Package 'Certara.RDarwin'

May 7, 2026

Title Interface for 'pyDarwin' Machine Learning Pharmacometric Model Development

Version 1.2.0

Description Utilities that support the usage of 'pyDarwin' (<https://certara.github.io/pyDarwin/>) for ease of setup and execution of a machine learning based pharmacometric model search with Certara's Non-Linear Mixed Effects (NLME) modeling engine.

Depends R (>= 3.6)

License LGPL-3

URL https://certara.github.io/R-Darwin/

Encoding UTF-8

RoxygenNote 7.3.2

Imports methods, magrittr, jsonlite, ssh

Suggests testthat (>= 3.0.0), dplyr, tidyr, tibble, knitr, rmarkdown, Certara.DarwinReporter, Certara.RsNLME, geometry

Config/testthat/edition 3

NeedsCompilation no

Author Michael Tomashevskiy [aut], James Craig [aut, cre], Certara USA, Inc [cph, fnd]

Maintainer James Craig <james.craig@certara.com>

Repository CRAN

Date/Publication 2025-12-01 00:30:02 UTC

Contents

`add_Covariate` *Add Covariate into PML models*

Description

Add Covariate into PML models

Usage

```
add_Covariate(
  PMLParametersSets,
  Name,
  Type = "Continuous",
  StParmNames = NULL,
  State = "Present",
  Direction = "Forward",
  Center = "None",
  Categories = c(),
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`Name`	A character string representing the name of the covariate to be added.
`Type`	A character specifying the type of the covariate. Possible values are: • `Continuous` A covariate can take values on a continuous scale. • `Categorical` A covariate can only take a finite number of values. • `Occasion` The associated PK parameter may vary within an individual from one event to the next, called interoccasion variability.
`StParmNames`	Character or character vector specifying names of structural parameters to which covariates should be added. Can be set to `NULL` or not specified, for such case, covariate will be added to all structural parameters.
`State`	A character string representing the presence of the covariate on the structural parameters. Possible values are: • `None` The covariate does not have an effect on any structural parameter. • `Present` The covariate has an effect on the structural parameters (the default). • `Searched` The effect of the covariate on structural parameters is searched.

Direction	A character string representing the direction of the Covariate. Options are Forward, Backward, Interpolate. Default is Forward. Interpolate is only applicable to Type == "Continuous".
Center	A character string (None, Mean or Median) or numeric value representing the center of the Covariate. Default is None. Valid only if Type == "Continuous".
Categories	A numeric vector representing the categories (at least two) of the covariate. Applicable only if Type is either Occasion or Categorical. The first category is set to the reference category for categorical covariate. If a named vector is used, the names are used as labels for the given dataset,
PMLStructures	Character or character vector specifying names of PML structures to which the covariate will be added. For the naming covention of PMLStructures, see Details section of create_ModelPK() for PK models and create_ModelPD() for PD models.

Details

- If Covariate already exists, it will be substituted with a new instance with given properties. New covariate will have default bound omegas/thetas. The user can change thetas with modify_Theta() and omegas with modify_Omega().
- The current functionality does not support adding or modifying custom covariates that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Covariates() modify_Theta() modify_Omega()

Functions used for Covariate specification: Covariate(), create_ModelPD(), create_ModelPK(), remove_Covariate()

Examples

```
PMLParametersSets <- create_ModelPK()

PMLParametersSetsWT <-
 add_Covariate(PMLParametersSets,
               Name = "WT",
               Type = "Continuous",
               State = "Present",
               Direction = "Forward",
               Center = 70)

PMLParametersSetsWTCL <-
 add_Covariate(PMLParametersSets = PMLParametersSetsWT,
               Name = "Race",
               Type = "Categorical",
               State = "Searched",
```

```
                Direction = "Backward",
                Categories = c(1,2,3),
                StParmNames = "Cl",
                PMLStructure = "PK1IVC")
```

`add_CustomSpace` *Add a Custom Space to a PMLModels Object*

Description

Adds a new model space, defined by custom PML code, to an existing collection of model spaces (a `PMLModels` object).

Usage

