Package 'lixoftConnectors'

October 11, 2024

Type Package

Title R connectors for Lixoft Suite (@Lixoft)

Version 2019.2

Date 2019-09-10

Author LIXOFT

Maintainer LIXOFT <support@lixoft.com>

Depends R (>= 3.0.0), RJSONIO

Collate apiTools.R displayTools.R lixoftEnvironment.R apiManager.R commons-projectManagement.R commons-settings.R commons-scenario.R commons-results.R mlx-settings.R mlx-scenario.R mlx-populationParameters.R mlx-individualModel.R mlx-covariateModel.R mlx-observationModel.R mlx-results.R mlx-core.R mlx-modelBuilding.R pkx-settings.R pkx-scenario.R pkx-results.R smlx-core.R

Description This package provides R connectors for Monolix - PKanalix - Simulx (@Lixoft) to create, edit and run Mlxtran projects from R command prompt.

License [BSD_2_clause + file LICENSE] (license lixoft)

RoxygenNote 6.1.1

NeedsCompilation no

R topics documented:

`abort` *[Monolix - PKanalix] Stop the current task run*

Description

Stop the current task run.

Usage

```
abort()
```

See Also

`runScenario`

Examples

```
## Not run:
abort()

## End(Not run)
```

---

addCategoricalTransformedCovariate
[Monolix] Add categorical transformed covariate

Description

Create a new categorical covariate by transforming an existing one. Transformed covariates cannot be use to produce new covariates. Call getCovariateInformation to know which covariates can be transformed.

Usage

```
addCategoricalTransformedCovariate(...)
```

Arguments

...	A list of comma-separated pairs {transformedCovariateName = { from = (*array<(string)>*)["basicCovariateNames"], transformed = (*array<array<string»*)"transformation"} }

See Also

getCovariateInformation removeCovariate

Examples

```
## Not run:
addCategoricalTransformedCovariate( Country2 = list(reference = "A1",
          from = "Country", transformed = list( A1 = c("A","B"), A2 = c("C")))
          )

## End(Not run)
```

---

addContinuousTransformedCovariate
[Monolix] Add continuous transformed covariate

Description

Create a new continuous covariate by transforming an existing one. Transformed covariates cannot be use to produce new covariates. Call getCovariateInformation to know which covariates can be transformed.

Usage

```
addContinuousTransformedCovariate(...)
```

Arguments

...	A list of comma-separated pairs {transformedCovariateName = (*string*)"transformation"}

See Also

getCovariateInformation removeCovariate

Examples

```
## Not run:
addContinuousTransformedCovariate( tWt2 = "3*exp(Wt)"  )

## End(Not run)
```

`addMixture` *[Monolix] Add mixture to the covariate model Add a new latent covariate to the current model giving its name and its modality number.*

Description

[Monolix] Add mixture to the covariate model

Add a new latent covariate to the current model giving its name and its modality number.

Usage

```
addMixture(...)
```

Arguments

...	A list of comma-separated pairs {latentCovariateName = (*int*)modalityNumber}

See Also

getCovariateInformation removeCovariate

Examples

```
## Not run:
addMixture(lcat = 2)

## End(Not run)
```

`computeChartsData` *[Monolix] Compute the charts data*

Description

Compute and export the charts data o scenario. Notice that it does not impact the current scenario. Call

1. isRunning to check if the scenario is still running and get information about the current task,
2. abort to stop the execution.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE.

Usage

```
computeChartsData(wait = TRUE)
```

Arguments

wait	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.

See Also

isRunning abort

Examples

```
## Not run:
computeChartsData()

## End(Not run)
```

---

computePredictions *[Monolix] Compute predictions from the structural model*

Description

[**MlxCore**][*Prediction*]
Call the monolix prediction function to compute observation models values on observation times for each subject of a set of individuals.

Usage

```
computePredictions(individualParameters, individualIds = NULL)
```

Arguments

individualParameters	Individual parameter values associated to each one of the individual parameters present in the project, for a set of subjects which must be coherent with the list of individuals ids passed in "individualIds" field (ie, this length of the subject set must be the sum of the subject number of all the individuals selected by the "individualIds" field). This input field accepts a dataframe indexed by individual parameter names (columns) and subject indexes (rows).
individualIds	[optional] *vector<int>* Ids of the individuals for which observation models should be computed. By default, all the individuals present in the project are considered.

Value

For each prediction names, a vector giving the computed prediction at observation times for each subject.

See Also

getIndividualParameterModel getEstimatedIndividualParameters

Examples

```
## Not run:
ids = c(1,4)
individualValuesForAllIndiv = getEstimatedIndividualParameters()$saem

predictions = computePredictions( individualParameters = individualValuesForAllIndiv[ids,],
                                  individualIds = ids )

predictions
  -> $Cc
       [3.8,6.75,...,3.4,5.1,...]
       |    id=1    |    id=4    |

## End(Not run)
```

`computeSimulations` *[Simulx] Low level function for predictions computation and data sampling from* `Mlxtran` *models*

Description

Information: This function is intented to be an interface between mlxR package (@Lixoft) and Monolix software (@Lixoft) computation kernel. It doesn't aim to be called directly.

Usage

```
computeSimulations(data, settings = list())
```

Arguments

`data`	A list generated by mlxR package (@Lixoft)
`settings`	A list of optional settings

- `seed` : initialization of the random number generator (integer),
- `load.design` : TRUE/FALSE (if load.design is not defined, a test is automatically performed to check if a new design has been defined),
- `data.in` : TRUE/FALSE (default=FALSE)
- `id.out` : add columns id (when N=1) and group (when #group=1), TRUE/FALSE (default=FALSE)
- `kw.max` : maximum number of trials for generating a positive definite covariance matrix (default = 100)
- `sep` : the field separator character (default = ",")
- `digits` : number of decimal digits in output files (default = 5)
- `disp.iter` : TRUE/FALSE (default = FALSE) display replicate and population numbers
- `replacement` : TRUE/FALSE (default = FALSE) sample id's with/without replacement
- `out.trt` : TRUE/FALSE (default = TRUE) output of simulx includes treatment

Value

A list of data frames. Each data frame is an output of simulx

---

`getCAIndividualParameters`
[PKanalix] Get CA individual parameters

Description

Get the estimated values for each subject of some of the individual CA parameters of the current project.

Usage

```
getCAIndividualParameters(...)
```

Arguments

`...`	(*string*) Name of the individual parameters whose values must be displayed.

Value

A data frame giving the estimated values of the individual parameters of interest for each subject, and a list of their associated statistics.

Examples

```
## Not run:
indivParams = getCAIndividualParameters() # retrieve all the available individual parameters values.
indivParams = getCAIndividualParameters("ka", "V") # retrieve ka and V values for all individuals.

  $parameters->
      id  ka    V
      1   0.8  1.2
      .   ...  ...
      N   0.4  2.2


## End(Not run)
```

getCASettings *[PKanalix] Get the settings associated to the compartmental analysis*

Description

Get the settings associated to the compartmental analysis. Associated settings are:

"weightingCA"	(*string*)	Type of weighting ob
"pool"	(*logical*)	Fit with individual pa
"initialValues" (*list*)	list(param = value, ...): value = initial value of individual parameter param.	
"blqMethod"	(*string*)	Method by which the

Usage

```
getCASettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

setCASettings

Examples

```
## Not run:
getCASettings() # retrieve a list of all the CA methodology settings
getCASettings("weightingca","blqmethod") # retrieve a list containing only the value of the settings whose name

## End(Not run)
```

getConditionalDistributionSamplingSettings *[Monolix] Get conditional distribution sampling settings*

Description

Get the conditional distribution sampling settings. Associated settings are:

"ratio"	(*0< double <1*)	Width of the confidence interval.
"nbMinIterations"	(*int >=1*)	Minimum number of iterations.
"nbSimulatedParameters"	(*int >=1*)	Number of replicates.

Usage

```
getConditionalDistributionSamplingSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

setConditionalDistributionSamplingSettings

Examples

```
## Not run:
getConditionalDistributionSamplingSettings()
# retrieve all the conditional distribution sampling settings
getConditionalDistributionSamplingSettings("ratio","nbMinIterations")
# retrieve only the ratio and nbMinIterations settings values

## End(Not run)
```

---

getConditionalModeEstimationSettings

[Monolix] Get conditional mode estimation settings

Description

Get the conditional mode estimation settings. Associated settings are:

"nbOptimizationIterationsMode"	(*int >=1*)	Maximum number of iterations.
"optimizationToleranceMode"	(*double >0*)	Optimization tolerance.

