionally, the function performs a validity check to ensure that it is not executed during a context where helper functions are prohibited (validRun). If such a context is detected, an error is raised to prevent execution.

Value

NULL This function updates the Excel workbook in place and does not return a value. It is called for its side effects. Add Default Configuration for Histograms

See Also

Other plot configuration helper function: `addDefaultConfigForDistributionsVsDemographics()`, `addDefaultConfigForPKBoxwhsikerPlots()`, `addDefaultConfigForPKForestPlots()`, `addDefaultConfigForTimePr`

Other functions to generate plots displaying distribution vs demographics: `plotDistributionVsDemographics()`, `validateDistributionVsDemographicsConfig()`

`addDefaultConfigForPKBoxwhsikerPlots` *Add Default Configuration for Box-and-Whisker Plots*

Description

This function adds a default configuration sheet for box-and-whisker plots to the plot configuration table. It can either create a new sheet or overwrite an existing one based on the specified parameters.

Usage

```
addDefaultConfigForPKBoxwhsikerPlots(
  projectConfiguration,
  pkParameterDT,
  sheetName = "PKParameter_Boxplot",
  overwrite = FALSE
)
```

Arguments

`projectConfiguration`	A ProjectConfiguration class object containing configuration details, including: - 'plotsFile': A string representing the file path to the Excel workbook containing the plot configurations.
`pkParameterDT`	A data.table containing PK parameter data.

sheetName	A character string specifying the name of the sheet in the plot configuration table. Default is "PKParameter_Boxplot".
overwrite	A boolean indicating whether existing configurations should be overwritten. Default is FALSE.

Details

The function retrieves scenario definitions, output path IDs, and data groups from the project configuration. It checks if the specified sheet already exists and whether to overwrite it. If not, it creates a new header and fills in the default configuration values for the time profile plots.

Additionally, the function performs a validity check to ensure that it is not executed during a context where helper functions are prohibited (‘validRun‘). If such a context is detected, an error is raised to prevent execution.

Value

NULL This function updates the Excel workbook in place and does not return a value. It is called for its side effects.

See Also

Other plot configuration helper function: `addDefaultConfigForDistributionsVsDemographics()`, `addDefaultConfigForHistograms()`, `addDefaultConfigForPKForestPlots()`, `addDefaultConfigForTimeProfilePl`

Other functions to generate box whisker plots: `plotPKBoxwhisker()`, `validatePKBoxwhiskerConfig()`

`addDefaultConfigForPKForestPlots` *Add Default Configuration for PK Forest Plots*

Description

Adds default configurations for forest plots to the ‘Plots.xlsx‘ configuration file.

Usage

```
addDefaultConfigForPKForestPlots(
  projectConfiguration,
  pkParameterDT,
  sheetName = "PKParameter_Forest",
  overwrite = FALSE
)
```

Arguments

projectConfiguration	A ProjectConfiguration object.
pkParameterDT	A data.table containing PK parameter data.
sheetName	Name of the sheet to create.
overwrite	Logical indicating if existing data should be overwritten.

Value

NULL (invisible).

See Also

plotPKForestAggregatedAbsoluteValues, plotPKForestPointEstimateOfAbsoluteValues, plotPKForestAggregate
plotPKForestPointEstimateOfRatios,

Other plot functions: addDefaultConfigForTimeProfilePlots(), plotDistributionVsDemographics(),
plotHistograms(), plotPKBoxwhisker(), plotPKForestAggregatedAbsoluteValues(), plotPKForestAggregatedRa
plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimateOfRatios(),
plotSensitivity(), plotTimeProfiles()

Other plot configuration helper function: addDefaultConfigForDistributionsVsDemographics(),
addDefaultConfigForHistograms(), addDefaultConfigForPKBoxwhsikerPlots(), addDefaultConfigForTimeProfil

Other functions to generate forest plots: plotPKForestAggregatedAbsoluteValues(), plotPKForestAggregatedRatios
plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimateOfRatios(),
validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestAggregatedRatiosConfig(),
validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig()

---

addDefaultConfigForTimeProfilePlots
Add Default Configuration for Time Profile Plots

Description

This function adds a default configuration sheet for time profile plots to the plot configuration table. It can either create a new sheet or overwrite an existing one based on the specified parameters.

Usage

```
addDefaultConfigForTimeProfilePlots(
  projectConfiguration,
  dataObserved = NULL,
  sheetName = "TimeProfiles",
  overwrite = FALSE
)
```

Arguments

projectConfiguration	A 'ProjectConfiguration' class object containing configuration details, including: - 'plotsFile': A string representing the file path to the Excel workbook containing the plot configurations.
dataObserved	Optional. A data object containing observed data, if available.
sheetName	A character string specifying the name of the sheet in the plot configuration table. Default is 'TimeProfiles'.
overwrite	A boolean indicating whether existing configurations should be overwritten. Default is FALSE.

Details

The function retrieves scenario definitions, output path IDs, and data groups from the project configuration. It checks if the specified sheet already exists and whether to overwrite it. If not, it creates a new header and fills in the default configuration values for the time profile plots.

Additionally, the function performs a validity check to ensure that it is not executed during a context where helper functions are prohibited ('validRun'). If such a context is detected, an error is raised to prevent execution.

Value

NULL This function updates the Excel workbook in place and does not return a value. It is called for its side effects.

See Also

Other plot functions: addDefaultConfigForPKForestPlots(), plotDistributionVsDemographics(), plotHistograms(), plotPKBoxwhisker(), plotPKForestAggregatedAbsoluteValues(), plotPKForestAggregatedRat plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimateOfRatios(), plotSensitivity(), plotTimeProfiles()

Other plot configuration helper function: addDefaultConfigForDistributionsVsDemographics(), addDefaultConfigForHistograms(), addDefaultConfigForPKBoxwhsikerPlots(), addDefaultConfigForPKForestPl

---

addFiguresAndTables *Loop on figure keys, add figure or table*

Description

Loop on figure keys, add figure or table

Usage

```
addFiguresAndTables(
  keyList,
  subfolder,
  numbersOf,
  customStyles = list(),
  digitsOfSignificance = 3
)
```

Arguments

keyList	list of keys
subfolder	sub-folder of relative to .Rmd file where figures are saved
numbersOf	list with numbers of tables and figures in rmd
customStyles	custom-styles to render word document
digitsOfSignificance	significance digits for tables

Value

list of numbers and figures after loop

See Also

Other markdown helper function: mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdHeading(), mdLink(), mdNewline(), mdNewpage(), mdPaste(), mdPaste0(), mdTable()

---

addMessageToLog *Used to add message to log file*

Description

This function is for the usage outside a ‘captureLog‘ bracket. Inside the bracket ‘message("my message Text")‘ can be used

Usage

```
addMessageToLog(messageText)
```

Arguments

messageText	character with message text

See Also

Other log file management: captureLog(), initLogfunction(), saveSessionInfo(), setShowLogMessages(), writeTableToLog(), writeToLog()

addPredictedValues *Interpolates observed data based on simulated data.*

Description

This function performs linear extrapolation for increasing yValues and logarithmic interpolation for decreasing yValues based on the specified grouping identifiers. It modifies the input dtObserved data.table by adding a new column with the predicted values.

Usage

```
addPredictedValues(dtObserved, dtSimulated, identifier)
```

Arguments

dtObserved	A data.table containing observed data with columns 'xValues', 'yValues', and grouping identifiers.
dtSimulated	A data.table containing simulated data with columns 'xValues' and 'yValues'.
identifier	A character vector of column names used for grouping in the extrapolation and interpolation process.

Value

A data.table with an additional column named 'predicted' containing the extrapolated and interpolated values.

addSensitivityTable *Add Sensitivity Table to Project Configuration*

Description

This function adds a sensitivity table to the provided project configuration. It loads a template Excel file and populates it with parameter paths from the specified scenario.

Usage

```
addSensitivityTable(
  projectConfiguration,
  scenarioList = NULL,
  scenarioName,
  sheetName = scenarioName
)
```

Arguments

projectConfiguration	An object representing the project configuration.
scenarioList	A list of scenarios from which to extract parameter paths. Defaults to NULL.
scenarioName	The name of the scenario to extract parameter paths from.
sheetName	The name of the sheet in the Excel file to populate. Defaults to the value of scenarioName.

Value

The modified project configuration object.

---

aggregateObservedDataGroups
Aggregate Observed Data Groups

Description

This function aggregates observed data based on specified groups and an aggregation method. It allows for different aggregation techniques, including geometric and arithmetic standard deviations, percentiles, or a user-defined custom function.

Usage

```
aggregateObservedDataGroups(
  dataObserved,
  groups = NULL,
 aggregationFlag = c("GeometricStdDev", "ArithmeticStdDev", "Percentiles", "Custom"),
 percentiles = getOspsuite.plots.option(optionKey = OptionKeys$Percentiles)[c(1, 3, 5)],
  groupSuffix = "aggregated",
  customFunction = NULL,
  lloqCheckColumns2of3 = NULL,
  lloqCheckColumns1of2 = NULL
)
```

Arguments

dataObserved	A ‘data.table‘ containing observed data.
groups	A character vector specifying the groups to aggregate. If NULL, all available groups are used.
aggregationFlag	A character string indicating the aggregation method. Options include "GeometricStdDev", "ArithmeticStdDev", "Percentiles", or "Custom".
percentiles	A numeric vector of percentiles to calculate if ‘aggregationFlag‘ is "Percentiles". Default is c(5, 50, 95).

`groupSuffix` A character string to append to group names in the aggregated output. Default is 'aggregated'.

`customFunction` A custom function for aggregation if 'aggregationFlag' is "Custom". Default is NULL.

`lloqCheckColumns2of3`
A character vector specifying columns to check for LLOQ (Lower Limit of Quantification) for 1/3 data points. Default is NULL, is used only for 'aggregationFlag' "Custom".

`lloqCheckColumns1of2`
A character vector specifying columns to check for LLOQ for 2/3 data points. Default is NULL, is used only for 'aggregationFlag' "Custom".

Details

The function also checks for values below the Lower Limit of Quantification (LLOQ) and adjusts the aggregated results accordingly. For aggregationFlag 'GeometricStdDev' and 'GeometricStdDev', the statistics at any time point will only be calculated if at least 2/3 of the individual data were measured and were above the lower limit of quantification (lloq). For aggregationFlag 'Percentile', the statistics will only be calculated if less than or equal to 1/2 of the data is above LLOQ. If you use aggregationFlag 'Custom', please set parameters lloqCheckColumns2of3 and lloqCheckColumns1of2 accordingly.

A custom function should take a numeric vector 'y' as input and return a list containing: - 'yValues': The aggregated value (e.g., mean). - 'yMin': The lower value of the aggregated data (e.g., mean - sd). - 'yMax': The upper value of the aggregated data (e.g., mean + sd). - 'yErrorType': A string indicating the type of error associated with the aggregation, it is used in plot legends and captions. It must be a concatenation of the descriptor of yValues and the descriptor of yMin - yMax range separated by "|" (e.g., "mean | standard deviation" or "median | 5th - 95th percentile").

Value

A 'data.table' containing aggregated observed data.

See Also

Other observed data processing: `addBiometricsToConfig()`, `convertDataCombinedToDataTable()`, `convertDataTableToDataCombined()`, `readObservedDataByDictionary()`, `updateDataGroupId()`, `updateOutputPathId()`, `validateObservedData()`

BIOMETRICUNITS *enumeration biometric units*

Description

enumeration biometric units

Usage

```
BIOMETRICUNITS
```

Format

An object of class `list` of length 3.

See Also

Other enumerations: `DATACLASS`, `EXPORTDIR`, `TIMERANGE`

`calculatePKParameterForScenarios` *Calculate Pharmacokinetic (PK) Parameters*

Description

This function calculates pharmacokinetic (PK) parameters for specified scenarios based on project configuration and simulation results. It generates output files for each scenario in the designated output folder.

Usage

```
calculatePKParameterForScenarios(projectConfiguration, scenarioResults)
```

Arguments

`projectConfiguration`
: A list containing project configuration settings, including the output folder path and the PK parameter file.

`scenarioResults`
: A named list of scenario results, each containing simulation data for PK analysis.

Value

This function is called for its side effects and does not return a value.

See Also

Other scenario management: `createScenarios.wrapped()`, `loadPKParameter()`, `loadScenarioResultsToFramework()`
`runAndSaveScenarios()`, `runOrLoadScenarios()`

captureLog *function that catches messages, warnings, and errors. This function has to be initialized by 'initLogfunction' function*

Description

function that catches messages, warnings, and errors. This function has to be initialized by 'initLogfunction' function

Usage

```
captureLog(expr, finallyExpression = invisible())
```

Arguments

expr	The expression to evaluate.
finallyExpression	The expression to evaluate finally

See Also

Other log file management: addMessageToLog(), initLogfunction(), saveSessionInfo(), setShowLogMessages(), writeTableToLog(), writeToLog()

convertDataCombinedToDataTable *Converts object of class 'DataCombined' to data.table with attributes.*

Description

Format corresponds to 'data.table' produced by 'readObservedDataByDictionary'.

Usage

```
convertDataCombinedToDataTable(datacombined)
```

Arguments

datacombined	The input 'DataCombined' object.

Value

A 'data.table' containing the converted data.

See Also

Other observed data processing: addBiometricsToConfig(), aggregateObservedDataGroups(), convertDataTableToDataCombined(), readObservedDataByDictionary(), updateDataGroupId(), updateOutputPathId(), validateObservedData()

`convertDataTableToDataCombined` *Converts data.table with observed data to 'ospsuite::DataCombined' object*

Description

The 'data.table' must be formatted like a table produced by 'readObservedDataByDictionary'.

Usage

```
convertDataTableToDataCombined(dataDT)
```

Arguments

`dataDT`	A 'data.table' to convert.

Value

An object of class 'DataCombined'.

See Also

Other observed data processing: addBiometricsToConfig(), aggregateObservedDataGroups(), convertDataCombinedToDataTable(), readObservedDataByDictionary(), updateDataGroupId(), updateOutputPathId(), validateObservedData()

`createDocumentFromTemplate` *Opens a template file a new document*

Description

Opens a template file a new document

Usage

```
createDocumentFromTemplate(
  template = "template_workflow",
 templatePath = system.file("templates", package = "ospsuite.reportingframework")
)
```

Arguments

`template`	name of template script
`templatePath`	path of template script

See Also

Other Addins Support: `createWorkflowTemplate()`, `openEPackageTemplate()`, `openFigureTemplate()`, `openWorkflowTemplate()`

`createProjectConfiguration` *#' Create a 'ProjectConfiguration'*

Description

Create a 'ProjectConfigurationRF' based on the '"ProjectConfiguration.xlsx"'

based on esqlabsR::ProjectConfiguration but with additional file information for PK Parameter definitions

Usage

```
createProjectConfiguration(path = file.path("ProjectConfiguration.xlsx"))
```

Arguments

`path`	path to the 'ProjectConfiguration.xlsx' file. default to the 'ProjectConfiguration.xlsx' file located in the working directory.

Value

Object of type 'ProjectConfigurationRF'

See Also

Other project initialization: `ProjectConfigurationRF`, `getQCpassedEnvironmentVariable()`, `initProject()`, `setWorkflowOptions()`

createScenarios.wrapped

Create Scenario objects from 'ScenarioConfiguration' objects

Description

wrap of 'esqlabsR::createDefaultProjectConfiguration()' with 'esqlabsR::createScenarios()' as input

Usage

```
createScenarios.wrapped(projectConfiguration, scenarioNames = NULL)
```

Arguments

projectConfiguration	Object of class 'ProjectConfiguration' containing information on paths and file names
scenarioNames	Names of the scenarios that are defined in the excel file. If NULL (default), all scenarios specified in the excel file will be created.

Value

Named list of Scenario objects.

See Also

Other scenario management: calculatePKParameterForScenarios(), loadPKParameter(), loadScenarioResultsToFr runAndSaveScenarios(), runOrLoadScenarios()

createWorkflowTemplate

Opens the workflow template as new document

Description

Opens the workflow template as new document

Usage

```
createWorkflowTemplate()
```

See Also

Other Addins Support: createDocumentFromTemplate(), openEPackageTemplate(), openFigureTemplate(), openWorkflowTemplate()

DATACLASS *enumeration keys for 'DataClass'*

Description

enumeration keys for 'DataClass'

Usage

```
DATACLASS
```

Format

An object of class `list` of length 5.

See Also

Other enumerations: BIOMETRICUNITS, EXPORTDIR, TIMERANGE

EXPORTDIR *enumeration keys for exportDirectories*

Description

enumeration keys for exportDirectories

Usage

```
EXPORTDIR
```

Format

An object of class `list` of length 3.