```
add_CustomSpace(Spaces, CustomCode, SpaceName = character())
```

Arguments

`Spaces`	A `PMLModels` object (a named list) representing the existing collection of model spaces to which the new custom space will be added. This can be an empty list or a previously created `PMLModels` object.
`CustomCode`	A character string containing the complete custom PML code (e.g., the content of a `test(){...}` block, potentially excluding the `test(){` and closing `}` depending on usage context, although including them is safer for parsing). Multi-line strings are collapsed. Cannot be empty.
`SpaceName`	An optional character string specifying the name for this custom model space. This name will be used as the key for this model within the returned `PMLModels` list. If omitted or an empty string (`""`), a unique identifier will be created by concatenating the letter `"l"` with the number of characters in `CustomCode`. Providing a meaningful name is recommended for clarity, especially when working with multiple custom models. Ensure provided names are unique if creating multiple custom spaces intended to coexist.

Details

This function serves as a wrapper around `create_CustomSpace()`. It first calls `create_CustomSpace` using the provided `CustomCode` and `SpaceName` to parse the code and create a representation of the new custom space. The name of this new space is either the provided `SpaceName` or one automatically generated by `create_CustomSpace` if `SpaceName` was omitted or empty (e.g., `"l<number>"` based on code length).

Value

An updated `PMLModels` object (a named list) containing all the original spaces plus the newly added custom space.

See Also

create_CustomSpace(), add_Spaces(), create_ModelPK(), create_ModelPD()

Examples

```
# Start with some built-in models
pk_models <- create_ModelPK(CompartmentsNumber = 1)

# Define custom code
custom_pml <- "test() {
  cfMicro(A1, Cl / V)
  dosepoint(A1)
  C = A1 / V
  error(CEps = 1)
  observe(CObs = C + CEps)
  stparm(V = tvV * exp(nV))
  stparm(Cl = tvCl * exp(nCl))
  fixef(tvV = c(, 1, ))
  fixef(tvCl = c(, 1, ))
  ranef(block(nV, nCl) = c(1, 0.001, 1))
}
"

# Add custom space with an explicit name
all_models <-
  add_CustomSpace(pk_models, custom_pml, SpaceName = "1cptOmegaBlock")
names(all_models)

# Add another custom space with auto-generated name
all_models_2 <- add_CustomSpace(all_models,
  "test() {
  cfMicro(A1, Cl / V)
  dosepoint(A1)
  C = A1 / V
  error(CEps = 1)
  observe(CObs = C + C ^ (0.5) * CEps)
  stparm(V = tvV * exp(nV))
  stparm(Cl = tvCl * exp(nCl))
  fixef(tvV = c(, 1, ))
  fixef(tvCl = c(, 1, ))
  ranef(block(nV, nCl) = c(1, 0.001, 1))
}
")
names(all_models_2) # Will include original names + "1<number>"
```

---

add_Dosepoint *Add Dosepoint in PML models*

Description

Add Dosepoint in PML models

Usage

```
add_Dosepoint(
  PMLParametersSets,
  DosepointName = "A1",
  State = "Present",
  tlag = c(),
  bioavail = c(),
  duration = c(),
  rate = c(),
  PMLStructures = NULL
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
DosepointName	A character string giving the name of the Dosepoint.
State	A character string giving the state of the Dosepoint, must be one of "None", "Present", "Searched". Default is "Present".
tlag	An optional parameter for the time lag. Can be an StParm object, an Expression object, or a character string (which will be converted to an Expression).
bioavail	An optional parameter for bioavailability. Can be an StParm object, an Expression object, or a character string (which will be converted to an Expression).
duration	An optional parameter for the duration of infusion. Can be an StParm object, an Expression object, or a character string (which will be converted to an Expression).
rate	An optional parameter for the rate of infusion. Can be an StParm object, an Expression object, or a character string (which will be converted to an Expression).
PMLStructures	Character or character vector specifying names of PML structures in which the dosepoint statement will be modified. For the naming convention of PMLStructures, see Details section of get_PMLParametersSets().

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Dosepoints(), modify_Dosepoint()

Functions used for Dosepoint specification: Dosepoint(), create_ModelPK(), modify_Dosepoint()

Examples

```
PMLParametersSets <-
  create_ModelPK(CompartmentsNumber = c(1, 2, 3),
                 Absorption = c("First-Order"))

# modify dosepoint
PMLParametersSets <-
  modify_Dosepoint(PMLParametersSets,
                   DosepointName = "Aa",
                   tlag = StParm(StParmName = "Tlag",
                                 State = "Present"),
                   bioavail = StParm(StParmName = "F",
                                     State = "Present"))

# add dosepoint
PMLParametersSets <-
 add_Dosepoint(PMLParametersSets,
               DosepointName = "A1",
               bioavail = Expression("1 - F"),
               duration = Expression("Tlag"))
```

---

add_Spaces *Add Multiple Model Spaces to a PMLModels Object*

Description

Merges a list of new model spaces into an existing `PMLModels` object.