Usage

```
getConditionalModeEstimationSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

setConditionalModeEstimationSettings

Examples

```
## Not run:
getConditionalModeEstimationSettings()
# retrieve a list of all the conditional mode estimation settings
getConditionalModeEstimationSettings("nbOptimizationIterationsMode")
# retrieve only the nbOptimizationIterationsMode setting value

## End(Not run)
```

---

getContinuousObservationModel

[Monolix] Get continuous observation models information

Description

Get a summary of the information concerning the continuous observation models in the project. The following informations are provided.

- prediction: (*vector*<*string*>) name of the associated prediction
- formula: (*vector*<*string*>) formula applied on the observation
- distribution: (*vector*<*string*>) distribution of the observation in the Gaussian space. The distribution type can be "normal", "logNormal", or "logitNormal".
- limits: (*vector*< *pair*<*double,double*> >) lower and upper limits imposed to the observation. Used only if the distribution is logitNormal. If there is no logitNormal distribution, this field is empty.
- errormodel: (*vector*<*string*>) type of the associated error model
- autocorrelation: (*vector*<*bool*>) defines if there is auto correlation

Call getObservationInformation to get a list of the continuous observations present in the current project.

Usage

```
getContinuousObservationModel()
```

Value

A list associating each continuous observation to its model properties.

See Also

getObservationInformation setObservationDistribution setObservationLimits setErrorModel setAutocorrelation

Examples

```
## Not run:
obsModels = getContinuousObservationModel()
obsModels
 -> $prediction
      c(Conc = "Cc")
    $formula
      c(Conc = "Conc = Cc + (a+b*Cc)*e")
    $distribution
      c(Conc = "logitNormal")
    $limits
      list(Conc = c(0,11.5))
    $errormodel
      c(Conc = "combined1")
    $autocorrelation
      c(Conc = TRUE)

## End(Not run)
```

`getCorrelationOfEstimates`

[Monolix] Get the inverse of the Fisher Matrix

Description

Get the inverse of the last estimated Fisher matrix computed either by all the Fisher methods used during the last scenario run or by the specific one passed in argument.
WARNING: The Fisher matrix cannot be accessible until the Fisher algorithm has been launched once.
The user can choose to display only the Fisher matrix estimated with a specific method.
Existing Fisher methods :

Fisher by Linearization	"linearization"
Fisher by Stochastic Approximation	"stochasticApproximation"

WARNING: Only the methods which have been used during the last scenario run can provide results.

Usage

```
getCorrelationOfEstimates(method = "")
```

Arguments

`method`	[optional](*string*) Fisher method whose results should be displayed. If this field is not specified, the results provided by all the methods used during the last scenario run are displayed.

Value

A list whose each field contains the Fisher matrix computed by one of the available Fisher methods used during the ast scenario run. A matrix is defined as a structure containing the following fields :

rownames	list of row names
columnnames	list of column names
rownumber	number of rows
data	vector<...> containing matrix raw values (column major)

Examples

```
## Not run:
getCorrelationOfEstimates("linearization")
 -> list( linearization = list(data = c(1,0,0,0,1,-0.06,0,-0.06,1),
                               rownumber = 3,
                               rownames = c("Cl_pop","omega_Cl","a"),
                               columnnames = c("Cl_pop","omega_Cl","a")))


getCorrelationOfEstimates()
 -> list(linearization = list(...), stochasticApproximation = list(...) )

## End(Not run)
```

---

getCovariateInformation

Get covariates information

Description

Get the name, the type and the values of the covariates present in the project.

Usage

```
getCovariateInformation()
```

Value

A list containing the following fields :

- name : (*vector<string>*) covariate names
- type : (*vector<string>*) covariate types. Existing types are "continuous", "continuoustransformed", "categorical", "categoricaltransformed"./ In Monolix mode, "latent" covariates are also allowed.
- [Monolix] modalityNumber : (*vector<int>*) number of modalities (for latent covariates only)
- covariate : a data frame giving the values of continuous and categorical covariates for each subject. Latent covariate values exist only if they have been estimated, ie if the covariate is used and if the population parameters have been estimated. Call `getEstimatedIndividualParameters` to retrieve them.

Examples

```
## Not run:
info = getCovariateInformation() # Monolix mode with latent covariates
info
  -> $name
     c("sex","wt","lcat")
  -> $type
     c(sex = "categorical", wt = "continuous", lcat = "latent")
  -> $modalityNumber
     c(lcat = 2)
  -> $covariate
     id   sex    wt
      1    M   66.7
      .    .      .
      N    F   59.0

## End(Not run)
```

`getData` *Get project data*

Description

Get a description of the data used in the current project. Available informations are:

- dataFile (*string*): path to the data file
- header (*array<character>*): vector of header names
- headerTypes (*array<character>*): vector of header types
- observationNames (*vector<string>*): vector of observation names
- observationTypes (*vector<string>*): vector of observation types
- nbSSDoses (*int*): number of doses (if there is a SS column)

Usage

```
getData()
```

Value

A list describing project data.

See Also

`setData`

Examples

```
## Not run:
data = getData()
data
-> $dataFile
     "/path/to/data/file.txt"
   $header
```

```
    c("ID","TIME","CONC","SEX","OCC")
  $headerTypes
    c("ID","TIME","OBSERVATION","CATEGORICAL COVARIATE","IGNORE")
  $observationNames
    c("concentration")
  $observationTypes
    c(concentration = "continuous")

## End(Not run)
```

`getDataSettings` *[PKanalix] Get the data settings associated to the non compartmental analysis*

Description

Get the data settings associated to the non compartmental analysis. Associated settings are:

"urinevolume"	(*string*)	regressor name used as urine volume.
"datatype"	(*list*)	list("obsId" = *string*("plasma" or "urine"). The type of data associated with each *obsId*: observa

Usage

```
getDataSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

setNCASettings

Examples

```
## Not run:
getDataSettings() # retrieve a list of all the NCA methodology settings
getDataSettings("urinevolume") # retrieve a list containing only the value of the settings whose name has been p

## End(Not run)
```

`getEstimatedIndividualParameters` *[Monolix] Get last estimated individual parameter values*

Description

Get the last estimated values for each subject of some of the individual parameters present within the current project.
WARNING: Estimated individual parameters values cannot be accessible until the individual estimation algorithm has been launched once.
NOTE: The user can choose to display only the individual parameter values estimated with a specific method.
Existing individual estimation methods :

Conditional Mean SAEM	"saem"
Conditional Mean	"conditionalMean"
Conditional Mode	"conditionalMode"

WARNING: Only the methods which have been used during the last scenario run can provide estimation results.

Usage

```
getEstimatedIndividualParameters(..., method = "")
```

Arguments

`...`	(*string*) Name of the individual parameters whose values must be displayed. Call `getIndividualParameterModel` to get a list of the individual parameters present within the current project.
`method`	[optional](*string*) Individual parameter estimation method whose results should be displayed. If there are latent covariate used in the model, the estimated modality is displayed too If this field is not specified, the results provided by all the methods used during the last scenario run are displayed.

Value

A data frame giving, for each wanted method, the last estimated values of the individual parameters of interest for each subject with the corresponding standard deviation values.

See Also

`getEstimatedRandomEffects`

Examples

```
## Not run:
indivParams = getEstimatedIndividualParameters()
# retrieve the values of all the available individual parameters for all methods
  -> $saem
      id   Cl     V      ka
```

```
1   0.28  7.71   0.29
.   ...    ...    ...
N   0.1047.62   1.51

indivParams = getEstimatedIndividualParameters("Cl", "V", method = "conditionalMean")
# retrieve the values of the individual parameters "Cl" and "V"
# estimated by the conditional mode method

## End(Not run)
```

`getEstimatedLogLikelihood`

[Monolix] Get Log-Likelihood values

Description

Get the values computed by using a log-likelihood algorithm during the last scenario run, with or without a method-based filter.
WARNING: The log-likelihood values cannot be accessible until the log-likelihood algorithm has been launched once.
The user can choose to display only the log-likelihood values computed with a specific method.
Existing log-likelihood methods :

Log-likelihood by Linearization	"linearization"
Log-likelihood by Important Sampling	"importanceSampling"

WARNING: Only the methods which have been used during the last scenario run can provide results.

Usage

```
getEstimatedLogLikelihood(method = "")
```

Arguments

`method`	[optional](*string*) Log-likelihood method whose results should be displayed. If this field is not specified, the results provided by all the methods used during the last scenario run are retrieved.

Value

A list associating the name of each method passed in argument to the corresponding log-likelihood values computed by during the last scenario run.

Examples

```
## Not run:
getEstimatedLogLikelihood()
 -> list(  linearization = [LL = -170.505, AIC = 350.280, BIC = 365.335] ,
           importanceSampling = [...] )
```

```
getEstimatedLogLikelihood("linearization")
 -> list(  linearization = [LL = -170.505, AIC = 350.280, BIC = 365.335] )

## End(Not run)
```

`getEstimatedPopulationParameters` *[Monolix] Get last estimated population parameter value*

Description

Get the last estimated value of some of the population parameters present within the current project (fixed effects + individual variances + correlations + latent probabilities + error model parameters). WARNING: Estimated population parameters values cannot be accessible until the SAEM algorithm has been launched once.

Usage

```
getEstimatedPopulationParameters(...)
```

Arguments

`...`	[optional] (*array<string>*) Names of the population parameters whose value must be displayed. Call `getPopulationParameterInformation` to get a list of the population parameters present within the current project. If this field is not specified, the function will retrieve the values of all the available population parameters.

Value

A named vector containing the last estimated value of each one of the population parameters passed in argument.

Examples

```
## Not run:
getEstimatedPopulationParameters("V_pop") -> [V_pop = 0.5]
getEstimatedPopulationParameters("V_pop","Cl_pop") -> [V_pop = 0.5, Cl_pop = 0.25]
getEstimatedPopulationParameters() -> [V_pop = 0.5, Cl_pop = 0.25, ka_pop = 0.05]

## End(Not run)
```

`getEstimatedRandomEffects`

[Monolix] Get estimated the random effects

Description

Get the random effects for each subject of some of the individual parameters present within the current project.
WARNING: Estimated random effects cannot be accessible until the individual estimation algorithm has been launched once.
The user can choose to display only the random effects estimated with a specific method.
NOTE: The random effects are defined in the gaussian referential, e.g. if ka is lognormally distributed around ka_pop, eta_i = log(ka_i)-log(ka_pop) Existing individual estimation methods :

Conditional Mean SAEM	"saem"
Conditional Mean	"conditionalMean"
Conditional Mode	"conditionalMode"

WARNING: Only the methods which have been used during the last scenario run can provide estimation results. Please call `getLaunchedTasks` to get a list of the methods whose results are available.

Usage

```
getEstimatedRandomEffects(..., method = "")
```

Arguments

`...`	(*string*) Name of the individual parameters whose random effects must be displayed. Call `getIndividualParameterModel` to get a list of the individual parameters present within the current project.
`method`	[optional](*string*) Individual parameter estimation method whose results should be displayed. If this field is not specified, the results provided by all the methods used during the last scenario run are displayed.

Value

A data frame giving, for each wanted method, the last estimated eta values of the individual parameters of interest for each subject with the corresponding standard deviation values.