See Also

Other enumerations: BIOMETRICUNITS, DATACLASS, TIMERANGE

exportRandomPopulations *Export Random Populations*

Description

This function generates virtual populations based on demographic data and exports them to CSV files.

Usage

```
exportRandomPopulations(
  projectConfiguration,
  populationNames = NULL,
  customParameters = NULL,
  overwrite = FALSE
)
```

Arguments

projectConfiguration	A list containing project configuration details, including: - populationsFile: Path to the Excel file containing population demographics. - populationsFolder: Directory where the generated CSV files will be saved.
populationNames	A character vector of population names to generate. If NULL, all populations in the demographics sheet will be considered.
customParameters	A list of custom parameters to set for the populations. Each item in the list should be a list containing: - path: The parameter path to set. - values: A vector of values to assign to the parameter.
overwrite	A logical indicating whether to overwrite existing population files. Default is FALSE.

Value

This function does not return a value. It generates and exports CSV files for the specified populations.

See Also

Other population export: exportVirtualTwinPopulations(), setupVirtualTwinPopConfig()

Examples

```
## Not run:
exportRandomPopulations(projectConfiguration,
  populationNames = c("Population1", "Population2"),
  customParameters = list(list(path = "param1", values = c(1, 2))),
  overwrite = TRUE
)

## End(Not run)
```

`exportSimulationWorkflowToEPackage` *Export Simulation Workflow*

Description

This function facilitates the export of a simulation workflow to an electronic package (ePackage). It initializes a 'WorkflowScriptExporter' object and executes a series of operations to prepare the workflow for export, including generating workflow text, retrieving input files, configuring scenario details, and exporting necessary files.

Usage

```
exportSimulationWorkflowToEPackage(
  projectConfiguration,
  wfIdentifier,
  scenarioNames,
  fileNameReplacements = c()
)
```

Arguments

`projectConfiguration`	An object of class 'projectConfiguration' that contains various configuration details, including paths for input files, output directories, and any other relevant settings required for the workflow export.
`wfIdentifier`	A unique identifier (integer) for the workflow. This identifier is used to distinguish this workflow from others and is included in the names of the exported files.
`scenarioNames`	A character vector of scenario names that should be included in the workflow export. These scenarios determine which specific configurations and input files will be processed and exported as part of the ePackage.
`fileNameReplacements`	An optional character vector that specifies replacements for file names. The vector should contain pairs of values, where the first value in each pair represents the original file name, and the second value represents the desired replacement

name. This allows for customization of exported file names to meet specific naming conventions or requirements. For Files not listed in this variable the source filename is converted to a valid filename.

Value

Invisible NULL. The function does not return any value but performs a series of operations that result in the creation of exported files and configurations necessary for the ePackage. If successful, the function will complete without errors; otherwise, it will raise an informative error.

See Also

Other electronic package: `exportTLFWorkflowToEPackage()`, `importWorkflow()`

Examples

```
## Not run:
# Example usage of exportSimulationWorkflow function
exportSimulationWorkflow(
  projectConfiguration = projectConfiguration,
  wfIdentifier = 1,
  scenarioNames = c("Scenario1", "Scenario2"),
  fileNameReplacements = c("oldName.pkml", "new_name.pkml")
)

## End(Not run)
```

`exportTLFWorkflowToEPackage`

Export TLF Workflow to E-Package

Description

This function facilitates the export of a TLF (Table-Linked Format) workflow to an electronic package (ePackage). It initializes a 'WorkflowScriptExporter' object, extracts code chunks from the provided R Markdown file, evaluates the code chunks, and executes a series of operations to prepare the workflow for export. This includes generating workflow text, retrieving input files, configuring scenario details, and exporting necessary files.

Usage

```
exportTLFWorkflowToEPackage(
  projectConfiguration,
  wfIdentifier,
  workflowRmd,
  fileNameReplacements = c()
)
```

Arguments

`projectConfiguration`
: An object of class 'projectConfiguration' that contains various configuration details, including paths for input files, output directories, and any other relevant settings required for the workflow export.

`wfIdentifier`
: A unique identifier (integer) for the workflow. This identifier is used to distinguish this workflow from others and is included in the names of the exported files.

`workflowRmd`
: Path to the input markdown file for the TLF workflow. This file should contain the necessary code chunks that are to be extracted and evaluated for the export process.

`fileNameReplacements`
: An optional character vector that specifies replacements for file names. The vector should contain pairs of values, where the first value in each pair represents the original file name, and the second value represents the desired replacement name. This allows for customization of exported file names to meet specific naming conventions or requirements. For files not listed in this variable, the source filename is converted to a valid filename.

Value

Invisible NULL. The function does not return any value but performs a series of operations that result in the creation of exported files and configurations necessary for the ePackage. If successful, the function will complete without errors; otherwise, it will raise an informative error.

See Also

Other electronic package: `exportSimulationWorkflowToEPackage()`, `importWorkflow()`

Examples

```
## Not run:
# Example usage of exportTLFWorkflowToEPackage function
exportTLFWorkflowToEPackage(
  projectConfiguration = projectConfiguration,
  wfIdentifier = 1,
  workflowRmd = "path/to/workflow.Rmd",
  fileNameReplacements = c("oldName.pkml", "new_name.pkml")
)

## End(Not run)
```

`exportVirtualTwinPopulations` *Generate Virtual Twin Population*

Description

This function generates virtual twin populations based on the provided model and project configuration.

Usage

```
exportVirtualTwinPopulations(
  projectConfiguration,
  modelFile,
  overwrite = FALSE,
  populationNames = NULL
)
```

Arguments

`projectConfiguration`	A list containing project configuration details, including file paths for populations and scenarios.
`modelFile`	A string representing the name of the model file to be loaded. This file is used for unit conversion
`overwrite`	A logical indicating whether to overwrite existing files. Defaults to FALSE
`populationNames`	a character vector defining the population which should be exported. If NULL (default) all will be exported

See Also

Other population export: `exportRandomPopulations()`, `setupVirtualTwinPopConfig()`

`formatPercentiles` *This function takes a numeric vector of percentiles and maps specific values to their corresponding labels. It also formats other numeric values based on whether they are integers or not.*

Description

This function takes a numeric vector of percentiles and maps specific values to their corresponding labels. It also formats other numeric values based on whether they are integers or not.

Usage

```
formatPercentiles(percentiles, suffix = "", allAsPercentiles = FALSE)
```

Arguments

percentiles	A numeric vector of percentiles (0, 50, 100, and other values).
suffix	A character string to append to formatted percentile values.
allAsPercentiles	boolean, if FALSE for percentiles = 0,50,100 min, median and max is returned otherwise 0th 50th 100th

Value

A vector containing the mapped labels for specific percentiles and formatted strings for others.

---

getDataGroups *Load the Properties for Data Groups*

Description

This function loads the properties for data groups from the specified workbook.

Usage

```
getDataGroups(wbPlots)
```

Arguments

wbPlots	The path to the workbook containing the data groups.

Value

A 'data.table' with data group IDs.

See Also

Other get identifier: getModelParameterDefinitions(), getOutputPathIds(), getScenarioDefinitions(), getTimeRangeTags(), loadConfigTableEnvironment()

getModelParameterDefinitions *Load the Model Parameter Definitions*

Description

This function loads the model parameter definitions from the specified workbook.

Usage

```
getModelParameterDefinitions(wbPlots)
```

Arguments

wbPlots	The path to the workbook containing the model parameter definitions.

Value

A 'data.table' with model parameter definitions.

See Also

Other get identifier: getDataGroups(), getOutputPathIds(), getScenarioDefinitions(), getTimeRangeTags(), loadConfigTableEnvironment()

getOutputPathIds *Load the Output Configurations*

Description

This function loads the output configurations from the specified workbook.

Usage

```
getOutputPathIds(wbPlots)
```

Arguments

wbPlots	The path to the workbook containing the output configurations.

Value

A 'data.table' with output configurations.

See Also

Other get identifier: getDataGroups(), getModelParameterDefinitions(), getScenarioDefinitions(), getTimeRangeTags(), loadConfigTableEnvironment()

getQCpassedEnvironmentVariable

Get QC Passed Environment Variable

Description

This function retrieves the value of the environment variable 'QCpassed'. It attempts to convert the value to a logical type. If the environment variable is not set, is empty, or is non-logical, a warning is issued and the function defaults the value to 'FALSE'.

Usage

```
getQCpassedEnvironmentVariable()
```

Value

A logical value indicating whether the QC passed ('TRUE' or 'FALSE').

See Also

Other project initialization: `ProjectConfigurationRF`, `createProjectConfiguration()`, `initProject()`, `setWorkflowOptions()`

Examples

```
## Not run:
# Set the environment variable for testing
Sys.setenv(QCpassed = "TRUE")
getQCpassedEnvironmentVariable() # Should return TRUE

# Unset the environment variable for testing
Sys.unsetenv("QCpassed")
getQCpassedEnvironmentVariable() # Should return FALSE and issue a warning

## End(Not run)
```

getScenarioDefinitions

Load the Scenario Definitions

Description

This function loads the scenario definitions from the specified workbooks.

Usage

```
getScenarioDefinitions(wbScenarios, wbPlots = NULL)
```

Arguments

wbScenarios	The path to the workbook containing additional scenario definitions.
wbPlots	The path to the workbook containing the scenario definitions.

Value

A 'data.table' with scenario definitions.

See Also

Other get identifier: getDataGroups(), getModelParameterDefinitions(), getOutputPathIds(), getTimeRangeTags(), loadConfigTableEnvironment()

---

getTimeRangeTags *Load the Time Range Tags*

Description

This function loads the time range tags from the specified workbook.

Usage

```
getTimeRangeTags(wbPlots)
```

Arguments

wbPlots	The path to the workbook containing the time range tags.

Value

A 'data.table' with time range tags.

See Also

Other get identifier: getDataGroups(), getModelParameterDefinitions(), getOutputPathIds(), getScenarioDefinitions(), loadConfigTableEnvironment()

`importWorkflow` *Import Workflow*

Description

This function imports a workflow from an electronic package (ePackage) into a project directory. It initializes the project configuration, retrieves relevant project files, and synchronizes scenarios with plots. The function ensures that the necessary directories exist and that the configuration is set up correctly for further processing.

Usage

```
importWorkflow(
  projectDirectory,
  wfIdentifier,
  ePackageFolder,
  configurationDirectory
)
```

Arguments

`projectDirectory`	A character string representing the path to the project directory where the workflow will be imported.
`wfIdentifier`	An integer identifier for the workflow being imported. This identifier is used to distinguish between different workflows.
`ePackageFolder`	A character string representing the path to the electronic package folder that contains the workflow files and configurations to be imported.
`configurationDirectory`	A character string representing the directory where the configuration files will be stored.

Value

An object containing the updated project configuration after the workflow has been imported.

See Also

Other electronic package: `exportSimulationWorkflowToEPackage()`, `exportTLFWorkflowToEPackage()`

initLogfunction *Initializing a Log Function*

Description

This function initialize the logging during a workflow. It is called at the start of the workflow script. It is used to configure options for the log file folder, warnings which should not logged and messages which should not logged.

Usage

```
initLogfunction(
  projectConfiguration,
  warningsNotDisplayed = c("introduced infinite values",
   "Each group consists of only one observation", "rows containing non-finite values",
    "rows containing missing values", "Ignoring unknown parameters",
   "was deprecated in ggplot2", "font family not found in Windows font database",
    "mbcsToSbcs"),
  messagesNotDisplayed = c("Each group consists of only one observation"),
  verbose = TRUE
)
```

Arguments

projectConfiguration	Object of class 'ProjectConfiguration' containing information on paths and file names
warningsNotDisplayed	A list of warnings that should not be logged.
messagesNotDisplayed	A list of messages that should not be logged.
verbose	boolean, if true log message will be shown on the console

See Also

Other log file management: addMessageToLog(), captureLog(), saveSessionInfo(), setShowLogMessages(), writeTableToLog(), writeToLog()

Examples

```
## Not run:
# Initialize the log function
logFunction <- initLogfunction(rootDirectory = "path/to/project")

## End(Not run)
```

`initProject` *Initialize Project Directory*

Description

This function initializes a project directory by creating necessary subdirectories and copying configuration files from a source Excel file. It reads the directory structure and files specified in the provided Excel template and sets up the project environment accordingly.

Usage

```
initProject(
  configurationDirectory = ".",
 sourceConfigurationXlsx = system.file("templates", "ProjectConfiguration.xlsx", package
    = "ospsuite.reportingframework"),
 templatePath = system.file("templates", package = "ospsuite.reportingframework"),
  overwrite = FALSE
)
```

Arguments

`configurationDirectory`	A character string specifying the path to the project directory to be initialized. Defaults to the current working directory ('.').
`sourceConfigurationXlsx`	A character string representing the path to the source Excel file containing the project configuration. By default, it uses the template provided by the 'ospsuite.reportingframework' package.
`templatePath`	path of all template files
`overwrite`	A logical value indicating whether to overwrite existing files in the project directory. Defaults to FALSE, meaning existing files will not be overwritten.

Value

This function returns an invisible NULL. It is used for its side effects of creating directories and copying files rather than producing a value.

See Also

Other project initialization: `ProjectConfigurationRF`, `createProjectConfiguration()`, `getQCpassedEnvironmentVar`
`setWorkflowOptions()`

loadConfigTableEnvironment *Load Configuration Tables*

Description

This function loads various configuration tables from specified Excel files into a global environment.

Usage

```
loadConfigTableEnvironment(projectConfiguration)
```

Arguments

`projectConfiguration`
: An object of class 'ProjectConfiguration' containing paths to Excel files.

Value

An invisible value indicating that the loading was successful.

See Also

Other get identifier: `getDataGroups()`, `getModelParameterDefinitions()`, `getOutputPathIds()`, `getScenarioDefinitions()`, `getTimeRangeTags()`

loadPKParameter *Load Pharmacokinetic (PK) Parameters for Specified Scenarios*

Description

This function loads pharmacokinetic (PK) parameters for specified scenarios based on project configuration and a list of scenarios. It processes each scenario and returns the results as a data.table.

Usage

```
loadPKParameter(projectConfiguration, scenarioListOrResult)
```

Arguments

`projectConfiguration`
: A list containing project configuration settings, including the PK parameter file.

`scenarioListOrResult`
: A named list of scenarios for which PK parameters are to be loaded.

Value

A data.table containing the processed PK analyses for all specified scenarios.

A data.table containing the processed PK analyses.

See Also

Other scenario management: `calculatePKParameterForScenarios()`, `createScenarios.wrapped()`, `loadScenarioResultsToFramework()`, `runAndSaveScenarios()`, `runOrLoadScenarios()`

---

loadScenarioResultsToFramework

Load existing scenario results

Description

This function loads the results of specified scenarios. If the results do not exist, it returns an error.

Usage

```
loadScenarioResultsToFramework(projectConfiguration, scenarioNames)
```

Arguments

`projectConfiguration`	Configuration for the project, containing paths and settings necessary to load the results.
`scenarioNames`	Character vector of the names of the scenarios whose results are to be loaded.

Value

A list containing the loaded scenario results, including population data if available. throws Error if the scenario results do not exist.