Usage

```
add_Spaces(Spaces, NewSpaces, NewSpacesNames = character())
```

Arguments

`Spaces`	A `PMLModels` object (a named list) representing the existing collection of model spaces. Can be an empty list or a previously created `PMLModels` object.
`NewSpaces`	A list where each element is an internal representation of a single model space (e.g., the list structure produced by `create_ModelPK` or `create_CustomSpace`). If this list is named, these names will be used unless overridden by `NewSpacesNames`.
`NewSpacesNames`	An optional character vector providing explicit names for the spaces listed in `NewSpaces`. If provided: If omitted, the names attached to the `NewSpaces` list itself will be used.

Details

This function provides a general mechanism for combining collections of model spaces.

Naming and Collision Handling: This behavior differs from `add_CustomSpace()`, which automatically renames on collision. `add_Spaces` requires unique, non-colliding names for the merge.

Value

An updated PMLModels object (a named list) containing all the original spaces plus all spaces from NewSpaces. The class "PMLModels" is preserved.

See Also

add_CustomSpace() for adding a single custom space with automatic renaming.

Examples

```
pk_model1 <-
  create_ModelPK(CompartmentsNumber = 1,
                 Absorption = "Intravenous")
pk_models2 <- create_ModelPK(CompartmentsNumber = c(2, 3),
                             Absorption = "First-Order")

# Combine two PMLModels objects (using names from pk_models2)
combined1 <- add_Spaces(pk_model1, pk_models2)
names(combined1)

# Combine using explicit new names
combined2 <-
  add_Spaces(pk_model1,
             pk_models2,
             NewSpacesNames = c("Model_A", "Model_B"))
names(combined2)

# Add a list containing a single custom space
custom_pml <- "test(){ fixef(p=1) }"
custom_space_list <-
  create_CustomSpace(custom_pml, "MyCustom")
combined3 <-
  add_Spaces(pk_model1, custom_space_list)
names(combined3)
```

---

add_StParm *Add Structural parameter into PML models Dosepoints*

Description

Add Structural parameter into PML models Dosepoints

Usage

```
add_StParm(
  PMLParametersSets,
  StParmName,
  Type = "LogNormal",
```

```
  State = "Present",
  ThetaStParm = list(),
  OmegaStParm = list(),
  Covariates = list(),
  PMLStructures = NULL,
  DosepointArgName = character()
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
StParmName	Character specifying the name of the structural parameter to be added.

Type	Character specifying the type of the structural parameter. Options are
	• LogNormal The PML statement of the structural parameter will look like the following:
	`stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`
	• LogNormal1 The PML statement of the structural parameter will look like the following:
	`stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))`
	• LogNormal2 The PML statement of the structural parameter will look like the following:
	`stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))`
	• LogitNormal The PML statement of the structural parameter will look like the following:
	`stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`
	• Normal The PML statement of the structural parameter will look like the following:
	`stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))`

State	character string that indicates the presence of the structural parameter. Options are: • None The structural parameter does not exist in the specified PMLStructures. • Present The structural parameter exists in the specified PMLStructures (the default). • Searched The presence of the structural parameter is searched.
ThetaStParm	A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + StParmName.
OmegaStParm	An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + StParmName
Covariates	A list of covariates (Covariate instances) that should be included in the structural parameter statement.
PMLStructures	Character or character vector specifying names of PML structures to which the structural parameter will be added. For the naming convention of PMLStructures, see Details section of get_PMLParametersSets().

DosepointArgName
: Character specifying the name of the argument in the Dosepoint() instance to add/update the associated structural parameter. Options are bioavail, rate, duration, tlag. Not applicable for custom models

Details

- only special Dosepoint() related structural parameters could be added to built-in models (i.e. created using either create_ModelPD() or create_modelPK(). Due to ambiguity of situation when a structural parameter is added with State == 'None', a warning is given for such cases.
- A structural parameter could be added to the custom model if it not presented in the model yet (as a custom or built-in structural parameter).

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

Dosepoint() list_StParms()

Functions used for StParm specification: StParm(), create_ModelPD(), create_ModelPK(), modify_StParm(), modify_StParmCustom(), remove_StParm()

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(CompartmentsNumber = c(1, 2, 3))

# add Rate structural parameter for all PMLModels
PMLParametersSetsVModDuration <-
 add_StParm(PMLParametersSets,
            StParmName = "Duration",
            ThetaStParm = Theta("tvD",
                          InitialEstimates = 2),
            OmegaStParm = Omega(Name = "nD",
                                State = "Searched"),
            DosepointArgName = "duration")
```