See Also

`getEstimatedIndividualParameters`

Examples

```
## Not run:
etaParams = getEstimatedRandomEffects()
# retrieve the values of all the available random effects for all methods
# without the associated standard deviations
  -> $saem
```

```
      id    Cl     V      ka
       1   0.28  7.71   0.29
       .   ...    ...    ...
       N   0.1047.62   1.51

etaParams = getEstimatedRandomEffects("Cl", "V", method = "conditionalMode")
# retrieve the values of the individual parameters "Cl" and "V"
# estimated by the conditional mean from SAEM algorithm

## End(Not run)
```

`getEstimatedStandardErrors`
[Monolix] Get standard errors of population parameters

Description

Get the last estimated standard errors of population parameters computed either by all the Fisher methods used during the last scenario run or by the specific one passed in argument.
WARNING: The standard errors cannot be accessible until the Fisher algorithm has been launched once.
Existing Fisher methods :

Fisher by Linearization	"linearization"
Fisher by Stochastic Approximation	"stochasticApproximation"

WARNING: Only the methods which have been used during the last scenario run can provide results.

Usage

```
getEstimatedStandardErrors(method = "")
```

Arguments

`method`	[optional](*string*) Fisher method whose results should be displayed. If this field is not specified, the results provided by all the methods used during the last scenario run are retrieved

Value

A list associating each retrieved Fisher algorithm method to the standard errors of population parameters computed during its last run.

Examples

```
## Not run:
getEstimatedStandardErrors() -> list( linearization = [...], stochasticApproximation = [...] )
getEstimatedStandardErrors("linearization") -> list( linearization = [...] )

## End(Not run)
```

getGeneralSettings *[Monolix] Get project general settings*

Description

Get a summary of the common settings for Monolix algorithms. Associated settings are:

"autoChains"	(*bool*)	Automatically adjusted the number of chains to have at least a minimum number of sul
"nbChains"	(*int >0*)	Number of chains. *Used only if "autoChains" is set to FALSE.*
"minIndivForChains"	(*int >0*)	Minimum number of individuals by chain.

Usage

```
getGeneralSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

setGeneralSettings

Examples

```
## Not run:
getGeneralSettings() # retrieve a list of all the general settings

getGeneralSettings("nbChains","autoChains")
# retrieve only the nbChains and autoChains settings values.

## End(Not run)
```

getGlobalObsIdToUse *[PKanalix] Get the global observation id used in both the compartmental and non compartmental analysis*

Description

Get the global observation id used in both the compartmental and non compartmental analysis.

Usage

```
getGlobalObsIdToUse()
```

Value

the observation id used in computations.

See Also

setGlobalObsIdToUse

Examples

```
## Not run:
getGlobalObsIdToUse() #

## End(Not run)
```

---

getIndividualParameterModel

[Monolix] Get individual parameter model

Description

Get a summary of the information concerning the individual parameter model. The available informations are:

- name: (*string*) name of the individual parameter
- distribution: (*string*) distribution of the parameter values. The distribution type can be "normal", "logNormal", or "logitNormal".
- formula: (*string*) formula applied on individual parameters distribution
- variability: a list giving, for each variability level, if individual parameters have variability or not
- covariateModel: a list giving, for each individual parameter, if the related covariates are used or not. If no covariate is used, this field is empty.
- correlationBlocks : a list giving, for each variability level, the blocks of the correlation matrix of the random effects. A block is represented by a vector of individual parameter names. If there is no block, this field is empty.

Usage

```
getIndividualParameterModel()
```

Value

A list of individual parameter model properties.

See Also

setIndividualParameterDistribution setIndividualParameterVariability setCovariateModel

Examples

```
## Not run:
indivModel = getIndividualParameterModel()
indivModel
 -> $name
      c("ka","V","Cl")
    $distribution
      c(ka = "logNormal", V = "normal", Cl = "logNormal")
    $formula
      "\\tlog(ka) = log(ka_pop) + eta_ka\\n\\n
       \\tlog(V) = V_pop + eta_V\\n\\n
       \\tlog(Cl) = log(Cl_pop) + eta_Cl\\n\\n"
    $variability
      list( id = c(ka = TRUE, V = FALSE, Cl = TRUE) )
    $covariateModel
      list( ka = c(age = TRUE, sex = FALSE, wt = TRUE),
            V = c(age = FALSE, sex = FALSE, wt = FALSE),
            Cl = c(age = FALSE, sex = FALSE, wt = FALSE) )
    $correlationBlocks
      list( id = c("ka","V","Tlag") )

## End(Not run)
```

---

getLastRunStatus *[Monolix - PKanalix] Get last run status*

Description

Return an execution report about the last run with a summary of the error which could have occurred.

Usage

```
getLastRunStatus()
```

Value

A structure containing

1. a boolean which equals TRUE if the last run has successfully completed,
2. a summary of the errors which could have occurred.

See Also

runScenario abort isRunning

Examples

```
## Not run:
lastRunInfo = getLastRunStatus()
lastRunInfo$status
 -> TRUE
lastRunInfo$report
 -> ""

## End(Not run)
```

getLaunchedTasks *[Monolix] Get tasks with results*

Description

Get a list of the tasks which have results to provide. A task is the association of:

- an algorithm (string)
- a vector of methods (string) relative to this algorithm for the standardErrorEstimation and the loglikelihoodEstimation, TRUE or FALSE for the other one.

Usage

```
getLaunchedTasks()
```

Value

The list of tasks with results, indexed by algorithm names.

Examples

```
## Not run:
tasks = getLaunchedTasks()
tasks
 -> $populationParameterEstimation = TRUE
    $conditionalModeEstimation = TRUE
   $standardErrorEstimation = "linearization"

## End(Not run)
```

getLixoftConnectorsState *Get lixoftConnectors API current state*

Description

Retrieve information about the Lixoft software the lixoftConnectors are currently interfacing with.

Usage

```
getLixoftConnectorsState(quietly = FALSE)
```

Arguments

quietly	*boolean* If TRUE, no warning is raised when lixoftConnectors package is not initialized. Equals FALSE by default.

Value

A structure containing:

- `path`: path to Lixoft installation directory
- `software`: software name
- `version`: Lixoft softwares suite version

`getLixoftEnvInfo` *Get information about LixoftEnvironment object*

Description

Get information about LixoftEnvironment object

Usage

```
getLixoftEnvInfo()
```

`getLogLikelihoodEstimationSettings` *[Monolix] Get LogLikelihood algorithm settings*

Description

Get the loglikelihood estimation settings. Associated settings are:

"nbFixedIterations"	(*int >0*)	Monte Carlo size for the loglikelihood evaluation.
"samplingMethod"	(*string*)	Should the loglikelihood estimation use a given number of freedom d
"nbFreedomDegrees"	(*int >0*)	Degree of freedom of the Student t-distribution. *Used only if "samplin*
"freedomDegreesSampling"	(*vector<int(>0)>*)	Sequence of freedom degrees to be tested. *Used only if "samplingMet*

Usage

```
getLogLikelihoodEstimationSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

`setLogLikelihoodEstimationSettings`

Examples

```
## Not run:
getLogLikelihoodEstimationSettings() # retrieve a list of all the loglikelihood estimation settings
getLogLikelihoodEstimationSettings("nbFixedIterations","samplingMethod")
# retrieve only nbFixedIterations and samplingMethod settings values

## End(Not run)
```

`getMCMCSettings` *[Monolix] Get MCMC algorithm settings*

Description

Get the MCMC algorithm settings of the current project. Associated settings are:

"strategy"	(*vector<int>[3]*)	Number of calls for each one of the three MCMC kernels.
"acceptanceRatio"	(*double*)	Target acceptance ratio.

Usage

```
getMCMCSettings(...)
```

Arguments

`...`	[optional] (string) Names of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

`setMCMCSettings`

Examples

```
## Not run:
getMCMCSettings() # retrieve a list of all the MCMC settings
getMCMCSettings("strategy") # retrieve only the strategy setting

## End(Not run)
```

`getModelBuildingResults` *[Monolix] Get the results of the model building*

Description

Get the results (detailled models) of the model building

Usage

```
getModelBuildingResults()
```

Value

The results of model building All the detailed tried models are returned

- LL: result of -2*Log-Likelihood
- BICc: modified BIC.
- individualModels: (*data.frame*) individual model for each individual parameter. The columns are the covariates and the elements of the data.frame notes if a covariate is used or not for the current parameter.

COSSAC send 2 additional fields:

- tested: (*vector<string>*) first element is the individual parameter and the second one is the covariate. This combination notes if the covariate is tested or not with respect to the previous model.
- bestModel (*boolean*) best model amongst all the tried models according to the chosen criterion.

SAMBA send the error model and covariance model information if there are exist

- errorModels: chosen type for each error model
- covarianceModels: chosen correlations between individual parameters

See Also

`runModelBuilding`

Examples

```
## Not run:
getModelBuildingResults()

## End(Not run)
```

`getModelBuildingSettings` *[Monolix] Get model building settings*

Description

Get the settings that will be used during the run of model building.

Usage

```
getModelBuildingSettings()
```

Value

The list of settings

- covariates: (*vector<string>*) covariate names
- parameters: (*vector<string>*) parameters names
- strategy: (*string*) strategy to search best model ([cossac], samba, scm)
- criterion: (*string*) crtierion to search best model ([BIC], LRT)
- relationships: (*data.frame<parameters, covariates, locked>*) Use to lock relationships between parameters and covariates. By default, all the combinations are possible. This parameter forces the use or not of some combinations. See example where *ka* must have *SEX* and *V* must not have *WEIGHT*
- threshold$lrt: threshold used by criterion LRT to continue or not to improve the model (first element is for forward and the second one is for the backward method)
- threshold$correlation: threshold used by cossac to choose what combinations (parameter-covariate) must be tried as next candidate model (first element is for forward and the second one is for the backward method)
- useLin: (*boolean*) computes linearization ([TRUE]) or the Importance Sampling (FALSE)
- useSAMBABeforeCossac: (*boolean*) gives the possibility to launch one SAMBA iteration before the COSSAC strategy (TRUE, [FALSE])

See Also

runModelBuilding

Examples

```
## Not run:
set = getModelBuildingSettings()
set$relationships[1,] = c("ka", "SEX", TRUE)
set$relationships[2,] = c("V", "WEIGHT", FALSE)

-> set$relationships
  parameters covariates locked
1         ka        SEX   TRUE
2          V     WEIGHT  FALSE

runModelBuilding(settings = set)

## End(Not run)
```

getNCAIndividualParameters *[PKanalix] Get NCA individual parameters*

Description

Get the estimated values for each subject of some of the individual NCA parameters of the current project.