See Also

Other scenario management: `calculatePKParameterForScenarios()`, `createScenarios.wrapped()`, `loadPKParameter()`, `runAndSaveScenarios()`, `runOrLoadScenarios()`

`mdBullet` *Insert a bulleted item, making sure that there are sufficient newlines to build a bulleted list*

Description

Insert a bulleted item, making sure that there are sufficient newlines to build a bulleted list

Usage

```
mdBullet(..., bullet = "-", level = 1)
```

Arguments

`...`	passed to ‘mdPaste‘
`bullet`	‘character‘ type of bullet
`level`	‘integer‘, specifies the indentation level. ‘level=1‘ has no indentation

See Also

Other markdown helper function: `addFiguresAndTables()`, `mdBullet0()`, `mdCaption()`, `mdFigure()`, `mdFootNote()`, `mdHeading()`, `mdLink()`, `mdNewline()`, `mdNewpage()`, `mdPaste()`, `mdPaste0()`, `mdTable()`

`mdBullet0` *Insert a bulleted item, making sure that there are sufficient newlines to build a bulleted list*

Description

Insert a bulleted item, making sure that there are sufficient newlines to build a bulleted list

Usage

```
mdBullet0(..., bullet = "-", level = 1)
```

Arguments

`...`	passed to ‘mdPaste0‘
`bullet`	‘character‘ type of bullet
`level`	‘integer‘, specifies the indentation level. ‘level=1‘ has no indentation

See Also

Other markdown helper function: `addFiguresAndTables()`, `mdBullet()`, `mdCaption()`, `mdFigure()`, `mdFootNote()`, `mdHeading()`, `mdLink()`, `mdNewline()`, `mdNewpage()`, `mdPaste()`, `mdPaste0()`, `mdTable()`

mdCaption *add Caption to .Rmd*

Description

add Caption to .Rmd

Usage

```
mdCaption(subfolder, captionFile, captionPrefix, captionStyle = NULL)
```

Arguments

subfolder	The folder where the file is located relative to Rmd
captionFile	file containing caption
captionPrefix	'a character which starts the caption like 'Figure 1:'
captionStyle	'custom-style for captions'

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdFigure(), mdFootNote(), mdHeading(), mdLink(), mdNewline(), mdNewpage(), mdPaste(), mdPaste0(), mdTable()

mdFigure *mdFigure*

Description

Include a figure file with caption

Usage

```
mdFigure(
  figureNumber,
  figureFile,
  captionFile,
  footNoteFile = NULL,
  subfolder,
  addNewPage = TRUE,
  customStyles = list()
)
```

Arguments

figureNumber	number of figure used in caption prefix
figureFile	file of Figure
captionFile	file containing caption
footNoteFile	file containing footnotes
subfolder	The folder where the file is located relative to Rmd
addNewPage	boolean if TRUE (default) new page is added after
customStyles	list of custom styles usable for figure and table captions and footnotes

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFootNote(), mdHeading(), mdLink(), mdNewline(), mdNewpage(), mdPaste(), mdPaste0(), mdTable()

---

mdFootNote *add footnote lines*

Description

add footnote lines

Usage

```
mdFootNote(subfolder, footNoteFile, footNoteCustomStyle = NULL)
```

Arguments

subfolder	The folder where the file is located relative to Rmd
footNoteFile	file containing footnotes
footNoteCustomStyle	a character describing custom style footnotes

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdHeading(), mdLink(), mdNewline(), mdNewpage(), mdPaste(), mdPaste0(), mdTable()

mdHeading *mdHeading*

Description

Insert a heading with a specified level.

Usage

```
mdHeading(..., level = 1, newlines = 2)
```

Arguments

...	passed to mdPaste
level	The header level, i.e. the number of '#'s. Defaults to 1.
newlines	The number of newlines inserted after the heading. Defaults to 2.

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdLink(), mdNewline(), mdNewpage(), mdPaste(), mdPaste0(), mdTable()

mdLink *mdLink*

Description

Insert a link to another file, and a newline.

Usage

```
mdLink(label, filename, folder, prefix = "")
```

Arguments

label	The text to display as the link.
filename	The file to link to.
folder	The folder where the file is located.
prefix	An optional prefix to place in front of the link. Defaults to "".

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdHeading(), mdNewline(), mdNewpage(), mdPaste(), mdPaste0(), mdTable()

mdNewline *mdNewline*

Description

'cat' a line end and start a new line.

Usage

```
mdNewline(n = 1)
```

Arguments

n	Number of new lines. Defaults to 1.

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdHeading(), mdLink(), mdNewpage(), mdPaste(), mdPaste0(), mdTable()

mdNewpage *mdNewpage*

Description

Insert a page break and a newline.

Usage

```
mdNewpage()
```

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdHeading(), mdLink(), mdNewline(), mdPaste(), mdPaste0(), mdTable()

mdPaste *mdPaste*

Description

'cat' text using 'paste' and optionally end with newlines.

Usage

```
mdPaste(..., sep = " ", collapse = NULL, newlines = 1)
```

Arguments

...	Elements to be 'paste'd.
sep	Passed to 'paste'. Defaults to '" "'.
collapse	Passed to 'paste'. Defaults to 'NULL'.
newlines	Number of newlines to insert afterwards. Defaults to 1.

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdHeading(), mdLink(), mdNewline(), mdNewpage(), mdPaste0(), mdTable()

mdPaste0 *mdPaste0*

Description

'cat' text using 'paste0' and optionally end with newlines.

Usage

```
mdPaste0(..., collapse = NULL, newlines = 1)
```

Arguments

...	Elements to be 'paste0'd.
collapse	Passed to 'paste0'. Defaults to 'NULL'.
newlines	Number of newlines to insert afterwards. Defaults to 1.

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdHeading(), mdLink(), mdNewline(), mdNewpage(), mdPaste(), mdTable()

mdTable *mdTable*

Description

Insert a table into a markdown document. Essentially a wrapper for ‘knitr::kable‘ with ‘format = "markdown"‘ and newlines.

Usage

```
mdTable(
  tableNumber,
  tableCsv,
  captionFile,
  footNoteFile,
  subfolder,
  customStyles,
  digitsOfSignificance = 3,
  addNewPage = TRUE,
  ...
)
```

Arguments

tableNumber	number of table for caption prefix
tableCsv	path of .csv file
captionFile	file containing caption
footNoteFile	file containing footnotes
subfolder	The folder where the file is located relative to Rmd
customStyles	list of custom styles usable for figure and table captions and footnotes
digitsOfSignificance	significance digits to display (default 3)
addNewPage	boolean if TRUE (default) new page is added after
...	passed to ‘knitr::kable‘

Value

‘x‘, invisibly

See Also

Other markdown helper function: addFiguresAndTables(), mdBullet(), mdBullet0(), mdCaption(), mdFigure(), mdFootNote(), mdHeading(), mdLink(), mdNewline(), mdNewpage(), mdPaste(), mdPaste0()

`mergeRmds` *Creates Rmd file which include specified .Rmd s*

Description

Creates Rmd file which include specified .Rmd s

Usage

```
mergeRmds(
  newName = "appendix",
  title = "Appendix",
 sourceRmds = c("Demographics", "TimeProfile", "PKParameter", "DDIRatio", "myFigures"),
  projectConfiguration
)
```

Arguments

`newName`	new Name of .Rmd
`title`	Title of new Rmd
`sourceRmds`	list of .rmds to include
`projectConfiguration`	Object of class 'ProjectConfiguration' containing information on paths and file names

`openEPackageTemplate` *Opens the template for figure creation as new document*

Description

Opens the template for figure creation as new document

Usage

```
openEPackageTemplate()
```

See Also

Other Addins Support: `createDocumentFromTemplate()`, `createWorkflowTemplate()`, `openFigureTemplate()`, `openWorkflowTemplate()`

openFigureTemplate *Opens the template for figure creation as new document*

Description

Opens the template for figure creation as new document

Usage

```
openFigureTemplate()
```

See Also

Other Addins Support: `createDocumentFromTemplate()`, `createWorkflowTemplate()`, `openEPackageTemplate()`, `openWorkflowTemplate()`

openWorkflowTemplate *Opens the workflow template as new document*

Description

Per default the workflow template of the package is loaded. To customize it save the file with name "template_workflow.R" in your template directory and set the path to the template via option ‘options(OSPSuite.RF.PathForWorkflowTemplate = ’myTemplateDirectory’)‘

Usage

```
openWorkflowTemplate()
```

See Also

Other Addins Support: `createDocumentFromTemplate()`, `createWorkflowTemplate()`, `openEPackageTemplate()`, `openFigureTemplate()`

`ospsuite_plotPredictedVsObserved` *Plots predicted vs observed data, grouped by "group".*

Description

This function visualizes the relationship between predicted and observed values, allowing for easy identification of discrepancies.

Usage

```
ospsuite_plotPredictedVsObserved(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  ...
)
```

Arguments

`plotData`	An object of class DataCombined or a data.frame generated by a DataCombined object. Units should be consistent; if not, use ‘convertUnits(myDataCombined)‘ to standardize.
`metaData`	A list containing metadata for the plot. If NULL, a default list is constructed from the data.
`mapping`	additional mappings specified by user
`...`	Additional arguments passed to ‘ospsuite.plots::plotPredVsObs‘.

Value

A ‘ggplot2‘ plot object representing predicted vs observed values.

Examples

```
## Not run:
# Generate a predicted vs observed plot for the provided data
plotPredictedVsObserved(convertUnits(
  myDataCombined,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`
))

## End(Not run)
```

`ospsuite_plotQuantileQuantilePlot` *Plots a Quantile-Quantile plot, grouped by "group".*

Description

This function visualizes the distribution of predicted vs observed values using a Q-Q plot.

Usage

```
ospsuite_plotQuantileQuantilePlot(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  ...
)
```

Arguments

`plotData`	An object of class DataCombined or a data.frame generated by a DataCombined object. Units should be consistent; if not, use ‘convertUnits(myDataCombined)‘ to standardize.
`metaData`	A list containing metadata for the plot. If NULL, a default list is constructed from the data.
`mapping`	additional mappings specified by user
`...`	Additional arguments passed to ‘ospsuite.plots::plotQQ‘.

Value

A ‘ggplot2‘ plot object representing the Q-Q plot.

Examples

```
## Not run:
# Generate a Q-Q plot for the provided data
plotQuantileQuantilePlot(convertUnits(
  myDataCombined,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`
))

## End(Not run)
```

`ospsuite_plotResidualsAsHistogram` *Plots residuals as a histogram, grouped by "group".*

Description

This function generates a histogram of the residuals, providing a visual representation of their distribution.

Usage

```
ospsuite_plotResidualsAsHistogram(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  ...
)
```

Arguments

`plotData`	An object of class DataCombined or a data.frame generated by a DataCombined object. Units should be consistent; if not, use ‘convertUnits(myDataCombined)‘ to standardize.
`metaData`	A list containing metadata for the plot. If NULL, a default list is constructed from the data.
`mapping`	additional mappings specified by user
`...`	Additional arguments passed to ‘ospsuite.plots::plotHistogram‘.

Value

A ‘ggplot2‘ plot object representing the histogram of residuals.

Examples

```
## Not run:
# Generate a histogram of residuals for the provided data
plotResidualsAsHistogram(convertUnits(
  myDataCombined,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`
))

## End(Not run)
```

`ospsuite_plotResidualsVsObserved` *Plots residuals vs observed values, grouped by "group".*

Description

This function visualizes the residuals against observed values, helping to assess model performance.

Usage

```
ospsuite_plotResidualsVsObserved(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  ...
)
```

Arguments

`plotData`	An object of class DataCombined or a data.frame generated by a DataCombined object. Units should be consistent; if not, use ‘convertUnits(myDataCombined)‘ to standardize.
`metaData`	A list containing metadata for the plot. If NULL, a default list is constructed from the data.
`mapping`	additional mappings specified by user
`...`	Additional arguments passed to ‘ospsuite.plots::plotResVsCov‘.

Value

A ‘ggplot2‘ plot object representing residuals vs observed values.

Examples

```
## Not run:
# Generate a residuals vs observed plot for the provided data
plotResidualsVsObserved(convertUnits(
  myDataCombined,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`
))

## End(Not run)
```

`ospsuite_plotResidualsVsTime` *Plots residuals vs Time, grouped by "group".*

Description

This function visualizes the residuals over time, providing insights into the accuracy of the model predictions.

Usage

```
ospsuite_plotResidualsVsTime(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  ...
)
```

Arguments

`plotData`	An object of class DataCombined or a data.frame generated by a DataCombined object. Units should be consistent; if not, use ‘convertUnits(myDataCombined)‘ to standardize.
`metaData`	A list containing metadata for the plot. If NULL, a default list is constructed from the data.
`mapping`	additional mappings specified by user
`...`	Additional arguments passed to ‘ospsuite.plots::plotResVsCov‘.

Value

A ‘ggplot2‘ plot object representing residuals vs time.

Examples

```
## Not run:
# Generate a residuals vs time plot for the provided data
plotResidualsVsTime(convertUnits(
  myDataCombined,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`
))

## End(Not run)
```

ospsuite_plotTimeProfile *Creates a time profile plot for given data.*

Description

This function generates a time profile plot using ggplot2, where the data is grouped by a column named "group".

Usage

```
ospsuite_plotTimeProfile(
  plotData,
  metaData = NULL,
  mapping = ggplot2::aes(),
  observedMapping = mapping,
  ...
)
```

Arguments

plotData	An object of class DataCombined or a data.frame generated by a DataCombined object. It is essential that the units are consistent; if not, use 'convertUnits(myDataCombined)' to standardize.
metaData	A list containing metadata for the plot. If NULL, a default list is constructed from the data.
mapping	additional mappings specified by user
observedMapping	Aesthetic mappings for the observed data, typically created with 'ggplot2::aes()'.
...	Additional arguments passed to 'ospsuite.plots::plotTimeProfile'.

Value

A 'ggplot2' plot object representing the time profile.

Examples

```
## Not run:
# Generate a time profile plot for the provided data
plotTimeProfile(convertUnits(
  myDataCombined,
  xUnit = ospUnits$Time$h,
  yUnit = ospUnits$`Concentration [mass]`$`µg/l`
))

## End(Not run)
```

`plotDistributionVsDemographics` *Plot Distribution vs Demographics*

Description

Generates plots comparing distributions against demographic data based on the provided configuration and data.

Usage

```
plotDistributionVsDemographics(
  projectConfiguration,
  onePlotConfig,
  pkParameterDT = NULL,
  scenarioList = NULL,
  asStepPlot = TRUE,
 aggregationFlag = c("Percentiles", "GeometricStdDev", "ArithmeticStdDev", "Custom"),
  customFunction = NULL,
  percentiles = ospsuite.plots::getOspsuite.plots.option(optionKey =
    OptionKeys$Percentiles)[c(1, 3, 5)],
  facetAspectRatio = 0.5,
  colorVector = c(scenario = NA, referenceScenario = NA),
  ...
)
```

Arguments

`projectConfiguration`	A configuration object for the project, containing necessary settings.
`onePlotConfig`	A configuration table for the specific plot, detailing plot parameters.
`pkParameterDT`	A data.table containing pharmacokinetic (PK) parameter data (optional).
`scenarioList`	A list of scenarios to consider for the plot (optional).
`asStepPlot`	Logical indicating if the plot should be rendered as a step plot (default is TRUE).
`aggregationFlag`	A character vector specifying the aggregation function for simulated data. Options include "GeometricStdDev" (default), "ArithmeticStdDev", "Percentiles", and "Custom".
`customFunction`	Optional. A custom aggregation function that accepts a numeric vector and returns a list with: - ‘yValues‘: The aggregated value (e.g., mean). - ‘yMin‘: The lower bound of the aggregated data (e.g., mean - sd). - ‘yMax‘: The upper bound of the aggregated data (e.g., mean + sd). - ‘yErrorType‘: A string indicating the error type for the aggregation, used for plot legends.
`percentiles`	A numeric vector of percentiles to consider if the aggregation method is set to "Percentiles" (default is c(5, 50, 95)).

`facetAspectRatio`
: A numeric value for the aspect ratio of the facets (default is 0.5).

`colorVector`
: A named vector for colors corresponding to scenarios (default is c(scenario = NA, referenceScenario = NA)).

`...`
: Additional arguments passed to the plotting functions 'ospsuite.plots::plotHistogram' or 'plotRangeDistribution'.

Details

This function is intended to be used in combination with the 'runPlot' function. Refer to the vignette 'Plot and Report Generation' for more details. It can be configured to plot PK parameters or population parameters, with the X-axis always displaying the population parameter.

A helper function, 'addDefaultConfigForDistributionsVsDemographics', is available to streamline the setup of your Excel configuration sheet for histograms. This function automatically populates a new sheet with default values that can be customized as needed.

Value

A list of plot objects generated based on the input configuration.

See Also

Other plot functions: `addDefaultConfigForPKForestPlots()`, `addDefaultConfigForTimeProfilePlots()`, `plotHistograms()`, `plotPKBoxwhisker()`, `plotPKForestAggregatedAbsoluteValues()`, `plotPKForestAggregatedRat`
`plotPKForestPointEstimateOfAbsoluteValues()`, `plotPKForestPointEstimateOfRatios()`, `plotSensitivity()`, `plotTimeProfiles()`

Other functions to generate plots displaying distribution vs demographics: `addDefaultConfigForHistograms()`, `validateDistributionVsDemographicsConfig()`

Examples

```
## Not run:
# Example for plotting a histogram of PK-parameters of a scenario compared to a reference scenario:
runPlot(
  nameOfplotFunction = "plotDistributionVsDemographics",
  projectConfiguration = projectConfiguration,
  configTableSheet = "DistributionVsRange",
  inputs = list(
    scenarioList = scenarioList,
    pkParameterDT = pkParameterDT,
    aggregationFlag = "GeometricStdDev"
  )
)

## End(Not run)
```

`plotHistograms` *Plot Histograms*

Description

Generates histogram plots based on the provided configuration and data.