Covariate *Create a new Covariate object and validate it*

Description

Create a new Covariate object and validate it

Usage

```
Covariate(
  Name = character(),
  Type = "Continuous",
  StParmName = character(),
  State = "Present",
  Direction = "Forward",
  Center = "None",
  Categories = c(),
  Thetas = c(),
  Omegas = c(),
  PMLStructure = character()
)
```

Arguments

Name	Character specifying the name of the covariate.
Type	A character specifying the type of the covariate. Possible values are: • Continuous A covariate can take values on a continuous scale. • Categorical A covariate can only take a finite number of values. • Occasion The associated PK parameter may vary within an individual from one event to the next, called interoccasion variability.
StParmName	A character specifying the corresponding structural parameter name.
State	A character string representing the presence of the covariate on the structural parameters. Possible values are: • None The covariate does not have an effect on any structural parameter. • Present The covariate has an effect on the structural parameters (the default). • Searched The effect of the covariate on structural parameters is searched.
Direction	A character string representing the direction of the Covariate. Options are Forward, Backward, Interpolate. Default is Forward. Interpolate is only applicable to Type == "Continuous".
Center	A character string (None, Mean or Median) or numeric value representing the center of the Covariate. Default is None. Valid only if Type == "Continuous".
Categories	A numeric vector representing the categories (at least two) of the covariate. Applicable only if Type is either Occasion or Categorical. The first category is set to the reference category for categorical covariate. If a named vector is used, the names are used as labels for the given dataset,
Thetas	A list of Theta objects representing Thetas covariate effects. Only applicable if Type is either Categorical or Continuous. If Type == "Continuous", one Theta corresponding to current Covariate should be presented. If Type == "Categorical", thetas corresponding to each category (except the reference category) can be specified. If not given, theta(s) will be automatically generated with initial estimate set to 0.0.

Omegas	A list of Omega objects representing the Omegas of the inter-occasion random effects. Applicable only if Type == "Occasion". The number of Omegas should be equal to the number of categories provided. If not given, Omegas will be created automatically with initial estimate set to 1.0.
PMLStructure	PML structure current Covariate instance belongs to.

Value

A Covariate object

See Also

Functions used for Covariate specification: add_Covariate(), create_ModelPD(), create_ModelPK(), remove_Covariate()

Examples

```
WT_Covariate <-
  Covariate(Name = "WT",
            Type = "Continuous",
            StParmName = "V",
            State = "Present",
            Direction = "Forward",
            Center = 70,
            Thetas = Theta("dVdWT", 1))

Race_Covariate <-
  Covariate(
    Name = "Race",
    Type = "Categorical",
    StParmName = "V2",
    State = "Searched",
    Direction = "Backward",
    Center = "None",
    Categories = c(1,2,3))
```

---

create_CustomSpace *Create a Custom Space (PML Model) from Code*

Description

Parses a character string containing custom PML code to create a structured representation within the PMLModels framework. This allows integrating user-defined models with the package's modification and generation tools.

Usage

```
create_CustomSpace(CustomCode = character(), SpaceName = character())
```

Arguments

CustomCode	A character string containing the complete custom PML code (e.g., the content of a test(){...} block, potentially excluding the test(){ and closing } depending on usage context, although including them is safer for parsing). Multi-line strings are collapsed. Cannot be empty.
SpaceName	An optional character string specifying the name for this custom model space. This name will be used as the key for this model within the returned PMLModels list. If omitted or an empty string (""), a unique identifier will be created by concatenating the letter "l" with the number of characters in CustomCode. Providing a meaningful name is recommended for clarity, especially when working with multiple custom models. Ensure provided names are unique if creating multiple custom spaces intended to coexist.