Usage

```
getNCAIndividualParameters(...)
```

Arguments

`...`	(*string*) Name of the individual parameters whose values must be displayed.

Value

A data frame giving the estimated values of the individual parameters of interest for each subject, and a list of their associated statistics.

Examples

```
## Not run:
indivParams = getNCAIndividualParameters()
# retrieve the values of all the available parameters.

indivParams = getNCAIndividualParameters("Tmax","Clast")
# retrieve only the values of Tmax and Clast for all individuals.

  $parameters->
      id  Tmax Clast
      1   0.8  1.2
      .   ...  ...
      N   0.4  2.2


## End(Not run)
```

getNCASettings *[PKanalix] Get the settings associated to the non compartmental analysis*

Description

Get the settings associated to the non compartmental analysis. Associated settings are:

"administrationType"	(*list*)	list(key = "admId", value = *string*("intravenous" or "extravascular")). *admId*
"integralMethod"	(*string*)	Method for AUC and AUMC calculation and interpolation.
"partialAucTime"	(*list*)	The first element of the list is a bolean describing if this setting is used. The
"blqMethodBeforeTmax"	(*string*)	Method by which the BLQ data before Tmax should be replaced.
"blqMethodAfterTmax"	(*string*)	Method by which the BLQ data after Tmax should be replaced.
"ajdr2AcceptanceCriteria"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"extrapAucAcceptanceCriteria"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"spanAcceptanceCriteria"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"lambdaRule"	(*string*)	Main rule for the lambda_Z estimation.
"timeInterval"	(*vector*)	Time interval for the lambda_Z estimation when "*lambdaRule*" = "interval"
"timeValuesPerId"	(*list*)	list("idName" = idTimes,...): *idTimes* Observation times to use for the calcu
"nbPoints"	(*integer*)	Number of points for the lambda_Z estimation when "*lambdaRule*" = "poin
"maxNbOfPoints"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"startTimeNotBefore"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"weightingNCA"	(*string*)	Weighting method used for the regression that estimates lambda_Z.

Usage

```
getNCASettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

setNCASettings

Examples

```
## Not run:
getNCASettings() # retrieve a list of all the NCA methodology settings
getNCASettings("lambdaRule","integralMethod") # retrieve a list containing only the value of the settings whose

## End(Not run)
```

`getObservationInformation`

[Monolix] Get observations information

Description

Get the name, the type and the values of the observations present in the project.

Usage

```
getObservationInformation()
```

Value

A list containing the name of the observations and their values (id, time and observationName (and occasion if present in the data set))./ In Monolix mode, the observation type and the mapping with data set names is also retrieved.

Examples

```
## Not run:
info = getObservationInformation()
info
  -> $name
     c("concentration")
  -> $type # [ Monolix ]
     c(concentration = "continuous")
  -> $mapping # [ Monolix ]
     c(concentration = "CONC")
  -> $concentration
       id   time concentration
        1    0.5     0.0
        .    .       .
        N    9.0    10.8

## End(Not run)
```

`getPopulationParameterEstimationSettings` *[Monolix] Get population parameter estimation settings*

Description

Get the population parameter estimation settings. Associated settings are:

"nbBurningIterations"	(*int >=0*)	Number of iterations in the burn-in phase.
"nbExploratoryIterations"	(*int >=0*)	If "exploratoryAutoStop" is set to FALSE, it is the number of iteration
"exploratoryAutoStop"	(*bool*)	Should the exploratory step automatically stop.
"exploratoryInterval"	(*int >0*)	Minimum number of interation in the exploratory phase. *Used only if*
"exploratoryAlpha"	(*0<= double <=1*)	Convergence memory in the exploratory phase. *Used only if "explorat*
"nbSmoothingIterations"	(*int >=0*)	If "smoothingAutoStop" is set to FALSE, it is the number of iterations
"smoothingAutoStop"	(*bool*)	Should the smoothing step automatically stop.
"smoothingInterval"	(*int >0*)	inimum number of interation in the smoothing phase. *Used only if "sm*
"smoothingAlpha"	(*0.5< double <=1*)	Convergence memory in the smoothing phase. *Used only if "smoothin*
"smoothingRatio"	(*0< double <1*)	Width of the confidence interval. *Used only if "smoothingAutoStop" is*
"simulatedAnnealing"	(*bool*)	Should annealing be simulated.
"tauOmega"	(*double >0*)	Proportional rate on variance. *Used only if "simulatedAnnealing" is T*
"tauErrorModel"	(*double >0*)	Proportional rate on error model. *Used only if "simulatedAnnealing" i*
"variability"	(*string*)	Estimation method for parameters without variability: "firstStage" | "d
"nbOptimizationIterations"	(*int >=1*)	Number of optimization iterations.
"optimizationTolerance"	(*double >0*)	Tolerance for optimization.

Usage

```
getPopulationParameterEstimationSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

`setPopulationParameterEstimationSettings`

Examples

```
## Not run:
getPopulationParameterEstimationSettings()
# retrieve a list of all the population parameter estimation settings

getPopulationParameterEstimationSettings("nbBurningIterations","smoothingInterval")
# retrieve only the nbBurningIterations and smoothingInterval settings values

## End(Not run)
```

getPopulationParameterInformation *[Monolix] Get population parameters information*

Description

Get the name, the initial value, the estimation method and, if relevant, MAP parameters value of the population parameters present in the project. It is available for fixed effects, random effects, error model parameters, and latent covariates probabilities.

Usage

```
getPopulationParameterInformation()
```

Value

A data frame giving, for each population parameter, the corresponding :

- initialValue : (*double*) initial value
- method : (*string*) estimation method
- priorValue : (*double*) [MAP] typical value
- priorSD : (*double*) [MAP] standard deviation

See Also

setPopulationParameterInformation

Examples

```
## Not run:
info = getPopulationParameterInformation()
info
    name      initialValue   method   typicalValue  stdDeviation
  ka_pop           1.0      MLE             NA            NA
  V_pop           10.0      MAP           10.0           0.5
  omega_ka         1.0     FIXED            NA            NA

## End(Not run)
```

getPreferences *[Monolix - PKanalix] Get project preferences*

Description

Get a summary of the project preferences. Preferences are:

"relativePath"	(*bool*)	Use relative
"threads"	(*int >0*)	Number of t
"timeStamping"	(*bool*) Create an archive containing result files after each run.	
"dpi"	(*bool*) Apply high density pixel correction.	
"imageFormat"	(*string*) Image format used to save monolix graphics.	
"delimiter"	(*string*) Character use as delimiter in exported result files.	
"exportCharts"	(*bool*) Should graphics images be exported.	
"exportChartsData"	(*bool*) Should graphics data be exported.	
"headerAliases"	(*list("header" = vector<string>)*) For each header, the list of the recognized aliases.	

Usage

```
getPreferences(...)
```

Arguments

...	[optional] (string) Name of the preference whose value should be displayed. If no argument is provided, all the preferences are returned.

Value

An array which associates each preference name to its current value.

See Also

setGeneralSettings

Examples

```
## Not run:
getPreferences() # retrieve a list of all the general settings

getPreferences("imageFormat","exportCharts")
# retrieve only the imageFormat and exportCharts settings values

## End(Not run)
```

---

getProjectInformation *[Simulx] Get project information from Mlxtran file*

Description

[Simulx] Get project information from Mlxtran file

Usage

```
getProjectInformation(projectFile)
```

Arguments

projectFile	(string) Path to an Mlxtran project file

Value

A list containing:

- resultFolder
- dataFile
- headerTypes
- headerNames
- dataDelimiter
- observationNames
- observationMapping
- observationTypes (if there are several yTypes defined in the dataset)
- individualParameters

---

getProjectSettings *[Monolix - PKanalix] Get project settings*

Description

Get a summary of the project settings. Associated settings are:

"directory"	(*string*)	Path to the folder where s
"exportResults"	(*bool*)	Should results be exporte
"seed"	(*0< int <2147483647*) Seed used by random generators.	
"grid"	(*int*) Number of points for the continuous simulation grid.	
"nbSimulations"	(*int*) Simulation number.	
"dataAndModelNextToProject"	(*bool*) Should data and model files be saved next to project.	