Usage

```
plotHistograms(
  projectConfiguration,
  onePlotConfig,
  pkParameterDT = NULL,
  scenarioList = NULL,
  facetAspectRatio = 0.5,
  colorVector = c(scenario = NA, referenceScenario = NA),
  nMaxFacetRows = 2,
  ...
)
```

Arguments

`projectConfiguration`	A configuration object for the project, containing necessary settings.
`onePlotConfig`	A configuration table for the specific plot, detailing plot parameters.
`pkParameterDT`	A data.table containing pharmacokinetic (PK) parameter data (optional).
`scenarioList`	A list of scenarios to consider for the plot (optional).
`facetAspectRatio`	A numeric value for the aspect ratio of the facets (default is 0.5).
`colorVector`	A named vector for colors corresponding to scenarios (default is c(scenario = NA, referenceScenario = NA)).
`nMaxFacetRows`	Maximum number of facet rows (default is 2).
`...`	Additional arguments passed to the histogram plotting function.

Details

This function is intended to be used in combination with the 'runPlot' function. Refer to the vignette 'Plot and Report Generation' for more details. It can be configured to plot PK parameters or model parameters.

A helper function, 'addDefaultConfigForHistograms', is available to streamline the setup of your Excel configuration sheet for histograms. This function automatically populates a new sheet with default values that can be customized as needed.

Value

A list of histogram plot objects generated based on the input configuration.

See Also

Other plot functions: addDefaultConfigForPKForestPlots(), addDefaultConfigForTimeProfilePlots(), plotDistributionVsDemographics(), plotPKBoxwhisker(), plotPKForestAggregatedAbsoluteValues(), plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimate plotSensitivity(), plotTimeProfiles()

Other functions to generate histograms: addDefaultConfigForDistributionsVsDemographics(), validateHistogramsConfig()

Examples

```
## Not run:
# Example for plotting a histogram of PK-parameters of a scenario compared to a reference scenario:
runPlot(
  nameOfplotFunction = "plotHistograms",
  projectConfiguration = projectConfiguration,
  configTableSheet = "HistogramTest",
  inputs = list(
    scenarioList = scenarioList,
    pkParameterDT = pkParameterDT,
    colorVector = c(
      "PO application" = "red",
      "IV application" = "green"
    )
  )
)

## End(Not run)
```

---

plotPKBoxwhisker *Plot PK Box-and-Whisker*

Description

Generates box-and-whisker plots for pharmacokinetic (PK) parameters based on a provided project configuration and plot configuration. This function creates visual representations of the distribution of PK parameters across different scenarios, allowing for comparison of absolute values and ratios.

Usage

```
plotPKBoxwhisker(
  projectConfiguration,
  onePlotConfig,
  pkParameterDT,
  percentiles = getOspsuite.plots.option(optionKey = OptionKeys$Percentiles),
  xAxisTextAngle = 0,
  colorVector = c(scenario = NA, referenceScenario = NA),
  facetAspectRatio = 0.5,
```

```
  ...
)
```

Arguments

projectConfiguration	A ProjectConfiguration object that contains settings and paths relevant to the project.
onePlotConfig	A data.table containing configuration settings for a single plot, including details about which plots to generate and their formatting.
pkParameterDT	A data.table containing PK parameter data.
percentiles	A numeric vector specifying the percentiles to calculate for the box-and-whisker plots. Default is retrieved from project options.
xAxisTextAngle	An integer specifying the angle (in degrees) for rotating the x-axis text labels. Default is 0 (no rotation).
colorVector	A named vector specifying colors for the plots. Names should correspond to the characters defined in the configtable column colorLegend. If your color legend is for example 'DDI|control', your color vector could be colorVector = c(DDI = 'red',control = 'blue'), if no color is defined default values are used.
facetAspectRatio	A numeric value specifying the aspect ratio for faceting the plots. Default is 0.5, which may need adjustment based on the number of facets and plot dimensions.
...	Additional arguments passed to plotting functions for further customization.

Details

The function is intended to be used in combination with the 'runPlot' function. See the vignette 'Plot and Report Generation' for more details.

There exists a helper function 'addDefaultConfigForPKBoxwhsikerPlots' which is a helpful utility designed to streamline the process of setting up your Excel configuration sheet for box whisker plots. This function automatically populates a new sheet with default values that you can customize according to your needs.

Value

A list of ggplot objects representing the generated plots. Each plot can be rendered using ggplot2 or similar plotting systems. The list may include both absolute and ratio plots depending on the configuration.

See Also

Other plot functions: addDefaultConfigForPKForestPlots(), addDefaultConfigForTimeProfilePlots(), plotDistributionVsDemographics(), plotHistograms(), plotPKForestAggregatedAbsoluteValues(), plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimate
plotSensitivity(), plotTimeProfiles()

Other functions to generate box whisker plots: addDefaultConfigForPKBoxwhsikerPlots(), validatePKBoxwhiskerConfig()

Examples

```
# Example usage of plotPKBoxwhisker
## Not run:
plotList <- plotPKBoxwhisker(
  projectConfiguration = myProjectConfig,
  onePlotConfig = myPlotConfig,
  pkParameterDT = myPKData,
  percentiles = c(0.05, 0.25, 0.5, 0.75, 0.95),
  xAxisTextAngle = 45,
  colorVector = c(DDI = "red", control = "blue"),
  facetAspectRatio = 1,
)

## End(Not run)
```

`plotPKForestAggregatedAbsoluteValues` *PK Forest Plots*

Description

This is one of a group of functions which generates PK forest plots for pharmacokinetic (PK) parameters or population parameter, providing visualizations that can include absolute values or ratios, along with options for displaying variance or point estimates. These functions allow for the comparison of simulated and observed PK parameter data across different scenarios, aiding in the interpretation of variability and uncertainty in the estimates.

The aggregation of data can be customized using the ‘aggregationFlag‘ parameter, which allows for options such as "GeometricStdDev", "ArithmeticStdDev", "Percentiles", and "Custom" functions for calculating summary statistics.

Available functions are:

`plotPKForestAggregatedAbsoluteValues` Generates a PK forest plot displaying absolute values along with variance.

`plotPKForestPointEstimateOfAbsoluteValues` Generates a PK forest plot displaying point estimates.

`plotPKForestAggregatedRatios` Generates a PK forest plot displaying ratios of PK parameters along with variance.

`plotPKForestPointEstimateOfRatios` Generates a PK forest plot displaying point estimates of ratios of PK parameters.

Usage

```
plotPKForestAggregatedAbsoluteValues(
  projectConfiguration,
  onePlotConfig,
  pkParameterDT,
```

```
  dataObservedPK = NULL,
 aggregationFlag = c("GeometricStdDev", "ArithmeticStdDev", "Percentiles", "Custom"),
 percentiles = getOspsuite.plots.option(optionKey = OptionKeys$Percentiles)[c(1, 3, 5)],
  customFunction = NULL,
  scaleVectors = list(),
  labelWrapWidth = 10,
  vlineIntercept = NULL,
  withTable = TRUE,
  relWidths = NULL,
  digitsToRound = 2,
  digitsToShow = 2
)
```

Arguments

`projectConfiguration`
: A ProjectConfiguration object containing the necessary settings and file paths for the project.

`onePlotConfig`
: A data.table containing configuration settings for a single plot, including plot name, x-axis scale, and other aesthetic settings.

`pkParameterDT`
: A data.table containing simulated PK parameter data, including columns for scenario names, parameters, values, and output paths.

`dataObservedPK`
: Optional data.table containing observed PK parameter data for comparison, which can include columns for observed values and associated metadata.

`aggregationFlag`
: A character string indicating the method of aggregation for variance calculation. Options include "GeometricStdDev", "ArithmeticStdDev", and "Percentiles". (for ‘plotPKForestAggregatedAbsoluteValues‘ and ‘plotPKForestAggregatedRatios‘):

`percentiles`
: A numeric vector specifying which percentiles to calculate and display in the plot, only relevant if ‘aggregationFlag = "Percentiles"‘ (for ‘plotPKForestAggregatedAbsoluteValues‘ and ‘plotPKForestAggregatedRatios‘):

`customFunction`
: An optional custom function for aggregation. A custom function should take a numeric vector ‘y‘ as input and return a list containing:

  - ‘yValues‘: The aggregated value (e.g., mean). - ‘yMin‘: The lower value of the aggregated data (e.g., mean - sd). - ‘yMax‘: The upper value of the aggregated data (e.g., mean + sd). - ‘yErrorType‘: A string indicating the type of error associated with the aggregation, it is used in plot legends and captions. It must be a concatenation of the descriptor of ‘yValues‘ and the descriptor of ‘yMin - yMax‘ range separated by "|" (e.g., "mean | standard deviation" or "median | 5th - 95th percentile").

`scaleVectors`
: A list containing user-defined scale vectors for the plot aesthetics for simulated and observed data. The list can have up to two components: ‘simulated‘ and ‘observed‘, each containing a list of properties (‘color‘, ‘fill‘, ‘shape‘) to modify the default scale vector settings.

`labelWrapWidth`
: Numeric value specifying the maximum width for wrapping labels in the plot to enhance readability.

vlineIntercept	Optional numeric value indicating where to draw a vertical line on the plot, often used to denote a reference value.
withTable	logical, if TRUE (default) values are displayed as table beside the plot
relWidths	Optional numeric vector specifying relative widths for the plot and table.
digitsToRound	An integer specifying the number of digits to round in the displayed values.
digitsToShow	An integer specifying the number of digits to show in the displayed values.

Details

The function distinguishes between two cases for ratio calculations:

Case 1: If the scenarios being compared are based on the **same populations** (same 'PopulationId' in scenario configuration), the plots will show summary statistics of individual ratios.

Case 2: If the scenarios are based on **different populations**, the plots will display the ratio of summary statistics. This ratio is only available for `plotPKForestPointEstimateOfRatios`

The uncertainty in estimates is calculated using a bootstrapping approach. A unique seed is set for each combination of scenario, reference scenario (if available), output path ID, and PK parameter identifier to ensure reproducibility. The number of bootstrap samples is defined by the 'nBootstrap' parameter, and the confidence intervals are determined using the 'confLevel' parameter (e.g., 0.9 for 90

For the bootstrapping process, a unique seed is set for each combination of scenario, 'referenceScenario' (if available), 'outputPathId', and 'PKParameter' identifier. This ensures that results are consistent when a specific combination is used in different plots and that they are reproducible.

In addition to the point estimates and their confidence intervals, the precision watermark is an important feature of these plots. The precision watermark indicates whether the precision requirements for the displayed estimates have been met. The precision is calculated based on the relationship between the estimated values ('xValues') and their corresponding minimum ('xMin') and maximum ('xMax') confidence interval bounds. If the calculated precision falls below a predefined threshold (e.g., 0.01), the watermark is activated, displaying a warning label such as "Outside precision requirement" on the plot. The threshold is set by the option 'OSPSuite.RF.RequiredPrecisison' This alert serves to inform users that the estimates may not be reliable due to insufficient sample sizes or high variability in the data, prompting them to consider increasing the sample size for improved precision. If the precision is sufficient, point estimates are displayed without confidence interval bounds.

By including the precision watermark, users are better equipped to assess the reliability of the point estimates and make informed decisions based on the visualized data.

To support the creation of the configuration table a support function `addDefaultConfigForPKForestPlots` is available

Value

A list containing the generated plot objects.

See Also

Other plot functions: addDefaultConfigForPKForestPlots(), addDefaultConfigForTimeProfilePlots(), plotDistributionVsDemographics(), plotHistograms(), plotPKBoxwhisker(), plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimateOfRatios(), plotSensitivity(), plotTimeProfiles()

Other functions to generate forest plots: addDefaultConfigForPKForestPlots(), plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimateOfRatios(), validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig()

`plotPKForestAggregatedRatios` *PK Forest Plots*

Description

This is one of a group of functions which generates PK forest plots for pharmacokinetic (PK) parameters or population parameter, providing visualizations that can include absolute values or ratios, along with options for displaying variance or point estimates. These functions allow for the comparison of simulated and observed PK parameter data across different scenarios, aiding in the interpretation of variability and uncertainty in the estimates.

The aggregation of data can be customized using the 'aggregationFlag' parameter, which allows for options such as "GeometricStdDev", "ArithmeticStdDev", "Percentiles", and "Custom" functions for calculating summary statistics.

Available functions are:

`plotPKForestAggregatedAbsoluteValues` Generates a PK forest plot displaying absolute values along with variance.

`plotPKForestPointEstimateOfAbsoluteValues` Generates a PK forest plot displaying point estimates.

`plotPKForestAggregatedRatios` Generates a PK forest plot displaying ratios of PK parameters along with variance.

`plotPKForestPointEstimateOfRatios` Generates a PK forest plot displaying point estimates of ratios of PK parameters.

Usage

```
plotPKForestAggregatedRatios(
  projectConfiguration,
  onePlotConfig,
  pkParameterDT,
  dataObservedPK = NULL,
 aggregationFlag = c("GeometricStdDev", "ArithmeticStdDev", "Percentiles", "Custom"),
 percentiles = getOspsuite.plots.option(optionKey = OptionKeys$Percentiles)[c(1, 3, 5)],
  customFunction = NULL,
```

```
  scaleVectors = list(),
  labelWrapWidth = 10,
  vlineIntercept = NULL,
  withTable = TRUE,
  relWidths = NULL,
  digitsToRound = 2,
  digitsToShow = 2
)
```

Arguments

projectConfiguration	A ProjectConfiguration object containing the necessary settings and file paths for the project.
onePlotConfig	A data.table containing configuration settings for a single plot, including plot name, x-axis scale, and other aesthetic settings.
pkParameterDT	A data.table containing simulated PK parameter data, including columns for scenario names, parameters, values, and output paths.
dataObservedPK	Optional data.table containing observed PK parameter data for comparison, which can include columns for observed values and associated metadata.
aggregationFlag	A character string indicating the method of aggregation for variance calculation. Options include "GeometricStdDev", "ArithmeticStdDev", and "Percentiles". (for ‘plotPKForestAggregatedAbsoluteValues‘ and ‘plotPKForestAggregatedRatios‘):
percentiles	A numeric vector specifying which percentiles to calculate and display in the plot, only relevant if ‘aggregationFlag = "Percentiles"‘ (for ‘plotPKForestAggregatedAbsoluteValues‘ and ‘plotPKForestAggregatedRatios‘):
customFunction	An optional custom function for aggregation. A custom function should take a numeric vector ‘y‘ as input and return a list containing: - ‘yValues‘: The aggregated value (e.g., mean). - ‘yMin‘: The lower value of the aggregated data (e.g., mean - sd). - ‘yMax‘: The upper value of the aggregated data (e.g., mean + sd). - ‘yErrorType‘: A string indicating the type of error associated with the aggregation, it is used in plot legends and captions. It must be a concatenation of the descriptor of ‘yValues‘ and the descriptor of ‘yMin - yMax‘ range separated by "|" (e.g., "mean | standard deviation" or "median | 5th - 95th percentile").
scaleVectors	A list containing user-defined scale vectors for the plot aesthetics for simulated and observed data. The list can have up to two components: ‘simulated‘ and ‘observed‘, each containing a list of properties (‘color‘, ‘fill‘, ‘shape‘) to modify the default scale vector settings.
labelWrapWidth	Numeric value specifying the maximum width for wrapping labels in the plot to enhance readability.
vlineIntercept	Optional numeric value indicating where to draw a vertical line on the plot, often used to denote a reference value.
withTable	logical, if TRUE (default) values are displayed as table beside the plot

relWidths	Optional numeric vector specifying relative widths for the plot and table.
digitsToRound	An integer specifying the number of digits to round in the displayed values.
digitsToShow	An integer specifying the number of digits to show in the displayed values.

Details

The function distinguishes between two cases for ratio calculations:

Case 1: If the scenarios being compared are based on the **same populations** (same 'PopulationId' in scenario configuration), the plots will show summary statistics of individual ratios.

Case 2: If the scenarios are based on **different populations**, the plots will display the ratio of summary statistics. This ratio is only available for plotPKForestPointEstimateOfRatios

The uncertainty in estimates is calculated using a bootstrapping approach. A unique seed is set for each combination of scenario, reference scenario (if available), output path ID, and PK parameter identifier to ensure reproducibility. The number of bootstrap samples is defined by the 'nBootstrap' parameter, and the confidence intervals are determined using the 'confLevel' parameter (e.g., 0.9 for 90

For the bootstrapping process, a unique seed is set for each combination of scenario, 'referenceScenario' (if available), 'outputPathId', and 'PKParameter' identifier. This ensures that results are consistent when a specific combination is used in different plots and that they are reproducible.