Details

This function acts as a parser for custom PML code. It attempts to identify and extract various PML statements within the provided CustomCode string. Recognized statements include:

- Observations/Responses: observe(), error(), multi(), ordinal(), count(), event(), LL()
- Structural Parameters: stparm()
- Covariates: covariate() (parsed as backward), fcovariate() (parsed as forward), interpolate()
- Dosing: dosepoint(), dosepoint2()
- Parameter Definitions: ranef(), fixef()
- Model Dynamics: deriv(), cfMicro(), cfMacro(), cfMacro1(), transit(), delayInfCpt()
- Other Compartments: urinecpt()

The function cleans the code (removes comments, standardizes spacing) for parsing most statements but uses the original code for stparm and fixef parsing to preserve complex structures.

The extracted components are stored as specific S3 objects (e.g., ObservationCustom, StParmCustom, CovariateCustom, DosepointCustom) within the list element corresponding to the SpaceName. This structured representation allows the custom model to be potentially inspected or manipulated by other package functions, although interacting with built-in models via functions like modify_StParm is generally more robust than modifying custom code components directly.

Value

A list object of class PMLModels. This list contains a *single* element, named according to the provided or generated SpaceName. The value of this element is an internal list structure (created by the internal CustomSpace function) holding the original code and parsed components.

Examples

```
# Example custom PML code (simplified)
custom_pml <- "test() {
  cfMicro(A1, Cl / V)
  dosepoint(A1)
```

```
  C = A1 / V
  error(CEps = 1)
  observe(CObs = C + CEps)
  stparm(V = tvV * exp(nV))
  stparm(Cl = tvCl * exp(nCl))
  fixef(tvV = c(, 1, ))
  fixef(tvCl = c(, 1, ))
  ranef(diag(nV, nCl) = c(1, 1))
}
"

# Create a custom space with an explicit name
custom_model_set <-
  create_CustomSpace(CustomCode = custom_pml,
                     SpaceName = "My1CptOral")

# Print the structure (will show parsed components within the list element)
# print(custom_model_set)

# List the name of the created space
names(custom_model_set) # Output: "My1CptOral"

# Create a custom space with an automatically generated name
custom_model_set_auto_name <- create_CustomSpace(CustomCode = custom_pml)
names(custom_model_set_auto_name)
```

`create_ModelEmax` *Get the list of objects describing the PML models by set of Emax parameters*

Description

This function provides the PML (Pharmacometric Modelling Language) Emax parameter sets based on the specified options. They are available as a list of specific S3 classes.

Usage

```
create_ModelEmax(
  Baseline = FALSE,
  Fractional = FALSE,
  Inhibitory = FALSE,
  Sigmoid = FALSE,
  ByVector = FALSE,
  ...
)
```

Arguments

Baseline	Logical indicating whether the Emax model contains a baseline response. If it is set to TRUE, the new parameter, E0, for baseline response is added to the model. Default is FALSE.
Fractional	Logical indicating whether the Emax model with baseline response is fractional. Applicable only for the Emax models with baseline response, otherwise a warning is given and current parameter is ignored. Default is FALSE.
Inhibitory	Logical indicating whether the model is inhibitory. If it is set to TRUE, the structural parameters 'EC50' and 'Emax' change to 'IC50' (concentration producing 50% of maximal inhibition) and 'Imax'. Default is FALSE.
Sigmoid	Logical indicating whether the model is sigmoidal. If it is set to TRUE, the Hill coefficient, 'Gam', is added to the model. Default is FALSE.
ByVector	Logical indicating whether each element in vectorized argument should be treated as a separate PML structure (i.e. treated as data.frame vectors), TRUE, or as parameters to obtain a pool (i.e. expanded) of PML structures, FALSE. Default is FALSE (one value for a function call).
...	Additional named arguments, including Structural parameters (StParm), Covariates, Dosepoints (for PK models), Thetas and Omegas. See 'Additional arguments' section.

Value

A list of PML models (PMLModels class instance) matching the specified options.