Usage

```
getProjectSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

getProjectSettings

Examples

```
## Not run:
getProjectSettings() # retrieve a list of all the project settings

getProjectSettings("directory","seed")
# retrieve only the directopry and the seed settings values

## End(Not run)
```

---

getSAEMiterations *[Monolix] Get SAEM algorithm iterations*

Description

Retrieve the successive values of some of the population parameters present within the current project (fixed effects + individual variances + correlations + latent probabilities + error model parameters) during the previous run of the SAEM algorithm.
WARNING: Convergence history of population parameters values cannot be accessible until the SAEM algorithm has been launched once.

Usage

```
getSAEMiterations(...)
```

Arguments

`...`	[optional] (*array<string>*) Names of the population parameters whose convergence history must be displayed. Call `getPopulationParameterInformation` to get a list of the population parameters present within the current project. If this field is not specified, the function will retrieve the values of all the available population parameters.

Value

A list containing a pair composed by the number of exploratory and smoothing iterations and a data frame which associates each wanted population parameter to its successive values over SAEM algorithm iterations.

Examples

```
## Not run:
report = getSAEMiterations()
report
  -> $iterationNumbers
       c(50,25)
     $estimates
          V    Cl
       0.25     0
        0.3   0.5
          .     .
       0.35  0.25

## End(Not run)
```

getScenario *[Monolix] Get current scenario*

Description

Get the list of tasks that will be run at the next call to runScenario, the associated method (linearization true or false), and the associated list of plots. The list of tasks consist of the following tasks: populationParameterEstimation, conditionalDistributionSampling, conditionalModeEstimation, standardErrorEstimation, logLikelihoodEstimation, and plots.

Usage

```
getScenario()
```

Value

The list of tasks that corresponds to the current scenario, indexed by algorithm names.

See Also

setScenario

Examples

```
## Not run:
scenario = getScenario()
scenario
 -> $tasks
   populationParameterEstimation conditionalDistributionSampling conditionalModeEstimation standardErrorEsti
   TRUE                  TRUE  TRUE               FALSE             FALSE             FALSE
    $linearization = T
    $plotList = "outputplot", "vpc"

## End(Not run)
```

getSimulatedIndividualParameters *Get simulated individual parameters*

Description

[Monolix] Get the simulated values for each replicate of each subject of some of the individual parameters present within the current project.
WARNING: Simulated individual parameters values cannot be accessible until the individual estimation with conditional mean algorithm has been launched once.

Usage

```
getSimulatedIndividualParameters(...)
```

Arguments

...	(*string*) Name of the individual parameters whose values must be displayed. Call getIndividualParameterModel to get a list of the individual parameters present within the current project.

Value

A list giving the last simulated values of the individual parameters of interest for each replicate of each subject.

See Also

getSimulatedRandomEffects

Examples

```
## Not run:
simParams = getSimulatedIndividualParameters()
# retrieve the values of all the available individual parameters

simParams
     rep   id    Cl     V     ka
      1    1   0.022  0.37  1.79
      1    2   0.033  0.42  -0.92
      .    .    ...    ...  ...
      2    1   0.021  0.33  1.47
      .    .    ...    ...  ...

## End(Not run)
```

---

getSimulatedRandomEffects

[Monolix] Get simulated random effects

Description

Get the simulated values for each replicate of each subject of some of the individual random effects present within the current project.
WARNING: Simulated individual random effects values cannot be accessible until the individual estimation algorithm with conditional mean has been launched once.

Usage

```
getSimulatedRandomEffects(...)
```

Arguments

...	(*string*) Name of the individual parameters whose values must be displayed. Call getIndividualParameterModel to get a list of the individual parameters present within the current project.

Value

A list giving the last simulated values of the individual random effects of interest for each replicate of each subject.

See Also

getIndividualParameterModel

Examples

```
## Not run:
simEtas = getSimulatedRandomEffects()
# retrieve the values of all the available individual random effects

simEtas
     rep   id    Cl     V     ka
      1    1   0.022  0.37  1.79
      1    2   0.033  0.42  -0.92
      .    .    ...    ...  ...
      2    1   0.021  0.33  1.47
      .    .    ...    ...  ...

## End(Not run)
```

---

getStandardErrorEstimationSettings

[Monolix] Get standard error estimation settings

Description

Get the standard error estimation settings. Associated settings are:

"minIterations"	(*int >=1*)	Minimum number of iterations.
"maxIterations"	(*int >=1*)	Maximum number of iterations.

Usage

```
getStandardErrorEstimationSettings(...)
```

Arguments

...	[optional] (string) Name of the settings whose value should be displayed. If no argument is provided, all the settings are returned.

Value

An array which associates each setting name to its current value.

See Also

setStandardErrorEstimationSettings

Examples

```
## Not run:
getStandardErrorEstimationSettings()
# retrieve a list of all the standard error estimation settings

getStandardErrorEstimationSettings("minIterations","maxIterations")
# retrieve only minIterations and maxIterations settings values

## End(Not run)
```

---

getStructuralModel *[Monolix - PKanalix]Get structural model file*

Description

Get the model file for the structural model used in the current project.

Usage

```
getStructuralModel()
```

Value

A string corresponding to the path to the structural model file.

See Also

setStructuralModel

Examples

```
## Not run:
getStructuralModel() => "/path/to/model/inclusion/modelFile.txt"

## End(Not run)
```

---

getVariabilityLevels *[Monolix] Get variability levels*

Description

Get a summary of the variability levels (inter-individual and/or intra-individual variability) present in the current project.

Usage

```
getVariabilityLevels()
```

Value

A collection of the variability levels present in the currently loaded project.

Examples

```
## Not run:
getVariabilityLevels()

## End(Not run)
```

`initializeLixoftConnectors` *Initialize lixoftConnectors API*

Description

Initialize lixoftConnectors API for a given software

Usage

```
initializeLixoftConnectors(software = "monolix", path = "",
  force = FALSE)
```

Arguments

`software`	(*character*) [optional] Name of the software to be loaded. By default, "monolix" software is used.
`path`	(*character*) [optional] Path to installation directory of the Lixoft suite. If lixoftConnectors library is not already loaded and no path is given, the directory written in the lixoft.ini file is used for initialization.
`force`	(*bool*) [optional] Should software switch security be overpassed or not. Equals FALSE by default.

Value

A boolean equaling TRUE if the initialization has been successful and FALSE if not.

Examples

```
## Not run:
initializeLixoftConnectors(software = "monolix", path = "/path/to/lixoftRuntime/")

## End(Not run)
```

isRunning *[Monolix - PKanalix] Get current scenario state*

Description

Check if a scenario is currently running. If yes, information about the current running task are displayed.

Usage

```
isRunning(verbose = FALSE)
```

Arguments

verbose	(*bool*) Should information about the current running task be displayed in the console or not. Equals FALSE by default.

Value

A boolean which equals TRUE if a scenario is currently running.

See Also

runScenario abort

Examples

```
## Not run:
isRunning()

## End(Not run)
```

lixoftDisplay *Display Lixoft API Structures*

Description

[**Tools**][*Display*]
Display the structures retrieved by the LixoftConnectors library in a user-friendly way.

Usage

```
lixoftDisplay(structure)
```

Arguments

structure	[*miscellanous*] The data structure to be displayed.

Examples

```
## Not run:
settings = getProjectSettings()
lixoftDisplay(settings)

## End(Not run)
```

`loadProject` *[Monolix - PKanalix] Load project from file*

Description

Load a project by parsing the mlxtran-formated file whose path has been given as an input. WARNING: R is sensitive between '\' and '/', only '/' can be used

Usage

```
loadProject(projectFile)
```

Arguments

`projectFile`	(*character*) Path to the project file. Can be absolute or relative to the current working directory.

See Also

`saveProject`

Examples

```
## Not run:
loadProject("/path/to/project/file.mlxtran") for Linux platform
loadProject("C:/Users/path/to/project/file.mlxtran") for Windows platform

## End(Not run)
```

`newProject` *[Monolix - PKanalix] Create new project*

Description

Create a new empty project providing model and data specification. The data specification is:

Monolix, PKanalix

- dataFile (*string*): path to the data file
- headerTypes (*array<character>*): vector of headers
- observationTypes [optional] (*list*): a list, indexed by observation name, giving the type of each observation present in the data file. If omitted, all the observations will be considered as "continuous"
- nbSSDoses (*int*): number of steady-state doses (if there is a SS column)

- mapping [optional](*list*): a list giving the observation name associated to each y-type present in the data file (this field is mandatory when there is a column tagged with the "obsid" headerType)

Please refer to setData documentation for a comprehensive description of the "data" argument structure.

Monolix only data
- projectFile (*string*): path to the datxplore or pkanalix project file defining the

Usage

```
newProject(modelFile = NULL, data)
```

Arguments

modelFile	(*character*) Path to the model file. Can be absolute or relative to the current working directory.
data	(*list*) Structure describing the data.

See Also

newProject saveProject

Examples

```
## Not run:
newProject(data = list(dataFile = "/path/to/data/file.txt",
                       headerTypes = c("IGNORE", "OBSERVATION"),
                       observationTypes = list(concentration = "continuous")),
           modelFile = "/path/to/model/file.txt")

newProject(data = list(dataFile = "/path/to/data/file.txt",
                       headerTypes = c("IGNORE", "OBSERVATION", "OBSID"),
                observationTypes = list(concentration = "continuous", effect = "discrete"),
                       mapping = list("1" = "concentration", "2" = "effect")),
           modelFile = "/path/to/model/file.txt")



[Monolix only]

  newProject(data = list(projectFile = "/path/to/project/file.datxplore"),
             modelFile = "/path/to/model/file.txt")

## End(Not run)
```

---

removeCovariate *[Monolix] Remove covariate*

Description

Remove some of the transformed covariates (discrete and continuous) and/or latent covariates. Call getCovariateInformation to know which covariates can be removed.

Usage

```
removeCovariate(...)
```

Arguments

...	A list of covariate names.

See Also

getCovariateInformation addContinuousTransformedCovariate addCategoricalTransformedCovariate addMixture

Examples

```
## Not run:
removeCovariate("tWt","lcat1")

## End(Not run)
```

---

runCAEstimation *[PKanalix] Estimate the individual parameters using compartmental analysis.*

Description

Estimate the CA parameters for each individual of the project.

Usage

```
runCAEstimation(wait = TRUE)
```

Arguments

wait	(*logical*) Should R wait for run completion before giving back the hand to the user. *TRUE* by default.

Examples

```
## Not run:
runCAEstimation()

## End(Not run)
```

runConditionalDistributionSampling

[Monolix] Sampling from the conditional distribution

Description

Estimate the individual parameters using conditional distribution sampling algorithm. The associated method keyword is "conditionalMean". By default, this task is not processed in the background of the R session. Notice that it does not impact the current scenario. Call

1. `isRunning` to check if the scenario is still running and get information about the current task,
2. `abort` to stop the execution.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE.

Usage