In addition to the point estimates and their confidence intervals, the precision watermark is an important feature of these plots. The precision watermark indicates whether the precision requirements for the displayed estimates have been met. The precision is calculated based on the relationship between the estimated values ('xValues') and their corresponding minimum ('xMin') and maximum ('xMax') confidence interval bounds. If the calculated precision falls below a predefined threshold (e.g., 0.01), the watermark is activated, displaying a warning label such as "Outside precision requirement" on the plot. The threshold is set by the option 'OSPSuite.RF.RequiredPrecisison' This alert serves to inform users that the estimates may not be reliable due to insufficient sample sizes or high variability in the data, prompting them to consider increasing the sample size for improved precision. If the precision is sufficient, point estimates are displayed without confidence interval bounds.

By including the precision watermark, users are better equipped to assess the reliability of the point estimates and make informed decisions based on the visualized data.

To support the creation of the configuration table a support function addDefaultConfigForPKForestPlots is available

Value

A list containing the generated plot objects.

See Also

Other plot functions: addDefaultConfigForPKForestPlots(), addDefaultConfigForTimeProfilePlots(), plotDistributionVsDemographics(), plotHistograms(), plotPKBoxwhisker(), plotPKForestAggregatedAbsoluteV plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimateOfRatios(), plotSensitivity(), plotTimeProfiles()

Other functions to generate forest plots: addDefaultConfigForPKForestPlots(), plotPKForestAggregatedAbsoluteVal
plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimateOfRatios(),
validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestAggregatedRatiosConfig(),
validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig()

`plotPKForestPointEstimateOfAbsoluteValues`
PK Forest Plots

Description

This is one of a group of functions which generates PK forest plots for pharmacokinetic (PK) parameters or population parameter, providing visualizations that can include absolute values or ratios, along with options for displaying variance or point estimates. These functions allow for the comparison of simulated and observed PK parameter data across different scenarios, aiding in the interpretation of variability and uncertainty in the estimates.

The aggregation of data can be customized using the ‘aggregationFlag‘ parameter, which allows for options such as "GeometricStdDev", "ArithmeticStdDev", "Percentiles", and "Custom" functions for calculating summary statistics.

Available functions are:

`plotPKForestAggregatedAbsoluteValues` Generates a PK forest plot displaying absolute values along with variance.

`plotPKForestPointEstimateOfAbsoluteValues` Generates a PK forest plot displaying point estimates.

`plotPKForestAggregatedRatios` Generates a PK forest plot displaying ratios of PK parameters along with variance.

`plotPKForestPointEstimateOfRatios` Generates a PK forest plot displaying point estimates of ratios of PK parameters.

Usage

```
plotPKForestPointEstimateOfAbsoluteValues(
  projectConfiguration,
  onePlotConfig,
  pkParameterDT,
  dataObservedPK = NULL,
  statFun = c(`geometric mean` = function(y) exp(mean(log(y[y > 0])))),
  confLevel = 0.9,
  nBootstrap = 100,
  scaleVectors = list(),
  labelWrapWidth = 10,
  vlineIntercept = NULL,
  withTable = TRUE,
  relWidths = NULL,
  digitsToRound = 2,
  digitsToShow = 2
)
```

Arguments

projectConfiguration	A ProjectConfiguration object containing the necessary settings and file paths for the project.
onePlotConfig	A data.table containing configuration settings for a single plot, including plot name, x-axis scale, and other aesthetic settings.
pkParameterDT	A data.table containing simulated PK parameter data, including columns for scenario names, parameters, values, and output paths.
dataObservedPK	Optional data.table containing observed PK parameter data for comparison, which can include columns for observed values and associated metadata.
statFun	A named list of functions used for statistical aggregation, typically including the geometric mean function for bootstrapping. This must be a function accepting as input a numeric vector and returning a numeric value. The input is formatted as a named list, the name is used in plot legends and captions c('geometric mean' = function(y) exp(mean(log(y[y>0])))) (for 'plotPKForestPointEstimateOfAbsoluteValues' and 'plotPKForestPointEstimateOfRatios')
confLevel	A numeric value between 0 and 1 indicating the desired confidence level for the intervals (e.g., 0.9 for 90 (for 'plotPKForestPointEstimateOfAbsoluteValues' and 'plotPKForestPointEstimateOfRatios')
nBootstrap	An integer indicating the number of bootstrap samples to draw for calculating confidence intervals. (for 'plotPKForestPointEstimateOfAbsoluteValues', 'plotPKForestPointEstimateOfRatios')
scaleVectors	A list containing user-defined scale vectors for the plot aesthetics for simulated and observed data. The list can have up to two components: 'simulated' and 'observed', each containing a list of properties ('color', 'fill', 'shape') to modify the default scale vector settings.
labelWrapWidth	Numeric value specifying the maximum width for wrapping labels in the plot to enhance readability.
vlineIntercept	Optional numeric value indicating where to draw a vertical line on the plot, often used to denote a reference value.
withTable	logical, if TRUE (default) values are displayed as table beside the plot
relWidths	Optional numeric vector specifying relative widths for the plot and table.
digitsToRound	An integer specifying the number of digits to round in the displayed values.
digitsToShow	An integer specifying the number of digits to show in the displayed values.

Details

The function distinguishes between two cases for ratio calculations:

Case 1: If the scenarios being compared are based on the **same populations** (same 'PopulationId' in scenario configuration), the plots will show summary statistics of individual ratios.

Case 2: If the scenarios are based on **different populations**, the plots will display the ratio of summary statistics. This ratio is only available for `plotPKForestPointEstimateOfRatios`

The uncertainty in estimates is calculated using a bootstrapping approach. A unique seed is set for each combination of scenario, reference scenario (if available), output path ID, and PK parameter identifier to ensure reproducibility. The number of bootstrap samples is defined by the ‘nBootstrap‘ parameter, and the confidence intervals are determined using the ‘confLevel‘ parameter (e.g., 0.9 for 90

For the bootstrapping process, a unique seed is set for each combination of scenario, ‘referenceScenario‘ (if available), ‘outputPathId‘, and ‘PKParameter‘ identifier. This ensures that results are consistent when a specific combination is used in different plots and that they are reproducible.

In addition to the point estimates and their confidence intervals, the precision watermark is an important feature of these plots. The precision watermark indicates whether the precision requirements for the displayed estimates have been met. The precision is calculated based on the relationship between the estimated values (‘xValues‘) and their corresponding minimum (‘xMin‘) and maximum (‘xMax‘) confidence interval bounds. If the calculated precision falls below a predefined threshold (e.g., 0.01), the watermark is activated, displaying a warning label such as "Outside precision requirement" on the plot. The threshold is set by the option ‘OSPSuite.RF.RequiredPrecisison‘ This alert serves to inform users that the estimates may not be reliable due to insufficient sample sizes or high variability in the data, prompting them to consider increasing the sample size for improved precision. If the precision is sufficient, point estimates are displayed without confidence interval bounds.

By including the precision watermark, users are better equipped to assess the reliability of the point estimates and make informed decisions based on the visualized data.

To support the creation of the configuration table a support function `addDefaultConfigForPKForestPlots` is available

Value

A list containing the generated plot objects.

See Also

Other plot functions: `addDefaultConfigForPKForestPlots()`, `addDefaultConfigForTimeProfilePlots()`, `plotDistributionVsDemographics()`, `plotHistograms()`, `plotPKBoxwhisker()`, `plotPKForestAggregatedAbsolute`
`plotPKForestAggregatedRatios()`, `plotPKForestPointEstimateOfRatios()`, `plotSensitivity()`, `plotTimeProfiles()`

Other functions to generate forest plots: `addDefaultConfigForPKForestPlots()`, `plotPKForestAggregatedAbsoluteVa`
`plotPKForestAggregatedRatios()`, `plotPKForestPointEstimateOfRatios()`, `validatePKForestAggregatedAbsolu`
`validatePKForestAggregatedRatiosConfig()`, `validatePKForestPointEstimateOfAbsoluteValuesConfig()`, `validatePKForestPointEstimateOfRatiosConfig()`

`plotPKForestPointEstimateOfRatios` *PK Forest Plots*

Description

This is one of a group of functions which generates PK forest plots for pharmacokinetic (PK) parameters or population parameter, providing visualizations that can include absolute values or ratios, along with options for displaying variance or point estimates. These functions allow for the comparison of simulated and observed PK parameter data across different scenarios, aiding in the interpretation of variability and uncertainty in the estimates.

The aggregation of data can be customized using the ‘aggregationFlag‘ parameter, which allows for options such as "GeometricStdDev", "ArithmeticStdDev", "Percentiles", and "Custom" functions for calculating summary statistics.

Available functions are:

`plotPKForestAggregatedAbsoluteValues` Generates a PK forest plot displaying absolute values along with variance.

`plotPKForestPointEstimateOfAbsoluteValues` Generates a PK forest plot displaying point estimates.

`plotPKForestAggregatedRatios` Generates a PK forest plot displaying ratios of PK parameters along with variance.

`plotPKForestPointEstimateOfRatios` Generates a PK forest plot displaying point estimates of ratios of PK parameters.

Usage

```
plotPKForestPointEstimateOfRatios(
  projectConfiguration,
  onePlotConfig,
  pkParameterDT,
  dataObservedPK = NULL,
  nBootstrap = 1000,
  confLevel = 0.9,
  statFun = c(`geometric mean` = function(y) exp(mean(log(y[y > 0])))),
  scaleVectors = list(),
  labelWrapWidth = 10,
  vlineIntercept = c(1),
  withTable = TRUE,
  relWidths = NULL,
  digitsToRound = 2,
  digitsToShow = 2
)
```

Arguments

`projectConfiguration`
: A ProjectConfiguration object containing the necessary settings and file paths for the project.

`onePlotConfig`
: A data.table containing configuration settings for a single plot, including plot name, x-axis scale, and other aesthetic settings.

pkParameterDT	A data.table containing simulated PK parameter data, including columns for scenario names, parameters, values, and output paths.
dataObservedPK	Optional data.table containing observed PK parameter data for comparison, which can include columns for observed values and associated metadata.
nBootstrap	An integer indicating the number of bootstrap samples to draw for calculating confidence intervals. (for 'plotPKForestPointEstimateOfAbsoluteValues', 'plotPKForestPointEstimateOfRatios')
confLevel	A numeric value between 0 and 1 indicating the desired confidence level for the intervals (e.g., 0.9 for 90 (for 'plotPKForestPointEstimateOfAbsoluteValues' and 'plotPKForestPointEstimateOfRatios')
statFun	A named list of functions used for statistical aggregation, typically including the geometric mean function for bootstrapping. This must be a function accepting as input a numeric vector and returning a numeric value. The input is formatted as a named list, the name is used in plot legends and captions c('geometric mean' = function(y) exp(mean(log(y[y>0])))) (for 'plotPKForestPointEstimateOfAbsoluteValues' and 'plotPKForestPointEstimateOfRatios')
scaleVectors	A list containing user-defined scale vectors for the plot aesthetics for simulated and observed data. The list can have up to two components: 'simulated' and 'observed', each containing a list of properties ('color', 'fill', 'shape') to modify the default scale vector settings.
labelWrapWidth	Numeric value specifying the maximum width for wrapping labels in the plot to enhance readability.
vlineIntercept	Optional numeric value indicating where to draw a vertical line on the plot, often used to denote a reference value.
withTable	logical, if TRUE (default) values are displayed as table beside the plot
relWidths	Optional numeric vector specifying relative widths for the plot and table.
digitsToRound	An integer specifying the number of digits to round in the displayed values.
digitsToShow	An integer specifying the number of digits to show in the displayed values.

Details

The function distinguishes between two cases for ratio calculations:

Case 1: If the scenarios being compared are based on the **same populations** (same 'PopulationId' in scenario configuration), the plots will show summary statistics of individual ratios.

Case 2: If the scenarios are based on **different populations**, the plots will display the ratio of summary statistics. This ratio is only available for `plotPKForestPointEstimateOfRatios`

The uncertainty in estimates is calculated using a bootstrapping approach. A unique seed is set for each combination of scenario, reference scenario (if available), output path ID, and PK parameter identifier to ensure reproducibility. The number of bootstrap samples is defined by the 'nBootstrap' parameter, and the confidence intervals are determined using the 'confLevel' parameter (e.g., 0.9 for 90

For the bootstrapping process, a unique seed is set for each combination of scenario, 'referenceScenario' (if available), 'outputPathId', and 'PKParameter' identifier. This ensures that results are consistent when a specific combination is used in different plots and that they are reproducible.

In addition to the point estimates and their confidence intervals, the precision watermark is an important feature of these plots. The precision watermark indicates whether the precision requirements for the displayed estimates have been met. The precision is calculated based on the relationship between the estimated values (‘xValues‘) and their corresponding minimum (‘xMin‘) and maximum (‘xMax‘) confidence interval bounds. If the calculated precision falls below a predefined threshold (e.g., 0.01), the watermark is activated, displaying a warning label such as "Outside precision requirement" on the plot. The threshold is set by the option ‘OSPSuite.RF.RequiredPrecisison‘ This alert serves to inform users that the estimates may not be reliable due to insufficient sample sizes or high variability in the data, prompting them to consider increasing the sample size for improved precision. If the precision is sufficient, point estimates are displayed without confidence interval bounds.

By including the precision watermark, users are better equipped to assess the reliability of the point estimates and make informed decisions based on the visualized data.

To support the creation of the configuration table a support function `addDefaultConfigForPKForestPlots` is available

Value

A list containing the generated plot objects.

See Also

Other plot functions: `addDefaultConfigForPKForestPlots()`, `addDefaultConfigForTimeProfilePlots()`, `plotDistributionVsDemographics()`, `plotHistograms()`, `plotPKBoxwhisker()`, `plotPKForestAggregatedAbsoluteV`, `plotPKForestAggregatedRatios()`, `plotPKForestPointEstimateOfAbsoluteValues()`, `plotSensitivity()`, `plotTimeProfiles()`

Other functions to generate forest plots: `addDefaultConfigForPKForestPlots()`, `plotPKForestAggregatedAbsoluteVal`, `plotPKForestAggregatedRatios()`, `plotPKForestPointEstimateOfAbsoluteValues()`, `validatePKForestAggregate`, `validatePKForestAggregatedRatiosConfig()`, `validatePKForestPointEstimateOfAbsoluteValuesConfig()`, `validatePKForestPointEstimateOfRatiosConfig()`

`plotSensitivity` *Plot Sensitivity Function*

Description

Generates sensitivity plots based on the provided project configuration and scenario list.

Usage

```
plotSensitivity(projectConfiguration, onePlotConfig, scenarioList)
```

Arguments

projectConfiguration	A list containing project configuration settings, including file paths and parameters.
onePlotConfig	A data frame containing the configuration for a single plot.
scenarioList	A list of scenarios for which the sensitivity analysis will be performed.

Value

A list of plots and tables, where each entry corresponds to a generated sensitivity plot or its data.

See Also

Other plot functions: addDefaultConfigForPKForestPlots(), addDefaultConfigForTimeProfilePlots(), plotDistributionVsDemographics(), plotHistograms(), plotPKBoxwhisker(), plotPKForestAggregatedAbsoluteV plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimate plotTimeProfiles()

---

plotTimeProfiles *Generate Time Profile Panels*

Description

The 'plotTimeProfiles' function creates a series of time profile plots as facet panels from the provided simulated and observed data. This function is designed to visualize complex time-dependent data by allowing for multiple plots organized in a user-defined layout.

Usage

```
plotTimeProfiles(
  projectConfiguration,
  onePlotConfig,
  dataObserved = NULL,
  scenarioResults,
  nMaxFacetRows = 4,
  facetAspectRatio = 0.5,
 aggregationFlag = c("GeometricStdDev", "ArithmeticStdDev", "Percentiles", "Custom"),
 percentiles = getOspsuite.plots.option(optionKey = OptionKeys$Percentiles)[c(1, 3, 5)],
  customFunction = NULL,
  checkForUnusedData = FALSE,
  referenceScaleVector = list(),
  ...
)
```

Arguments

`projectConfiguration`
: An object of class 'ProjectConfiguration' containing project settings, including file paths and scenario definitions. This is essential for the function to access the necessary data.

`onePlotConfig`
: A configuration table for the specific plot, detailing plot parameters.

`dataObserved`
: A 'data.table' (formatted as produced by 'readObservedDataByDictionary') or 'DataCombined' object containing the observed data to be plotted.

`scenarioResults`
: A list containing simulated scenario results.

`nMaxFacetRows`
: An integer specifying the maximum number of facet rows (default is 4).