```
runConditionalDistributionSampling(wait = TRUE)
```

Arguments

`wait`	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.

See Also

`isRunning` `abort`

Examples

```
## Not run:
runConditionalDistributionSampling()

## End(Not run)
```

runConditionalModeEstimation

[Monolix] Estimation of the conditional modes (EBEs)

Description

Estimate the individual parameters using the conditional mode estimation algorithm (EBEs). The associated method keyword is "conditionalMode". By default, this task is not processed in the background of the R session. Notice that it does not impact the current scenario. Call

1. `isRunning` to check if the scenario is still running and get information about the current task,
2. `abort` to stop the execution.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE.

Usage

```
runConditionalModeEstimation(wait = TRUE)
```

Arguments

wait	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.

See Also

isRunning abort

Examples

```
## Not run:
runConditionalModeEstimation()

## End(Not run)
```

---

runEstimation *[PKanalix] Run both non compartmental and compartmental analysis.*

Description

Run the NCA analysis and the CA analysis if the structural model for the CA calculation is defined.

Usage

```
runEstimation(wait = TRUE)
```

Arguments

wait	(*logical*) Should R wait for run completion before giving back the hand to the user. *TRUE* by default.

Examples

```
## Not run:
runEstimation()

## End(Not run)
```

`runLogLikelihoodEstimation` *[Monolix] Log-Likelihood estimation*

Description

Run the log-Likelihood estimation algorithm. By default, this task is not processed in the background of the R session. Notice that it does not impact the current scenario. Call

1. `isRunning` to check if the scenario is still running and get information about the current task,
2. `abort` to stop the execution.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE.
Existing methods:

Method	*Identifier*
Log-Likelihood estimation by linearization	linearization = T
Log-Likelihood estimation by Importance Sampling (default)	linearization = F

The Log-likelihood outputs(-2LL, AIC, BIC) are available using `getEstimatedLogLikelihood` function

Usage

```
runLogLikelihoodEstimation(linearization = FALSE, wait = TRUE)
```

Arguments

`linearization`	option (*boolean*)[optional] method to be used. When no method is given, the importance sampling is used by default.
`wait`	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.

See Also

`isRunning` `abort`

Examples

```
## Not run:
runLogLikelihoodEstimation(linearization = T)

## End(Not run)
```

runModelBuilding *[Monolix] Run model building*

Description

Run model building. Call

1. isRunning to check if the building is still running and get information about the current task,
2. stopModelBuilding to stop the execution.

Usage

```
runModelBuilding(wait = TRUE, ...)
```

Arguments

wait	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.
...	(*list<settings>*) Settings to initialize the model buildign algorithm. See getModelBuildingSettings

Details

To change the initialization before a run, use getModelBuildingSettings to receive all the settings. See example.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE.

See Also

getModelBuildingSettings getModelBuildingResults stopModelBuilding isRunning

Examples

```
## Not run:
runModelBuilding() # sequential run
set = getModelBuildingSettings()
runModelBuilding(settings = set) # sequential run
runModelBuilding(wait = TRUE) # background run

## End(Not run)
```

runNCAEstimation *[PKanalix] Estimate the individual parameters using non compartmental analysis.*

Description

Estimate the NCA parameters for each individual of the project.

Usage

```
runNCAEstimation(wait = TRUE)
```

Arguments

wait	(*logical*) Should R wait for run completion before giving back the hand to the user. *TRUE* by default.

Examples

```
## Not run:
runNCAEstimation()

## End(Not run)
```

runPopulationParameterEstimation *[Monolix] Population parameter estimation*

Description

Estimate the population parameters with the SAEM method. The associated method keyword is "saem". By default, this task is not processed in the background of the R session. Notice that it does not impact the current scenario. Call

1. `isRunning` to check if the scenario is still running and get information about the current task,
2. `abort` to stop the execution.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE. The initial values of the population parameters can be accessed by calling `getPopulationParameterInformation` and customized with `setPopulationParameterInformation`.
The estimated population parameters are available using `getEstimatedPopulationParameters` function.

Usage

```
runPopulationParameterEstimation(wait = TRUE)
```

Arguments

wait	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.

See Also

isRunning abort

Examples

```
## Not run:
runPopulationParameterEstimation()

## End(Not run)
```

---

runScenario *[Monolix] Run current scenario*

Description

Run the current scenario. By default, this task is processed sequentially. Call

1. isRunning to check if the scenario is still running and get information about the current task,
2. abort to stop the execution.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE.

Usage

```
runScenario(wait = TRUE)
```

Arguments

wait	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.

See Also

setScenario getScenario abort isRunning

Examples

```
## Not run:
runScenario() # sequential run
runScenario(wait = TRUE) # background run

## End(Not run)
```

`runStandardErrorEstimation` *[Monolix] Standard error estimation*

Description

Estimate the Fisher Information Matrix and the standard errors of the population parameters. By default, this task is not processed in the background of the R session. Notice that it does not impact the current scenario. Call

1. `isRunning` to check if the scenario is still running and get information about the current task,
2. `abort` to stop the execution.

To launch the function in the background, so that functions which do not modify the project ("get" functions for example) remains available, set the input argument "wait" to FALSE.

Usage

```
runStandardErrorEstimation(linearization = FALSE, wait = TRUE)
```

Arguments

`linearization`	option (*boolean*)[optional] method to be used. When no method is given, the stochastic approximation is used by default.
`wait`	(*bool*) Should R wait for run completion before giving back the hand to the user. Equals TRUE by default.

Details

Existing methods:

Method	*Identifier*
Estimate the FIM by Stochastic Approximation	linearization = F (default)
Estimate the FIM by Linearization	linearization = T

The Fisher Information Matrix is available using `getCorrelationOfEstimates` function, while the standard errors are avalaible using `getEstimatedStandardErrors` function.

See Also

`isRunning` `abort`

Examples

```
## Not run:
runStandardErrorEstimation(linearization = T)

## End(Not run)
```

saveProject *[Monolix - PKanalix] Save current project*

Description

Save the current project as an Mlxtran-formated file.

Usage

```
saveProject(projectFile = "")
```

Arguments

projectFile	[optional](*character*) Path where to save a copy of the current mlxtran model. Can be absolute or relative to the current working directory. If no path is given, the file used to build the current configuration is updated.

See Also

newProject loadProject

Examples

```
## Not run:
saveProject("/path/to/project/file.mlxtran") # save a copy of the model
saveProject() # update current model

## End(Not run)
```

setAutocorrelation *[Monolix] Set auto-correlation*

Description

Add or remove auto-correlation from the error model used on some of the observation models.
Call getObservationInformation to get a list of the observation models present in the current project.

Usage

```
setAutocorrelation(...)
```

Arguments

...	Sequence of comma-separated pairs {(*string*)"observationModel",(*boolean*)hasAutoCorrelation}.

See Also

getContinuousObservationModel

Examples

```
## Not run:
setAutocorrelation(Conc = TRUE)
setAutocorrelation(Conc = TRUE, Effect = FALSE)

## End(Not run)
```

`setCASettings` *[PKanalix] Get the settings associated to the compartmental analysis*

Description

Get the settings associated to the compartmental analysis. Associated settings names are:

"weightingCA"	(*string*)	Type of weighting ob
"pool"	(*logical*)	If TRUE, fit with indi
"initialValues" (*list*)	list(param = value, ...): value = initial value of individual parameter param.	
"blqMethod"	(*string*)	Method by which the

Usage

```
setCASettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getCASettings

Examples

```
## Not run:
setCASettings(weightingCA = "uniform", blqMethod = "zero") # set the settings whose name has been passed in argu
setCASettings(initialValues = list(Cl=0.4, V=.5, ka=0.04) # set the paramters CL, V, and ka to .4, .5 and .04 res

## End(Not run)
```

setConditionalDistributionSamplingSettings

[Monolix] Set conditional distribution sampling settings

Description

Set the value of one or several of the conditional distribution sampling settings. Associated settings are:

"ratio"	(*0< double <1*)	Width of the confidence interval.
"nbMinIterations"	(*int >=1*)	Minimum number of iterations.
"nbSimulatedParameters"	(*int >=1*)	Number of replicates.

Usage

```
setConditionalDistributionSamplingSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getConditionalDistributionSamplingSettings

Examples

```
## Not run:
setConditionalDistributionSamplingSettings(ratio = 0.05, nbMinIterations = 50)

## End(Not run)
```

setConditionalModeEstimationSettings

[Monolix] Set conditional mode estimation settings

Description

Set the value of one or several of the conditional mode estimation settings. Associated settings are:

"nbOptimizationIterationsMode"	(*int >=1*)	Maximum number of iterations.
"optimizationToleranceMode"	(*double >0*)	Optimization tolerance.

Usage

```
setConditionalModeEstimationSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getConditionalModeEstimationSettings

Examples

```
## Not run:
setConditionalModeEstimationSettings(nbOptimizationIterationsMode = 20,
                                     optimizationToleranceMode = 0.1)

## End(Not run)
```

---

setCorrelationBlocks *[Monolix] Set correlation block structure*

Description

Define the correlation block structure associated to some of the variability levels of the current project. Call getVariabilityLevels to get a list of the variability levels and getIndividualParameterModel to get a list of the available individual parameters within the current project.

Usage

```
setCorrelationBlocks(...)
```

Arguments

...	A list of comma-separated pairs {variabilityLevel = vector< (*array*<*string*>)parameterNames} > }.

See Also

getVariabilityLevels getIndividualParameterModel

Examples

```
## Not run:
setCorrelationBlocks(id = list( c("ka","V","Tlag") ), iov1 = list( c("ka","Cl"), c("Tlag","V") ) )

## End(Not run)
```

setCovariateModel *[Monolix] Set covariate model*

Description

Set which are the covariates influencing individual parameters present in the project. Call getIndividualParameterMod
to get a list of the individual parameters present within the current project. and getCovariateInformation
to know which are the available covariates for a given level of variability and a given individual parameter.