`facetAspectRatio`
: A numeric value specifying the aspect ratio for the facets (default is 0.5).

`aggregationFlag`
: A character vector indicating the type of aggregation function to use for simulated data. Options include "GeometricStdDev" (default), "ArithmeticStdDev", "Percentiles", and "Custom".

`percentiles`
: A numeric vector of percentiles to consider if the aggregation method is set to "Percentiles" (default is c(5, 50, 95)).

`customFunction`
: An optional custom function for aggregation. A custom function should take a numeric vector 'y' as input and return a list containing:

  - 'yValues': The aggregated value (e.g., mean). - 'yMin': The lower value of the aggregated data (e.g., mean - sd). - 'yMax': The upper value of the aggregated data (e.g., mean + sd). - 'yErrorType': A string indicating the type of error associated with the aggregation, it is used in plot legends and captions. It must be a concatenation of the descriptor of 'yValues' and the descriptor of 'yMin - yMax' range separated by "|" (e.g., "mean | standard deviation" or "median | 5th - 95th percentile").

`checkForUnusedData`
: A boolean indicating whether to perform quality control on data usage. If TRUE, 'plotList' contains an additional entry 'unusedData'.

`referenceScaleVector`
: A named list that configures colors for the display of reference scenarios. The names must be consistent with the labels defined in the config table column 'colorLegend'. The first entry corresponds to aesthetic 'color', and the second entry to aesthetic 'fill'.

  Example: referenceScaleVector = list(Simulation = c('blue', 'lightblue'), Reference = c('darkred', 'red')) # Default is list()

`...`
: Additional arguments passed to 'ospsuite.plots::plotTimeprofile'.

Details

The function primarily focuses on simulated scenarios, with each panel displaying the result of one scenario, or one scenario versus a reference scenario. Plots containing more than one scenario or only observed data are not supported.

The function is intended to be used in combination with the ‘runPlot‘ function. See the vignette ‘Plot and Report Generation‘ for more details.

There exists a helper function ‘addDefaultConfigForTimeProfilePlots‘ which is a helpful utility designed to streamline the process of setting up your Excel configuration sheet for time profile plots. This function automatically populates a new sheet with default values that you can customize according to your needs.

There is a tutorial for this function with many examples in the accompanying vignette: ‘Time Profile Plotting Tutorial‘.

Value

A list of generated time profile plots

See Also

Other plot functions: addDefaultConfigForPKForestPlots(), addDefaultConfigForTimeProfilePlots(), plotDistributionVsDemographics(), plotHistograms(), plotPKBoxwhisker(), plotPKForestAggregatedAbsoluteV plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimate plotSensitivity()

Examples

```
## Not run:
# Example of using the referenceScaleVector parameter for a DDI Analysis
referenceScaleVector <- list(
  DDI = c("#843C39FF", "#AD494AFF"),
  Control = c("#3182BDFF", "#6BAED6FF")
)
# The scenario would be labeled as 'DDI', with aesthetic color as dark red ("#843C39FF") and
# aesthetic fill as light red ("#AD494AFF").
# The reference scenario would be labeled 'Control',
# with aesthetic color as dark blue ("#3182BDFF") and
# aesthetic fill as light blue ("#6BAED6FF").

# Example of using the referenceScaleVector parameter for a comparison between a pediatric
# and an adult simulation using the default colors
referenceScaleVector <- list(
  "Pediatric population" = c(NA, NA),
  "Adult population" = c(NA, NA)
)

# The scenario would be labeled as 'Pediatric population', using the default colors with aesthetic
# color dark blue ("#3182BDFF") and aesthetic fill as light blue ("#6BAED6FF").
# The reference scenario would be labeled 'Adult population', with both aesthetic color and fill
# as 'grey'.

# Example of using the aggregationFlag parameter "Custom"
plotList <- runPlot(
  nameOfplotFunction = "plotTimeProfiles",
  configTableSheet = "myTimeProfile",
  projectConfiguration = projectConfiguration,
```

```
  suppressExport = TRUE,
  inputs = list(
    scenarioResults = scenarioResults,
    dataObserved = dataObserved,
    aggregationFlag = c("Custom"),
    customFunction = function(y) {
      list(
        yValues = mean(y),
        yMin = mean(y) - sd(y),
        yMax = mean(y) + sd(y),
        yErrorType = "mean | standard deviation"
      )
    }
  )
)

# Example of using the aggregationFlag parameter "Percentiles"
plotList <- runPlot(
  nameOfplotFunction = "plotTimeProfiles",
  configTableSheet = "myTimeProfile",
  projectConfiguration = projectConfiguration,
  suppressExport = TRUE,
  inputs = list(
    scenarioResults = scenarioResults,
    dataObserved = observedData,
    aggregationFlag = c("Percentiles"),
    percentiles = c(0.025, 0.5, 0.975)
  )
)

## End(Not run)
```

`ProjectConfigurationRF` *ProjectConfiguration*

Description

An object storing configuration used project-wide

Super class

`esqlabsR::ProjectConfiguration` -> `ProjectConfigurationRF`

Active bindings

`addOns` list with non default configurations

Methods

Public methods:

- ProjectConfigurationRF$new()
- ProjectConfigurationRF$print()
- ProjectConfigurationRF$addAddOnFileToConfiguration()
- ProjectConfigurationRF$addAddOnFolderToConfiguration()
- ProjectConfigurationRF$clone()

Method `new()`**:** Initializes the ProjectConfiguration object with a specified configuration file path.

Adds new line to configuration xlsx

Read configuration from file Initialize

Usage:

```
ProjectConfigurationRF$new(projectConfigurationFilePath = character())
```

Arguments:

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

Method `print()`**:** print prints a summary of the Project Configuration.

Usage:

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

Arguments:

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

Method `addAddOnFileToConfiguration()`**:** Adds an add-on file to the project configuration.

Usage:

```
ProjectConfigurationRF$addAddOnFileToConfiguration(
  property,
  value,
  description,
  templatePath
)
```

Arguments:

`property` A string representing the name of the property to add.

`value` A string representing the basename of the file to add.

`description` A string providing a description of the property.

`templatePath` A string representing the path of the file to add.

Method `addAddOnFolderToConfiguration()`**:** Adds an add-on file to the project configuration.

Usage:

```
ProjectConfigurationRF$addAddOnFolderToConfiguration(
  property,
  value,
  description
)
```

Arguments:

`property` A string representing the name of the property to add.

`value` A string representing the path of the value to add.

`description` A string providing a description of the property.

`templatePath` A string representing the path of the template file.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other project initialization: `createProjectConfiguration()`, `getQCpassedEnvironmentVariable()`, `initProject()`, `setWorkflowOptions()`

`readObservedDataByDictionary` *Read data by dictionary*

Description

This function reads and processes data based on the provided project configuration.

Usage

```
readObservedDataByDictionary(
  projectConfiguration,
  spreadData = TRUE,
  dataClassType = c("timeprofile", "pkParameter"),
  fileIds = NULL
)
```

Arguments

`projectConfiguration`	An object containing project configuration details, including the path to the data importer configuration file.
`spreadData`	If TRUE, information derived from observed data, such as identifiers and biometrics, is spread to other tables.
`dataClassType`	A character string indicating the type of data class to process. Options are "timeprofile" or "pkParameter".
`fileIds`	A character vector with file identifiers to be selected, if NULL (default) all are selected.

Value

A ‘data.table‘ containing the processed data based on the dictionary. The structure includes relevant columns defined in the data dictionary.

See Also

Other observed data processing: `addBiometricsToConfig()`, `aggregateObservedDataGroups()`, `convertDataCombinedToDataTable()`, `convertDataTableToDataCombined()`, `updateDataGroupId()`, `updateOutputPathId()`, `validateObservedData()`

`renderWord` *converts .Rmd file to word*

Description

converts .Rmd file to word

Usage

```
renderWord(
  fileName,
  wordConversionTemplate = NULL,
 customStyles = list(FigureCaption = NULL, FigureFootnote = NULL, TableCaption = NULL,
    TableFootnote = NULL),
  ...
)
```

Arguments

`fileName`	name of .Rmd file to convert to word format (".docx")
`wordConversionTemplate`	template used for conversion
`customStyles`	list of custom styles usable for figure and table captions and footnotes available list elements for styles are: ‘FigureCaption‘, ‘FigureFootnote‘, ‘TableCaption‘ and ‘TableFootnote‘ The selected styles should be defined in the ‘wordConversionTemplate‘
`...`	passed to render

Examples

```
## Not run:
# Example with customstyles for Footnotes
renderWord(
  fileName = "myReport.Rmd",
  wordConversionTemplate = "path/to/template/myTemplate.docx",
 customStyles = list(FigureFootnote = "myFootnoteFormat", TableFootnote = "myFootnoteFormat")
)
```

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

`runAndSaveScenarios` *Run and save scenarios*

Description

This function simulates a list of scenarios and saves the results. If results already exist for a scenario, it will overwrite them based on the provided options.

Usage

```
runAndSaveScenarios(
  projectConfiguration,
  scenarioList,
  simulationRunOptions = NULL,
  ...
)
```

Arguments

`projectConfiguration`	Configuration for the project, containing paths and settings necessary to run the simulations and save the results.
`scenarioList`	Named 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.
`...`	Additional arguments passed to 'esqlabsR::saveScenarioResults'.

Value

A list containing the simulation results for each scenario that was run.

See Also

Other scenario management: `calculatePKParameterForScenarios()`, `createScenarios.wrapped()`, `loadPKParameter()`, `loadScenarioResultsToFramework()`, `runOrLoadScenarios()`

Examples

```
## Not run:
runAndSaveScenarios(
  projectConfiguration = myProjectConfig,
  scenarioList = myScenarioList,
  simulationRunOptions = myRunOptions
)
```

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

`runOrLoadScenarios` *Run or load scenarios*

Description

This function checks if the simulation results for scenarios already exist. If they do, it loads them; otherwise, it runs the scenarios and saves the results.

Usage

```
runOrLoadScenarios(
  projectConfiguration,
  scenarioList,
  simulationRunOptions = NULL,
  ...
)
```

Arguments

`projectConfiguration`	Configuration for the project, containing paths and settings necessary to run the simulations and load the results.
`scenarioList`	Named list of Scenario objects to be managed.
`simulationRunOptions`	Object of type 'SimulationRunOptions' that will be passed to simulation runs. If 'NULL', default options are used.
`...`	Additional arguments passed to 'runAndSaveScenarios'.

Value

A list containing the simulation results for each scenario that was loaded or run.

See Also

Other scenario management: `calculatePKParameterForScenarios()`, `createScenarios.wrapped()`, `loadPKParameter()`, `loadScenarioResultsToFramework()`, `runAndSaveScenarios()`

runPlot *Run Plot Function*

Description

This function generates .Rmd files for creating plots based on user-defined configurations. It reads the specified configuration table, evaluates the selected plot function, and manages the output of plots and tables.

Usage

```
runPlot(
  projectConfiguration,
  nameOfplotFunction,
  configTableSheet = NULL,
  rmdName = configTableSheet,
  plotNames = NULL,
  suppressExport = FALSE,
  digitsOfSignificanceCSVDisplay = 3,
  inputs = list()
)
```

Arguments

projectConfiguration	A ProjectConfiguration object containing the project configuration settings. This should include paths to input files and output directories.
nameOfplotFunction	The name of the plot-function as character. The package provide a set of functions which can be addressed, but is also possible to generate a customized function (see details). Available are - plotTimeProfiles - plotPKBoxwhisker - plotPKForestAggregatedAbsoluteValues - plotPKForestPointEstimateOfAbsoluteValues - plotPKForestAggregatedRatios - plotPKForestPointEstimateOfRatios for more details check the help of these functions
configTableSheet	A character string representing the name of the sheet in the 'Plots' configuration table from which to read plot configurations. It should have at least the columns 'Header','Levels' and 'PlotName'
rmdName	A character string specifying the name of the resulting rmd and the sub-folder where results will be saved. The default value will be the sheet name of the plot configuration.
plotNames	A character vector of plot names to filter which plots should be generated. Default is NULL. If provided, 'suppressExport' is set to TRUE, all plots specified in the configuration will be processed and provided as list, but not exported to the Rmd file.

`suppressExport` A logical value indicating whether to suppress the export of the Rmd file. If TRUE, the function will return the generated plots as a list without creating the Rmd file

`digitsOfSignificanceCSVDisplay`
digits Of significance used for the display in the .Rmd for tables, which are exported as .csv

`inputs` A list of additional inputs for the plot function. This can include any parameters that the selected plot function requires.

Details

The ‘runPlot‘ function is designed to facilitate the generation of plots based on configurations defined in a specified configuration table. The function will: - Load the configuration table for plots based on the specified sheet name. - Validate the configuration and ensure that all required parameters are provided. - Execute the designated plot function. - Save the generated plots in the specified subfolder named rmdName, if export is not suppressed.

If the ‘plotNames‘ parameter is set, the function will suppress the Rmd export and return a list of plots generated instead. This is useful when you want to generate specific plots without creating an accompanying .Rmd file for fast check in the daily work.

Users can utilize the ‘openFigureTemplate‘ function to open a template for creating custom plot functions. This function can be called directly: ‘openFigureTemplate().‘ Additionally, the template is available as an RStudio Add-in for easy access.

Value

An invisible list of plots generated by the function. Each element in the list corresponds to a plot created during the execution. If ‘plotNames‘ is provided, the Rmd export is suppressed, and only the plot list is returned.

Examples

```
## Not run:
# Run the plot generation function with Rmd export
runPlot(
  projectConfiguration = projectConfiguration,
  nameOfplotFunction = "plotTimeProfiles",
  configTableSheet = "TimeProfiles",
  inputs = list(dataObserved = dataObserved, scenarioResults = scenarioResults)
)

# Run the plot generation function without Rmd export, returning a plot list
plotList <- runPlot(
  projectConfiguration = projectConfiguration,
  nameOfplotFunction = "plotTimeProfiles",
  configTableSheet = "TimeProfiles",
  plotNames = c("Plot1", "Plot2"),
  inputs = list(dataObserved = dataObserved, scenarioResults = scenarioResults)
)

# open the template for custom plot functions
```

```
openFigureTemplate()

## End(Not run)
```

`runSensitivityAnalysisForScenarios` *Run Sensitivity Analysis for Specified Scenarios*

Description

This function executes sensitivity analysis for the specified scenarios in the project configuration. It checks for the presence of the sensitivity file and runs the analysis, saving the results to CSV files.

Usage

```
runSensitivityAnalysisForScenarios(
  projectConfiguration,
  scenarioList,
  scenarioNames = NULL,
  sensitivitysheet,
  sensitivityAnalysisRunOptions =
    ospsuite::SensitivityAnalysisRunOptions$new(showProgress = TRUE),
  overwrite = TRUE
)
```

Arguments

`projectConfiguration`	An object representing the project configuration.
`scenarioList`	A list of scenarios to analyze.
`scenarioNames`	A vector of scenario names to run the analysis on. Defaults to NULL.
`sensitivitysheet`	The name of the sheet in the sensitivity Excel file to use for analysis.
`sensitivityAnalysisRunOptions`	Options for running the sensitivity analysis. Defaults to an instance of 'SensitivityAnalysisRunOptions' with 'showProgress' set to TRUE.
`overwrite`	A logical indicating whether to overwrite existing results. Defaults to TRUE.

saveSessionInfo *Save Session Info*

Description

This function can be called at the end of your script to save the session information, including the loaded packages and R version, into a log file. The path for the log file has to be initialized by the 'initLogfunction'

Usage

```
saveSessionInfo()
```

See Also

Other log file management: `addMessageToLog()`, `captureLog()`, `initLogfunction()`, `setShowLogMessages()`, `writeTableToLog()`, `writeToLog()`

Examples

```
## Not run:
# Save session info to log file
saveSessionInfo()

## End(Not run)
```

setExportAttributes *Set Export Attributes for Plot and Table Objects*

Description

This function prepares plot and table objects by setting their export attributes, such as caption and footnote lines, which will be used when exporting through the 'PlotManager' class.

Usage

```
setExportAttributes(
  object,
  caption = NULL,
  footNoteLines = NULL,
  exportArguments = NULL
)
```

Arguments

object	An object (e.g., a plot or table) for which export attributes will be set.
caption	A character string specifying the caption for the object. If NULL, no caption will be set.
footNoteLines	A character vector specifying footnote lines for the object. If, no footnote lines will be set.
exportArguments	A list of additional arguments such as width or height passed on to 'ospsuite.plots::exportPlot'.

Value

The modified object with the specified export attributes set.

---

setHeadersToLowerCase *Convert Data Table Column Names to Lowercase*

Description

This function takes a data.table and converts the first letter of all column names to lowercase.

Usage

```
setHeadersToLowerCase(dt)
```

Arguments

dt	A data.table object whose column names need to be converted to lowercase.

Value

The modified data.table with updated column names.

See Also

Other function to read from and write to xlsx: splitInputs(), xlsxAddDataUsingTemplate(), xlsxAddSheet(), xlsxCloneAndSet(), xlsxReadData(), xlsxWriteData()

Examples

```
dt <- data.table::data.table(FirstName = c("John", "Jane"), LastName = c("Doe", "Smith"))
dtLower <- setHeadersToLowerCase(dt)
print(names(dtLower)) # Result: c("firstname", "lastname")
```

setShowLogMessages *Function to switch the display of log messages on the console on and off*

Description

Function to switch the display of log messages on the console on and off

Usage

```
setShowLogMessages(verbose = TRUE)
```

Arguments

verbose	boolean, if true log message will be shown

See Also

Other log file management: addMessageToLog(), captureLog(), initLogfunction(), saveSessionInfo(), writeTableToLog(), writeToLog()

setupVirtualTwinPopConfig *Setup Virtual Twin Population Configuration*

Description

This function sets up a configuration sheet for virtual twin populations based on the provided project configuration and observed data.

Usage

```
setupVirtualTwinPopConfig(projectConfiguration, dataObserved, groups = NULL)
```

Arguments

projectConfiguration	A list containing project configuration details, including file paths for populations and scenarios.
dataObserved	A data object containing observed data, which may be of class "DataCombined".
groups	A character vector of group names to filter the virtual twin population. Defaults to NULL.

Value

Returns NULL invisibly if no groups are available for virtual twin population creation. Modifies the workbook specified in projectConfiguration.

See Also

Other population export: `exportRandomPopulations()`, `exportVirtualTwinPopulations()`

`setWorkflowOptions` *Sets options for a reporting frame work workflow.*

Description

Sets options for a reporting frame work workflow.

Usage

```
setWorkflowOptions(isValidRun = NULL)
```

Arguments

`isValidRun`	A logical value indicating if the run is valid. If TRUE, options are set for a valid run; if FALSE, options are set for an invalid run.

Details

This function configures options for a reporting framework workflow, which is typically applied in two distinct phases:

1. Exploratory Analysis: This phase involves preparation of outputs that are not yet qualified. In this case: - Watermarks are not applied. - Helper functions for building configuration tables are allowed.

2. Production of Final Output: This phase involves generating the "final" output (a valid run). In this case: - Figures should not have watermarks. - Functions that manipulate inputs are not allowed. - The workflow will stop if an error occurs during execution.

Relevant Options:

- 'ospsuite.plots.watermark_enabled': Set to TRUE when 'isValidRun' is FALSE to display watermarks on figures, and FALSE when 'isValidRun' is TRUE.

- 'OSPSuite.RF.skipFailingPlots': Set to TRUE when 'isValidRun' is FALSE to skip plots that fail to generate, and FALSE when 'isValidRun' is TRUE.

- 'OSPSuite.RF.stopHelperFunction': Set to TRUE when 'isValidRun' is TRUE to stop the execution of helper functions during valid runs.

See Also

Other project initialization: `ProjectConfigurationRF`, `createProjectConfiguration()`, `getQCpassedEnvironmentVar`
`initProject()`

splitInputs *Split the elements of a vector by comma*

Description

This function takes an original vector as input and splits its elements by comma.

Usage

```
splitInputs(originalVector)
```

Arguments

originalVector	The original vector to be split.

Value

A vector containing the split elements.

See Also

Other function to read from and write to xlsx: setHeadersToLowerCase(), xlsxAddDataUsingTemplate(), xlsxAddSheet(), xlsxCloneAndSet(), xlsxReadData(), xlsxWriteData()

Examples

```
## Not run:
originalVector <- c("group1, group2", "group3")
splitInputs(originalVector)
# Result: c('group1', 'group2', 'group3')

## End(Not run)
```

TestProjectBuilder *TestProjectBuilder*

Description

Manages the creation and writing test project, which can be used for this package or packages which use the reporting framework as base to build a test project

Super class

ospsuite.utils::Printable -> TestProjectBuilder

Active bindings

`rootDirectory` root directory of test project. This function synchronizes files from one directory to another, copying only the files that are not already present. This function updates the display names for scenarios and output paths in the project configuration.

Methods

Public methods:

- `TestProjectBuilder$new()`
- `TestProjectBuilder$iniTestProject()`
- `TestProjectBuilder$setupRandomPopulations()`
- `TestProjectBuilder$setupSimulations()`
- `TestProjectBuilder$setUpSensitivity()`
- `TestProjectBuilder$addRandomTPData()`
- `TestProjectBuilder$addRandomPKData()`
- `TestProjectBuilder$mockManualEditingsCleanup()`
- `TestProjectBuilder$mockManualEditingsPopulation()`
- `TestProjectBuilder$mockManualEditingsScenario()`
- `TestProjectBuilder$mockManualEditingsPKParameter()`
- `TestProjectBuilder$mockManualEditingsIndividuals()`
- `TestProjectBuilder$mockManualEditingsSensitivity()`
- `TestProjectBuilder$clone()`

Method `new()`**:** Initialize a new instance of the class.

Usage:

```
TestProjectBuilder$new()
```

Method `iniTestProject()`**:** A short description...
This function initializes the project structure and configuration for the test project.

Usage:

```
TestProjectBuilder$iniTestProject(rootDirectory, verbose = FALSE)
```

Arguments:

`rootDirectory` A character string specifying the root directory for the test project.

`verbose` A logical value indicating whether to print messages about the process.

Returns: A ProjectConfiguration object containing the details of the initialized project.

Method `setupRandomPopulations()`**:** This function sets up populations for the test project by copying necessary files and exporting defined populations.

Usage:

```
TestProjectBuilder$setupRandomPopulations(
  projectConfiguration,
  populationNames,
  writeTestData = FALSE,
  templateDir = NULL
)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing the project configuration details.

`populationNames` A character vector of population names to be copied.

`writeTestData` A logical value indicating whether newly created test data should be filed to the inst directory of the package.

`templateDir` character string specifying the installation directory for the package.

Returns: NULL

Method `setupSimulations()`**:** This function sets up simulations for the test project by synchronizing necessary directories and running initialized scenarios to calculate pharmacokinetic (PK) parameters.

Usage:

```
TestProjectBuilder$setupSimulations(
  projectConfiguration,
  scenarioList,
  writeTestData,
  templateDir = NULL
)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing the project configuration details.

`scenarioList` A list of scenarios initialized for the project.

`writeTestData` A logical value indicating whether newly created test data should be filed to the inst directory of the package.

`templateDir` A character string specifying the installation directory for the package.

Returns: A list containing the results from running the scenarios. Set Up Sensitivity Analysis This function sets up sensitivity analysis for the test project based on the defined scenarios.

Method `setUpSensitivity()`**:**

Usage:

```
TestProjectBuilder$setUpSensitivity(
  projectConfiguration,
  scenarioList,
  scenarioName,
  writeTestData,
  parameterList,
  sensitivitySheet = "smallSelection",
  templateDir = NULL
)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing project configuration details.

`scenarioList` A list of scenarios initialized for the project.

`scenarioName` The name of the scenario used for sensitivity

`writeTestData` A logical value indicating whether newly created test data should be filed to the inst directory of the package.

`parameterList` A named list where each name corresponds to a parameter in the Excel sheet, and each value is the new value to set for that parameter.

`sensitivitySheet` Name of sheet with parameter selection

`templateDir` A character string specifying the installation directory for the package.

Returns: A ProjectConfiguration object containing the updated project configuration.

Method `addRandomTPData()`**:** This function adds random time profile data to the test project for selected individuals.

Usage:

```
TestProjectBuilder$addRandomTPData(
  projectConfiguration,
  scenarioResults,
  ids,
  timePoints = c(15, 30, 45, 60, 90, c(3, 6, 9, 12, 18, 24) * 60),
  outputPathIds = c("Plasma", "CYP3A4total", "CYP3A4Liver"),
  sdShift = 0.05,
  lloqValue = 0.1
)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing project configuration details.

`scenarioResults` List of scenario results.

`ids` A vector of individual IDs to include in the random data.

`timePoints` A numeric vector specifying the time points at which to generate data (default is c(15, 30, 45, 60, 90, c(3, 6, 9, 12, 18, 24) * 60)).

`outputPathIds` A character vector specifying the output path IDs to include in the random data (default is c('Plasma', 'CYP3A4total', 'CYP3A4Liver')).

`sdShift` A numeric value indicating the standard deviation shift applied to the random values (default is 0.05).

`lloqValue` A numeric value representing the lower limit of quantification (default is 0.1).

Returns: Invisible NULL. This function adds random pharmacokinetic (PK) data to the test project, including shifting values and calculating statistics for selected individuals.

Method `addRandomPKData()`**:**

Usage:

```
TestProjectBuilder$addRandomPKData(
  projectConfiguration,
  pkParameterDT,
  ids,
  outputPathIds = c("Plasma", "CYP3A4total", "CYP3A4Liver"),
  sdShift = 0.05
)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing project configuration details.

`pkParameterDT` A data.table containing PK parameters and values.

`ids` A vector of individual IDs to include in the random data.

`outputPathIds` A character vector specifying the output path IDs to include in the random data (default is c('Plasma', 'CYP3A4total', 'CYP3A4Liver')).

`sdShift` A numeric value indicating the standard deviation shift applied to the random values (default is 0.05).

Returns: Invisible NULL.

Method `mockManualEditingsCleanup()`**:** This function cleans up the project configuration files by clearing existing data from relevant Excel sheets.

Usage:

```
TestProjectBuilder$mockManualEditingsCleanup(projectConfiguration)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing project configuration details.

Returns: NULL

Method `mockManualEditingsPopulation()`**:** This function adds biometric ranges for virtual pediatric and one adult populations to the configuration files

Usage:

```
TestProjectBuilder$mockManualEditingsPopulation(
  projectConfiguration,
  randomPops = data.table(populationName = c("adults", "toddler", "children",
    "school-children", "adolescents"), species = "Human", population =
   "European_ICRP_2002", numberOfIndividuals = 100, proportionOfFemales = 50, ageMin =
   c(20, 0.5, 2, 6, 12), ageMax = c(40, 2, 6, 12, 18), weightUnit = "kg", heightUnit =
   "cm", bMIUnit = "kg/m²", `protein Ontogenies` = "CYP3A4:CYP3A4, UGT1A4:UGT1A4")
)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing project configuration details.

`randomPops` data.table with content for sheet Demographics in Population.xlsx

Returns: NULL

Method `mockManualEditingsScenario()`**:** This function sets up scenarios based on the defined populations and adds PK Parameters and output paths to the project configuration.

Usage:

```
TestProjectBuilder$mockManualEditingsScenario(
  projectConfiguration,
  dtTestScenarios,
  pKParameter = NULL
)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing project configuration details.

`dtTestScenarios` 'data.table' with content for scenarios

`pKParameter` vector with pkParameter sheets

Returns: NULL

Method `mockManualEditingsPKParameter()`**:** This function adds PK parameters to the project configuration based on templates.

Usage:

```
TestProjectBuilder$mockManualEditingsPKParameter(projectConfiguration)
```

Arguments:

`projectConfiguration` A ProjectConfiguration object containing project configuration details.

Returns: NULL

Method `mockManualEditingsIndividuals()`**:** This function adjust the bio-metrics with in individuals and exports virtual twin populations.

Usage:

```
TestProjectBuilder$mockManualEditingsIndividuals(projectConfiguration)
```

Arguments:

`projectConfiguration` A 'ProjectConfiguration' object containing project configuration details.

Returns: 'data.table' with content of sheet 'VirtualTwinPopulation'

Method `mockManualEditingsSensitivity()`**:** This function modifies specific parameters in a given Excel sheet based on the provided parameter list and saves the changes back to the workbook.

Usage:

```
TestProjectBuilder$mockManualEditingsSensitivity(
  projectConfiguration,
  sheetName,
  parameterList
)
```

Arguments:

`projectConfiguration` A list containing project configuration details, including the path to the sensitivity file.

`sheetName` A string representing the name of the sheet in the Excel workbook that will be edited.

`parameterList` A named list where each name corresponds to a parameter in the Excel sheet, and each value is the new value to set for that parameter.

Returns: NULL This function is called for its side effects (i.e., modifying the Excel file).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`TIMERANGE` *enumeration keys for Time range shortcut*

Description

enumeration keys for Time range shortcut

Usage

```
TIMERANGE
```

Format

An object of class `list` of length 3.

See Also

Other enumerations: `BIOMETRICUNITS`, `DATACLASS`, `EXPORTDIR`

`updateDataGroupId` *Update Data Group IDs in Project Configuration*

Description

This function updates the Data Groups sheet in an Excel workbook based on a provided data table. It reads existing data group IDs, merges them with new data, and ensures that the identifiers remain unique.

Usage

```
updateDataGroupId(projectConfiguration, dataDT)
```

Arguments

`projectConfiguration`	An object of class 'ProjectConfiguration' containing configuration details
`dataDT`	A 'data.table' containing the new data to be added to the Data Groups sheet. It must have columns that can be matched with existing identifiers, including "group", "studyId", and "studyArm".

Details

The function loads the existing Data Groups from the specified Excel workbook, selects relevant columns from the input data, and appends new entries while maintaining the uniqueness of the identifiers. The updated data is then written back to the workbook.

Value

NULL This function updates the Excel workbook in place and does not return a value. It is called for its side effects.

See Also

Other observed data processing: `addBiometricsToConfig()`, `aggregateObservedDataGroups()`, `convertDataCombinedToDataTable()`, `convertDataTableToDataCombined()`, `readObservedDataByDictionary()`, `updateOutputPathId()`, `validateObservedData()`

---

updateOutputPathId *Update Output Path IDs in Project Configuration*

Description

This function updates the Outputs sheet in an Excel workbook based on a provided data table, ensuring that output path IDs remain unique.

Usage

```
updateOutputPathId(projectConfiguration, dataDT)
```

Arguments

`projectConfiguration`	An object of class 'ProjectConfiguration' containing configuration details.
`dataDT`	A 'data.table' containing new output path IDs to be added to the Outputs sheet. It must include a column named "outputPathId".

Details

The function loads the existing Output Paths from the specified Excel workbook, extracts unique output path IDs from the input data, and appends them while maintaining uniqueness. The updated data is then written back to the workbook.

Value

NULL This function updates the Excel workbook in place and does not return a value. It is called for its side effects.

See Also

Other observed data processing: addBiometricsToConfig(), aggregateObservedDataGroups(), convertDataCombinedToDataTable(), convertDataTableToDataCombined(), readObservedDataByDictionary(), updateDataGroupId(), validateObservedData()

`validateAtleastOneEntry` *check if at least one of the following columns is selected*

Description

check if at least one of the following columns is selected

Usage

```
validateAtleastOneEntry(configTablePlots, columnVector)
```

Arguments

`configTablePlots`	'data.table' Configuration table without header lines
`columnVector`	vector of columns to check

`validateConfigTablePlots` *validate types of plot configuration tables*

Description

validate types of plot configuration tables

Usage

```
validateConfigTablePlots(
  configTablePlots,
  charactersWithoutMissing = NULL,
  charactersWithMissing = NULL,
  numericColumns = NULL,
  logicalColumns = NULL,
  numericRangeColumns = NULL,
  subsetList = list()
)
```

Arguments

`configTablePlots`	'data.table' configuration table without header lines
`charactersWithoutMissing`	vector with character columns, where no missing value is allowed
`charactersWithMissing`	vector with character column, where values may missing
`numericColumns`	vector with numeric columns
`logicalColumns`	vector with booleans
`numericRangeColumns`	vector with columns where entries must be evaluate to a numeric of length 2
`subsetList`	list where each entry is a list: list(cols = 'vector with columns', allowedValues = vector with allowed values)

`validateDistributionVsDemographicsConfig` *Validate Distribution vs Demographics Configuration*

Description

Validates the configuration table for distribution vs demographics plots.

Usage

```
validateDistributionVsDemographicsConfig(configTable, scenarioList, ...)
```

Arguments

`configTable`	A data.table containing the configuration table.
`scenarioList`	List of scenarios
`...`	Additional arguments for validation.

Value

NULL (invisible).