Usage

```
setCovariateModel(...)
```

Arguments

...	A list of comma-separated pairs {parameterName = { covariateName = (*bool*)isInfluent, ...} }

See Also

getCovariateInformation

Examples

```
## Not run:
setCovariateModel( ka = c( Wt = FALSE, tWt = TRUE, lcat2 = TRUE),
                   Cl = c( SEX = TRUE )
                   )

## End(Not run)
```

setData *Set project data*

Description

Set project data giving a data file and specifying headers and observations types.

Usage

```
setData(dataFile, headerTypes, observationTypes, nbSSDoses = NULL)
```

Arguments

dataFile	(*character*): Path to the data file. Can be absolute or relative to the current working directory.
headerTypes	(*array<character>*): A collection of header types. The possible header types are: "ignore", "id", "time", "observation", "amount", "contcov", "catcov", "occ", "evid", "mdv", "obsid", "cens", "limit", "regressor","admid", "rate", "tinf", "ss", "ii", "addl", "date" Notice that these are not the types displayed in the interface, these one are shortcuts.
observationTypes	[optional] (*list*): A list giving the type of each observation present in the data file. If there is only one y-type, the corresponding observation name can be omitted. The possible observation types are "continuous", "discrete", and "event".
nbSSDoses	[optional] (*int*): Number of doses (if there is a SS column).

See Also

getData

Examples

```
## Not run:
setData(dataFile = "/path/to/data/file.txt",
        headerTypes = c("IGNORE", "OBSERVATION"), observationTypes = "continuous")
setData(dataFile = "/path/to/data/file.txt",
        headerTypes = c("IGNORE", "OBSERVATION", "YTYPE"),
        observationTypes = list(Concentration = "continuous", Level = "discrete"))

## End(Not run)
```

---

setDataSettings *[PKanalix] Set the value of one or several of the data settings associated to the non compartmental analysis*

Description

Set the value of one or several of the data settings associated to the non compartmental analysis. Associated settings names are:

"urinevolume"	(*string*)	regressor name used as urine volume.
"datatype"	(*list*)	list("obsId" = *string*("plasma" or "urine"). The type of data associated with each *obsId*. Defaul

Usage

```
setDataSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getDataSettings

Examples

```
## Not run:
setDataSettings("datatype" = list("Y" ="plasma")) # set the settings whose name has been passed in argument

## End(Not run)
```

---

setErrorModel *[Monolix] Set error model*

Description

Set the error model type to be used with some of the observation models. Call getObservationInformation to get a list of the observation models present in the current project.

Usage

```
setErrorModel(...)
```

Arguments

...	A list of comma-separated pairs {observationModel = (*string*)errorModelType}.

Details

Available error model types are :

"constant"	obs = pred + a*err
"proportional"	obs = pred + (b*pred)*err
"combined1"	obs = pred + (b*pred^c + a)*err
"combined2"	obs = pred + sqrt(a^2 + (b^2)*pred^(2c))*err

Error model parameters will be initialized to 1 by default. Call setPopulationParameterInformation to modify their initial value.
The value of the exponent parameter is fixed by default when using the "combined1" and "combined2" models.
Use setPopulationParameterInformation to enable its estimation.

See Also

getContinuousObservationModel setPopulationParameterInformation

Examples

```
## Not run:
setErrorModel(Conc = "constant", Effect = "combined1")

## End(Not run)
```

setGeneralSettings *[Monolix] Set common settings for algorithms*

Description

Set the value of one or several of the common settings for Monolix algorithms. Associated settings are:

"autoChains"	(*bool*)	Automatically adjusted the number of chains to have at least a minimum number of sul
"nbChains"	(*int >0*)	Number of chains to be used if "autoChains" is set to FALSE.
"minIndivForChains"	(*int >0*)	Minimum number of individuals by chain.

Usage

```
setGeneralSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getGeneralSettings

Examples

```
## Not run:
setGeneralSettings(autoChains = FALSE, nbchains = 10)

## End(Not run)
```

setGlobalObsIdToUse *[PKanalix] Set the global observation id used in both the compartmental and non compartmental analysis*

Description

Get the global observation id used in both the compartmental and non compartmental analysis.

Usage

```
setGlobalObsIdToUse(...)
```

Arguments

...	("id" *string*) the observation id from data section to use for computations.

See Also

getGlobalObsIdToUse

Examples

```
## Not run:
setGlobalObsIdToUse("id") #

## End(Not run)
```

`setIndividualLogitLimits` *[Monolix] Set individual parameter distribution limits*

Description

Set the minimum and the maximum values between the individual parameter can be used. Used only if the distribution of the parameter is "logitNormal", else wise it will not be taken into account

Usage

```
setIndividualLogitLimits(...)
```

Arguments

...	A list of comma-separated pairs {individualParameter = [(double)min,(double)max] }

See Also

getIndividualParameterModel

Examples

```
## Not run:
setIndividualLogitLimits( V = c(0, 1), ka = c(-1, 2) )

## End(Not run)
```

setIndividualParameterDistribution

[Monolix] Set individual parameter distribution

Description

Set the distribution of the estimated parameters. Available distributions are "normal", "logNormal" and "logitNormal".
Call `getIndividualParameterModel` to get a list of the available individual parameters within the current project.

Usage

```
setIndividualParameterDistribution(...)
```

Arguments

...	A list of comma-separated pairs {parameterName = (*string*)"distribution"}.

See Also

`getIndividualParameterModel`

Examples

```
## Not run:
setIndividualParameterDistribution(V = "logNormal")
setIndividualParameterDistribution(Cl = "normal", V = "logNormal")

## End(Not run)
```

setIndividualParameterVariability

[Monolix] Individual variability management

Description

Add or remove inter-individual and/or intra-individual variability from some of the individual parameters present in the project.
Call `getIndividualParameterModel` to get a list of the available parameters within the current project.

Usage

```
setIndividualParameterVariability(...)
```

Arguments

...	A list of comma-separated pairs {variabilityLevel = {individualParameterName = (*bool*)hasVariability} }.

See Also

getIndividualParameterModel

Examples

```
## Not run:
setIndividualParameterVariability(ka = TRUE, V = FALSE)
setIndividualParameterVariability(id = list(ka = TRUE), iov1 = list(ka = FALSE))

## End(Not run)
```

`setInitialEstimatesToLastEstimates` *[Monolix] Initialize population parameters with the last estimated ones*

Description

Set the initial value of all the population parameters present within the current project to the ones previously estimated. These the values will be used in the population parameter estimation algorithm during the next scenario run.
WARNING: If there is any set after a run, it will not be possible to set the initial values as the structure of the project has changed since last results.

Usage

```
setInitialEstimatesToLastEstimates(fixedEffectsOnly = FALSE)
```

Arguments

`fixedEffectsOnly`
: (*bool*) If this boolean is set to TRUE, only the fixed effects are initialized to their last estimated values. Otherwise, individual variances and error model parameters are re-initialized too. Equals FALSE by default.

See Also

getEstimatedPopulationParameters getPopulationParameterInformation

Examples

```
## Not run:
setInitialEstimatesToLastEstimates() # fixedEffectsOnly = FALSE by default
setInitialEstimatesToLastEstimates(TRUE)

## End(Not run)
```

setLogLikelihoodEstimationSettings *[Monolix] Set loglikelihood estimation settings*

Description

Set the value of the loglikelihood estimation settings. Associated settings are:

"nbFixedIterations"	(*int >0*)	Monte Carlo size for the loglikelihood evaluation.
"samplingMethod"	(*string*)	Should the loglikelihood estimation use a given number of freedom d
"nbFreedomDegrees"	(*int >0*)	Degree of freedom of the Student t-distribution. *Used only if "samplin*
"freedomDegreesSampling"	(*vector<int(>0)>*)	Sequence of freedom degrees to be tested. *Used only if "samplingMet*

Usage

```
setLogLikelihoodEstimationSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getLogLikelihoodEstimationSettings

Examples

```
## Not run:
setLogLikelihoodEstimationSettings(nbFixedIterations = 20000)

## End(Not run)
```

setMCMCSettings *[Monolix] Set settings associated to the MCMC algorithm*

Description

Set the value of one or several of the MCMC algorithm specific settings of the current project. Associated settings are:

"strategy"	(*vector<int>[3]*)	Number of calls for each one of the three MCMC kernels.
"acceptanceRatio"	(*double*)	Target acceptance ratio.

Usage

```
setMCMCSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getMCMCSettings

Examples

```
## Not run:
setMCMCSettings(strategy = c(2,1,2))

## End(Not run)
```

`setNCASettings` *[PKanalix] Set the value of one or several of the settings associated to the non compartmental analysis*

Description

Set the value of one or several of the settings associated to the non compartmental analysis. Associated settings are:

"administrationType"	(*list*)	list(key = "admId", value = *string*("intravenous" or "extravascular")). *admId*
"integralMethod"	(*string*)	Method for AUC and AUMC calculation and interpolation.
"partialAucTime"	(*list*)	The first element of the list is a bolean describing if this setting is used. The
"blqMethodBeforeTmax"	(*string*)	Method by which the BLQ data before Tmax should be replaced. Possible r
"blqMethodAfterTmax"	(*string*)	Method by which the BLQ data after Tmax should be replaced. Possible me
"ajdr2AcceptanceCriteria"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"extrapAucAcceptanceCriteria"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"spanAcceptanceCriteria"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"lambdaRule"	(*string*)	Main rule for the lambda_Z estimation. Possible rules are "R2", "interval",
"timeInterval"	(*vector*)	Time interval for the lambda_Z estimation when "*lambdaRule*" = "interval"
"timeValuesPerId"	(*list*)	list("idName" = idTimes,...): *idTimes* Observation times to use for the calcu
"nbPoints"	(*integer*)	Number of points for the lambda_Z estimation when "*lambdaRule*" = "poin
"maxNbOfPoints"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"startTimeNotBefore"	(*list*)	The first element of the list is a boolean describing if this setting is used. Th
"weightingNca"	(*string*)	Weighting method used for the regression that estimates lambda_Z. Possible