See Also

Other plot configuration validation function: `validateHistogramsConfig()`, `validatePKBoxwhiskerConfig()`, `validatePKForestAggregatedAbsoluteValuesConfig()`, `validatePKForestAggregatedRatiosConfig()`, `validatePKForestPointEstimateOfAbsoluteValuesConfig()`, `validatePKForestPointEstimateOfRatiosConfig()`, `validateSensitivityConfig()`, `validateTimeProfilesConfig()`

Other functions to generate plots displaying distribution vs demographics: `addDefaultConfigForHistograms()`, `plotDistributionVsDemographics()`

`validateGroupConsistency` *Validate Consistency of Values Within Groups*

Description

This function checks whether specified columns in a data table have consistent values within groups defined by one or more grouping columns. If any column contains more than one unique value within a group, an error is raised.

Usage

```
validateGroupConsistency(dt, valueColumns, groupingColumns = "plotName")
```

Arguments

`dt`	A data.table or data.frame containing the data to be validated.
`valueColumns`	A character vector of column names to check for consistency.
`groupingColumns`	A character vector of column names that define the groups. Default is 'plotName'.

Value

Returns NULL (invisible) if all checks pass. Raises an error if any value column contains inconsistent values within a group.

`validateHeaders` *checks if config table header and plot rows are strict separated*

Description

checks if config table header and plot rows are strict separated

Usage

```
validateHeaders(configTable)
```

Arguments

`configTable`	'data.table' configuration table to check

Value

configuration table without header lines

validateHistogramsConfig *Validate Histograms Configuration*

Description

Validates the configuration table for histogram plots.

Usage

```
validateHistogramsConfig(configTable, ...)
```

Arguments

configTable	A data.table containing the configuration table.
...	Additional arguments for validation.

Value

NULL (invisible).

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validatePKBoxwhiskerConfig(), validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestAggreg validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig(), validateSensitivityConfig(), validateTimeProfilesConfig()

Other functions to generate histograms: addDefaultConfigForDistributionsVsDemographics(), plotHistograms()

validateNumericVectorColumns *validate Numeric Range Columns*

Description

Validates numeric range columns in the provided data frame.

Usage

```
validateNumericVectorColumns(columns, data, ...)
```

Arguments

columns	A vector of column names to validate.
data	A data frame containing the columns to validate.
...	additionally parameters parsed to checkmate::assertNumeric

validateObservedData *Validate Observed Data*

Description

Checks the integrity and validity of observed data in a ‘data.table‘. This function verifies the presence of required attributes, checks for duplicates, and ensures that there are no missing or empty values in the relevant columns. Additionally, it checks for ambiguities in the Y unit and validates error type columns.

Usage

```
validateObservedData(dataDT, dataClassType)
```

Arguments

dataDT	A ‘data.table‘ containing observed data with the following relevant columns: - ‘individualId‘: Unique identifier for individuals. - ‘group‘: Group identifier. - ‘outputPathId‘: Identifier for output paths. - ‘xValues‘: Values for the x-axis. - ‘yUnit‘: Unit for the Y values. - ‘yErrorType‘: Type of error for Y values (optional). - ‘yErrorValues‘: Values representing errors (optional). - ‘yMin‘, ‘yMax‘: Minimum and maximum Y values (optional). - ‘lloq‘: Lower limit of quantification (optional). - ‘nBelowLLOQ‘: Count of values below the lower limit of quantification (optional).
dataClassType	A string indicating the type of data class (e.g., "timeprofile" or "pkParameter").

Details

The function performs several checks, including: - Ensuring all data columns have the appropriate attributes. - Verifying that the data is unique based on specified identifier columns. - Checking for NA or empty values in all relevant columns. - Ensuring that the Y unit is consistent across output paths. - Validating the presence of necessary columns based on the Y error type.

Value

NULL This function performs checks and stops execution if any validation fails. It does not return a value.

See Also

Other observed data processing: addBiometricsToConfig(), aggregateObservedDataGroups(), convertDataCombinedToDataTable(), convertDataTableToDataCombined(), readObservedDataByDictionary(), updateDataGroupId(), updateOutputPathId()

validatePKBoxwhiskerConfig

Validate PK Box-and-Whisker Configuration Table

Description

Validates the configuration table for PK box-and-whisker plots.

Usage

```
validatePKBoxwhiskerConfig(configTable, pkParameterDT, ...)
```

Arguments

configTable	A data.table containing the configuration table.
pkParameterDT	A data.table containing PK parameter data.
...	Additional arguments for validation.

Value

NULL (invisible).

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validateHistogramsConfig(), validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestAggrega validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig(), validateSensitivityConfig(), validateTimeProfilesConfig()

Other functions to generate box whisker plots: addDefaultConfigForPKBoxwhsikerPlots(), plotPKBoxwhisker()

validatePKForestAggregatedAbsoluteValuesConfig

Validate PK Forest Absolute Values with Variance Configuration

Description

Validates the configuration for PK forest plots with absolute values and variance.

Usage

```
validatePKForestAggregatedAbsoluteValuesConfig(configTable, pkParameterDT, ...)
```

Arguments

configTable	A data.table containing the configuration table.
pkParameterDT	A data.table containing PK parameter data.
...	Additional arguments for validation.

Value

NULL (invisible).

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validateHistogramsConfig(), validatePKBoxwhiskerConfig(), validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig(), validateSensitivityConfig(), validateTimeProfilesConfig()

Other functions to generate forest plots: addDefaultConfigForPKForestPlots(), plotPKForestAggregatedAbsoluteVal, plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimate, validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig()

---

validatePKForestAggregatedRatiosConfig

Validate PK Forest Ratios with Variance Configuration

Description

Validates the configuration for PK forest plots with ratios and variance.

Usage

```
validatePKForestAggregatedRatiosConfig(configTable, pkParameterDT, ...)
```

Arguments

configTable	A data.table containing the configuration table.
pkParameterDT	A data.table containing PK parameter data.
...	Additional arguments for validation.

Value

NULL (invisible).

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validateHistogramsConfig(), validatePKBoxwhiskerConfig(), validatePKForestAggregatedAbsoluteValuesCon validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig(), validateSensitivityConfig(), validateTimeProfilesConfig()

Other functions to generate forest plots: addDefaultConfigForPKForestPlots(), plotPKForestAggregatedAbsoluteVal plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimat validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig validatePKForestPointEstimateOfRatiosConfig()

---

validatePKForestPointEstimateOfAbsoluteValuesConfig

Validate PK Forest Absolute Values with Confidence Intervals Configuration

---

Description

Validates the configuration for PK forest plots with absolute values and confidence intervals.

Usage

```
validatePKForestPointEstimateOfAbsoluteValuesConfig(
  configTable,
  pkParameterDT,
  ...
)
```

Arguments

configTable	A data.table containing the configuration table.
pkParameterDT	A data.table containing PK parameter data.
...	Additional arguments for validation.

Value

NULL (invisible).

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validateHistogramsConfig(), validatePKBoxwhiskerConfig(), validatePKForestAggregatedAbsoluteValuesCon validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfRatiosConfig(), validateSensitivityConfig(), validateTimeProfilesConfig()

Other functions to generate forest plots: addDefaultConfigForPKForestPlots(), plotPKForestAggregatedAbsoluteVal plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimat validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfRatiosConfig()

validatePKForestPointEstimateOfRatiosConfig

Validate PK Forest Ratios with Confidence Intervals Configuration

Description

Validates the configuration for PK forest plots with ratios and confidence intervals.

Usage

```
validatePKForestPointEstimateOfRatiosConfig(configTable, pkParameterDT, ...)
```

Arguments

configTable	A data.table containing the configuration table.
pkParameterDT	A data.table containing PK parameter data.
...	Additional arguments for validation.

Value

NULL (invisible).

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validateHistogramsConfig(), validatePKBoxwhiskerConfig(), validatePKForestAggregatedAbsoluteValuesCon validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig(), validateSensitivityConfig(), validateTimeProfilesConfig()

Other functions to generate forest plots: addDefaultConfigForPKForestPlots(), plotPKForestAggregatedAbsoluteVa plotPKForestAggregatedRatios(), plotPKForestPointEstimateOfAbsoluteValues(), plotPKForestPointEstimat validatePKForestAggregatedAbsoluteValuesConfig(), validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig()

validateSensitivityConfig

Validate Sensitivity Configuration Table

Description

This function checks the headers of the configuration table, validates output IDs and data group IDs for plotting, and ensures the configuration adheres to specified criteria. It also checks for file existence in the output folder.

Usage

```
validateSensitivityConfig(configTable, ...)
```

Arguments

configTable	A data.table containing the configuration for sensitivity analysis.
...	Additional arguments passed to other functions.

Value

A validated data frame containing the configuration table for sensitivity plots.

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validateHistogramsConfig(), validatePKBoxwhiskerConfig(), validatePKForestAggregatedAbsoluteValuesCon validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig(), validateTimeProfilesConfig()

---

validateTimeProfilesConfig

Validation of config table for time profiles plots

Description

Validation of config table for time profiles plots

Usage

```
validateTimeProfilesConfig(
  configTable,
  dataObserved = NULL,
  scenarioResults,
  ...
)
```

Arguments

configTable	Plot configuration table.
dataObserved	A 'data.table' (formatted as produced by 'readObservedDataByDictionary') or 'DataCombined' object containing the observed data to be plotted.
scenarioResults	A list containing simulated scenario results.
...	Additional arguments passed to 'ospsuite.plots::plotTimeprofile'.

See Also

Other plot configuration validation function: validateDistributionVsDemographicsConfig(), validateHistogramsConfig(), validatePKBoxwhiskerConfig(), validatePKForestAggregatedAbsoluteValuesCon validatePKForestAggregatedRatiosConfig(), validatePKForestPointEstimateOfAbsoluteValuesConfig(), validatePKForestPointEstimateOfRatiosConfig(), validateSensitivityConfig()

---

writeTableToLog *write a table to the logfile*

Description

write a table to the logfile

Usage

```
writeTableToLog(dt, filename = "run.log")
```

Arguments

dt	table to log
filename	filename of log

See Also

Other log file management: addMessageToLog(), captureLog(), initLogfunction(), saveSessionInfo(), setShowLogMessages(), writeToLog()

---

writeToLog *Writing to Log*

Description

The ‘writeToLog‘ function is used to append log messages to a log file. The path for the logfile has to be initialized by the function initLogfile

Usage

```
writeToLog(type, msg, filename = NULL)
```

Arguments

type	The type of message (e.g., Error, Info).
msg	The message to be logged.
filename	The name of the log file.

See Also

Other log file management: addMessageToLog(), captureLog(), initLogfunction(), saveSessionInfo(), setShowLogMessages(), writeTableToLog()

Examples

```
## Not run:
# Write a log message
writeToLog(type = "Info", msg = "This is an information message", filename = "run.log")

## End(Not run)
```

`xlsxAddDataUsingTemplate` *Add new data to a config file using a template sheet*

Description

If the template does not exist in the configuration file in the project directory, it is taken from the configuration file of the package installation. In this case, formats are not preserved.

Usage

```
xlsxAddDataUsingTemplate(
  wb,
  templateSheet,
  sheetName,
  dtNewData,
  templateXlsx = "Plots.xlsx"
)
```

Arguments

wb	Current configuration file.
templateSheet	Name of the template sheet.
sheetName	Name of the new sheet.
dtNewData	A ‘data.table‘ with new data.
templateXlsx	A character string specifying the template xlsx file name (default is "Plots.xlsx").

Value

The current configuration file with the added sheet.

See Also

Other function to read from and write to xlsx: setHeadersToLowerCase(), splitInputs(), xlsxAddSheet(), xlsxCloneAndSet(), xlsxReadData(), xlsxWriteData()

Examples

```
## Not run:
wb <- loadWorkbook("config.xlsx")
newData <- data.table(Name = c("Gina", "Hank"), Age = c(29, 33))
xlsxAddDataUsingTemplate(wb, "TemplateSheet", "NewDataSheet", newData)

## End(Not run)
```

`xlsxAddSheet` *Add a new sheet to a workbook with data*

Description

This function wraps 'openxlsx::addWorksheet' and adds data to the newly created sheet. If the sheet already exists, it issues a warning and clears the existing content.

Usage

```
xlsxAddSheet(wb, sheetName, dt)
```

Arguments

`wb`	A workbook object created by 'openxlsx::loadWorkbook()'.
`sheetName`	A character string specifying the name of the sheet to add.
`dt`	A 'data.table' containing the data to be written to the new sheet.

Value

An invisible NULL value. The function performs an action (adding a sheet)

See Also

Other function to read from and write to xlsx: `setHeadersToLowerCase()`, `splitInputs()`, `xlsxAddDataUsingTemplate()`, `xlsxCloneAndSet()`, `xlsxReadData()`, `xlsxWriteData()`

Examples

```
## Not run:
library(openxlsx)
wb <- loadWorkbook("example.xlsx")
data <- data.table(Name = c("Alice", "Bob"), Age = c(30, 25))
xlsxAddSheet(wb, "NewSheet", data)

## End(Not run)
```

`xlsxCloneAndSet` *Clone a sheet and set new content*

Description

This function wraps ‘openxlsx::cloneWorksheet‘ but sets new content in the cloned sheet.

Usage

```
xlsxCloneAndSet(wb, clonedSheet, sheetName, dt)
```

Arguments

`wb`	A workbook object created by ‘openxlsx::loadWorkbook()‘.
`clonedSheet`	A character string specifying the name of the sheet to clone.
`sheetName`	A character string specifying the name of the new sheet.
`dt`	A ‘data.table‘ with new content.

Value

An invisible NULL value. The function performs an action (clone a sheet)

See Also

Other function to read from and write to xlsx: `setHeadersToLowerCase()`, `splitInputs()`, `xlsxAddDataUsingTemplate()`, `xlsxAddSheet()`, `xlsxReadData()`, `xlsxWriteData()`

Examples

```
## Not run:
wb <- loadWorkbook("example.xlsx")
data <- data.table(Name = c("Eve", "Frank"), Age = c(22, 35))
xlsxCloneAndSet(wb, "ExistingSheet", "ClonedSheet", data)

## End(Not run)
```

`xlsxReadData` *Read data from a worksheet*

Description

This function wraps ‘openxlsx::read.xlsx‘ and returns the data as a ‘data.table‘.

Usage

```
xlsxReadData(
  wb,
  sheetName = 1,
  skipDescriptionRow = FALSE,
  alwaysCharacter = c("Group", "Id$", "Ids$"),
  emptyAsNA = TRUE,
  convertHeaders = TRUE
)
```

Arguments

`wb`	A workbook object or a character string specifying the path to the xlsx file.
`sheetName`	A character string specifying the name of the sheet to read.
`skipDescriptionRow`	A logical value or an integer indicating whether the first row (or rows, if an integer) should be treated as description rows and skipped during reading. When set to TRUE, the first row is skipped; when set to a positive integer, that number of rows will be skipped. The 'Comment' column is also excluded from the resulting data table.
`alwaysCharacter`	A character vector with column names or regex patterns that should be returned as character (typically identifiers).
`emptyAsNA`	A logical value. If TRUE, empty strings in character columns are converted to NA. If FALSE NA is returned as empty string. Numeric columns Return always NA
`convertHeaders`	A logical value. If TRUE, column names are converted to start with a lowercase letter.

Value

A ‘data.table‘ containing the sheet data.

See Also

Other function to read from and write to xlsx: `setHeadersToLowerCase()`, `splitInputs()`, `xlsxAddDataUsingTemplate()`, `xlsxAddSheet()`, `xlsxCloneAndSet()`, `xlsxWriteData()`

Examples

```
## Not run:
wb <- loadWorkbook("example.xlsx")
data <- xlsxReadData(wb, "DataSheet", skipDescriptionRow = 1)
print(data)

## End(Not run)
```

`xlsxWriteData` *Write data to a worksheet, clearing existing content*

Description

This function wraps 'openxlsx::writeData', but deletes all current content before writing new data.

Usage

```
xlsxWriteData(wb, sheetName, dt)
```

Arguments

`wb`	A workbook object created by 'openxlsx::loadWorkbook()'.
`sheetName`	A character string specifying the name of the sheet where data will be written.
`dt`	A 'data.table' to write.

Value

An invisible NULL value. The function performs an action (adding data to a sheet)

See Also

Other function to read from and write to xlsx: `setHeadersToLowerCase()`, `splitInputs()`, `xlsxAddDataUsingTemplate(`
`xlsxAddSheet()`, `xlsxCloneAndSet()`, `xlsxReadData()`

Examples

```
## Not run:
wb <- loadWorkbook("example.xlsx")
data <- data.table(Name = c("Charlie", "Dana"), Age = c(28, 32))
xlsxWriteData(wb, "ExistingSheet", data)

## End(Not run)
```

Index