Usage

```
setNCASettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getNCASettings

Examples

```
## Not run:
setNCASettings(integralMethod = "LinLogTrapLinLogInterp", weightingnca = "uniform") # set the settings whose n
setNCASettings(administrationType = list("1"="extravascular")) # set the administration id "1" to extravascula
setNCASettings(startTimeNotBefore = list(TRUE, 15)) # set the estimation of the lambda_z with points with time c
setNCASettings(timeValuesPerId = list('1'=c(4, 6, 8, 30), '4'=c(8, 12, 18, 24, 30))) # set the points to use for
setNCASettings(timeValuesPerId = NULL) # set the points to use for the lambda_z to the default rule

## End(Not run)
```

---

setObservationDistribution

[Monolix] Set observation model distribution

Description

Set the distribution in the Gaussian space of some of the observation models. Available distribution types are "normal", "logNormal", or "logitNormal". Call getObservationInformation to get a list of the available observation models within the current project.

Usage

```
setObservationDistribution(...)
```

Arguments

...	A list of comma-separated pairs {observationModel = (*string*)"distribution"}.

See Also

getContinuousObservationModel

Examples

```
## Not run:
setObservationDistribution(Conc = "normal")
setObservationDistribution(Conc = "normal", Effect = "logNormal")

## End(Not run)
```

setObservationLimits *[Monolix] Set observation model distribution limits*

Description

Set the minimum and the maximum values between which some of the observations can be found. Used only if the distribution of the error model is "logitNormal", else wise it will not be taken into account

Usage

```
setObservationLimits(...)
```

Arguments

...	A list of comma-separated pairs {observationModel = [(double)min,(double)max] }

See Also

getContinuousObservationModel getObservationInformation

Examples

```
## Not run:
setObservationLimits( Conc = c(-Inf,Inf), Effect = c(0,Inf) )

## End(Not run)
```

setPopulationParameterEstimationSettings *[Monolix] Set population parameter estimation settings*

Description

Set the value of one or several of the population parameter estimation settings. Associated settings are:

"nbBurningIterations"	(*int >=0*)	Number of iterations in the burn-in phase.
"nbExploratoryIterations"	(*int >=0*)	If "exploratoryAutoStop" is set to FALSE, it is the number of iteration
"exploratoryAutoStop"	(*bool*)	Should the exploratory step automatically stop.
"exploratoryInterval"	(*int >0*)	Minimum number of interation in the exploratory phase. *Used only if*
"exploratoryAlpha"	(*0<= double <=1*)	Convergence memory in the exploratory phase. *Used only if "explorat*
"nbSmoothingIterations"	(*int >=0*)	If "smoothingAutoStop" is set to FALSE, it is the number of iterations
"smoothingAutoStop"	(*bool*)	Should the smoothing step automatically stop.
"smoothingInterval"	(*int >0*)	Minimum number of interation in the smoothing phase. *Used only if "*
"smoothingAlpha"	(*0.5< double <=1*)	Convergence memory in the smoothing phase. *Used only if "smoothin*
"smoothingRatio"	(*0< double <1*)	Width of the confidence interval. *Used only if "smoothingAutoStop" is*
"simulatedAnnealing"	(*bool*)	Should annealing be simulated.
"tauOmega"	(*double >0*)	Proportional rate on variance. *Used only if "simulatedAnnealing" is T*

"tauErrorModel"	(*double >0*)	Proportional rate on error model. *Used only if "simulatedAnnealing" i*
"variability"	(*string*)	Estimation method for parameters without variability: "firstStage" | "c
"nbOptimizationIterations"	(*int >=1*)	Number of optimization iterations.
"optimizationTolerance"	(*double >0*)	Tolerance for optimization.

Usage

```
setPopulationParameterEstimationSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = SettingValue}.

See Also

getPopulationParameterEstimationSettings

Examples

```
## Not run:
setPopulationParameterEstimationSettings(exploratoryAutoStop = TRUE, tauOmega = 0.95)

## End(Not run)
```

---

setPopulationParameterInformation

[Monolix] Population parameters initialization and estimation method

---

Description

Set the initial value, the estimation method and, if relevant, the MAP parameters of one or several of the population parameters present within the current project (fixed effects + individual variances + error model parameters). Available methods are:

- "FIXED": Fixed
- "MLE": Maximum Likelihood Estimation
- "MAP": Maximum A Posteriori

Call getPopulationParameterInformation to get a list of the initializable population parameters present within the current project.

Usage

```
setPopulationParameterInformation(...)
```

Arguments

...	A list of comma-separated pairs {paramName = list(initialValue = (*double*), method = (*string*)"method"}. In case of "MAP" method, the user can specify the associated typical value and standard deviation values by using an additional list elements {paramName = list(priorValue = (*double*)1, priorSD = (*double*)2)}. By default, the prior value corresponds to the the population parameter and the prior standard deviation is set to 1.

See Also

getPopulationParameterInformation

Examples

```
## Not run:
setPopulationParameterInformation(Cl_pop = list(initialValue = 0.5, method = "FIXED"),
                                  V_pop  = list(intialValue = 1),
                               ka_pop = list(method = "MAP", priorValue = 1, priorSD = 0.1))

## End(Not run)
```

`setPreferences` *[Monolix - PKanalix] Set preferences*

Description

Set the value of one or several of the project preferences. Prefenreces are:

"relativePath"	(*bool*)	Use relative
"threads"	(*int >0*)	Number of t
"timeStamping"	(*bool*) Create an archive containing result files after each run.	
"dpi"	(*bool*) Apply high density pixel correction.	
"imageFormat"	(*string*) Image format used to save monolix graphics.	
"delimiter"	(*string*) Character use as delimiter in exported result files.	
"exportCharts"	(*bool*) Should graphics images be exported.	
"exportChartsData"	(*bool*) Should graphics data be exported.	
"headerAliases"	(*list("header" = vector<string>)*) For each header, the list of the recognized aliases.	

Usage

```
setPreferences(...)
```

Arguments

...	A collection of comma-separated pairs {preferenceName = settingValue}.

See Also

getPreferences

Examples

```
## Not run:
setPreferences(exportCharts = FALSE, delimiter = ",")

## End(Not run)
```

setProjectSettings *[Monolix - PKanalix] Set project settings*

Description

Set the value of one or several of the settings of the project. Associated settings are:

"directory"	(*string*)	Path to the folder where s
"exportResults"	(*bool*)	Should results be exporte
"seed"	(*0< int <2147483647*) Seed used by random generators.	
"grid"	(*int*) Number of points for the continuous simulation grid.	
"nbSimulations"	(*int*) Simulation number.	
"dataAndModelNextToProject"	(*bool*) Should data and model files be saved next to project.	

Usage

```
setProjectSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getProjectSettings

Examples

```
## Not run:
setProjectSettings(directory = "/path/to/export/directory", seed = 12345)

## End(Not run)
```

setScenario *[Monolix] Set scenario*

Description

Clear the current scenario and build a new one from a given list of tasks, the linearization option and the list of plots. A task is the association of:

- a task
- a boolean

NOTE: by default the boolean is false, thus, the user can only state what will run during the scenario.

Usage

```
setScenario(...)
```

Arguments

...	A list of tasks as previously defined.

Details

NOTE: Within a MONOLIX scenario, the order according which the different algorithms are run is fixed:

Algorithm	Algorithm Keyword
Population Parameter Estimation	"populationParameterEstimation"
Conditional Mode Estimation (EBEs)	"conditionalModeEstimation"
Sampling from the Conditional Distribution	"conditionalDistributionSampling"
Standard Error and Fisher Information Matrix Estimation	"standardErrorEstimation"
LogLikelihood Estimation	"logLikelihoodEstimation"
Plots	"plots"

See Also

getScenario

Examples

```
## Not run:

scenario = getScenario()
scenario$tasks = c(populationParameterEstimation = T, conditionalModeEstimation = T, conditionalDistributionS
setScenario(scenario)

## End(Not run)
```

---

setStandardErrorEstimationSettings

[Monolix] Set standard error estimation settings

Description

Set the value of one or several of the standard error estimation settings. Associated settings are:

"minIterations"	(*int >=1*)	Minimum number of iterations.
"maxIterations"	(*int >=1*)	Maximum number of iterations.

Usage

```
setStandardErrorEstimationSettings(...)
```

Arguments

...	A collection of comma-separated pairs {settingName = settingValue}.

See Also

getStandardErrorEstimationSettings

Examples

```
## Not run:
setStandardErrorEstimationSettings(minIterations = 20, maxIterations = 250)

## End(Not run)
```

`setStructuralModel` *[Monolix - PKanalix] Set structural model file*

Description

Set the structural model. NOTE: In case of PKanalix, the user can only use a structural model from the library for the CA analysis. Thus, the structura model should be written 'lib:modelFromLibrary.txt'.

Usage

```
setStructuralModel(modelFile)
```

Arguments

`modelFile`	(*character*) Path to the model file. Can be absolute or relative to the current working directory.

See Also

getStructuralModel

Examples

```
## Not run:
setStructuralModel("/path/to/model/file.txt") # for Monolix
setStructuralModel("'lib:oral1_2cpt_kaClV1QV2.txt'") # for PKanalix

## End(Not run)
```

stopModelBuilding *[Monolix] Stop the model building*

Description

Stop the model building.

Usage

```
stopModelBuilding()
```

See Also

runModelBuilding isRunning

Examples

```
## Not run:
stopModelBuilding()

## End(Not run)
```

writeProjectModelSection *Write project model section [Simulx] Write the model section of a given Mlxtran project.*

Description

Write project model section [Simulx] Write the model section of a given Mlxtran project.

Usage

```
writeProjectModelSection(projectFile, model)
```

Arguments

projectFile	(string) Path to an Mlxtran project file
model	(string) Content of the model section to write

Index