Examples

```
# Get Emax model set with default options
PDParametersSets <- create_ModelEmax()

# Create Emax model set with all possible combinations
# will give a warning since When 'Baseline == FALSE',
# there could be no model with 'Fractional == TRUE'
PDParametersSets <-
  create_ModelEmax(Baseline = TRUE,
                   Fractional = c(FALSE, TRUE),
                   Inhibitory = c(FALSE, TRUE),
                   Sigmoid = c(FALSE, TRUE),
                   ByVector = FALSE)
```

---

create_ModelPD *Get the list of objects describing the PML models by set of PD parameters*

Description

This function provides the PML (Pharmacometric Modelling Language) PD parameter sets based on the specified options. They are available as a list of specific S3 classes.

Usage

```
create_ModelPD(
  Type = "Emax",
  Baseline = FALSE,
  Fractional = FALSE,
  Inhibitory = FALSE,
  Sigmoid = FALSE,
  ByVector = FALSE,
  ...
)
```

Arguments

Type	Pharmacodynamic model type. Currently, only Emax is supported.
Baseline	Logical indicating whether the Emax model contains a baseline response. If it is set to TRUE, the new parameter, E0, for baseline response is added to the model. Default is FALSE.
Fractional	Logical indicating whether the Emax model with baseline response is fractional. Applicable only for the Emax models with baseline response, otherwise a warning is given and current parameter is ignored. Default is FALSE.
Inhibitory	Logical indicating whether the model is inhibitory. If it is set to TRUE, the structural parameters 'EC50' and 'Emax' change to 'IC50' (concentration producing 50% of maximal inhibition) and 'Imax'. Default is FALSE.
Sigmoid	Logical indicating whether the model is sigmoidal. If it is set to TRUE, the Hill coefficient, 'Gam', is added to the model. Default is FALSE.
ByVector	Logical indicating whether each element in vectorized argument should be treated as a separate PML structure (i.e. treated as data.frame vectors), TRUE, or as parameters to obtain a pool (i.e. expanded) of PML structures, FALSE. Default is FALSE (one value for a function call).
...	Additional named arguments, including Structural parameters (StParm), Covariates, Dosepoints (for PK models), Thetas and Omegas. See 'Additional arguments' section.

Details

The names of PMLStructure are constructed by the following parts:

- Baseline if presented (abbreviated as 'E0'),
- Fractional if presented (abbreviated as '1+'),
- Inhibitory (abbreviated as 'Imax' if the model is inhibitory and 'Emax' otherwise),
- Sigmoid if presented (abbreviated as 'Gam').

Value

A list of PML models (PMLModels class instance) matching the specified options.

Additional arguments

Additional arguments (ellipsis) will be applied sequentially. They can be used to add or modify Structural parameters (StParm), Covariates, Observations, Dosepoints (for PK models); by the way it is advised to use specific functions for it (see 'See Also' section for the references). Also it is possible to modify Omegas and Thetas, but it is impossible to add them (they are parts of other structures). If `PMLStructure` argument is not specified, class instances will be modified or added in all PML structures. If `PMLStructure` argument is specified, class instances in the specified PML structure will be modified/added. Note that only one PML structure could be added to the class instance. If multiple structures should be modified, suggest to use specific functions.

See Also

Functions used for StParm specification: `StParm()`, `add_StParm()`, `create_ModelPK()`, `modify_StParm()`, `modify_StParmCustom()`, `remove_StParm()`

Functions used for Observation specification: `Observation()`, `ObservationCustom()`, `Sigmas()`, `create_ModelPK()`, `modify_Observation()`, `remove_Observation()`

Functions used for Omega specification: `Omega()`, `create_ModelPK()`, `modify_Omega()`

Functions used for Theta specification: `InitialEstimate()`, `Theta()`, `create_ModelPK()`, `modify_Theta()`

Functions used for Covariate specification: `Covariate()`, `add_Covariate()`, `create_ModelPK()`, `remove_Covariate()`

Examples

```
# Get PD model set with default options
PDParametersSets <- create_ModelPD(Type = "Emax")

# Create PD model set with all possible combinations
# will give a warning since When 'Baseline == FALSE',
# there could be no model with 'Fractional == TRUE'
PDParametersSets <-
  create_ModelPD(Type = "Emax",
                 Baseline = FALSE,
                 Inhibitory = c(FALSE, TRUE),
                 Sigmoid = c(FALSE, TRUE),
                 ByVector = FALSE)
```

---

create_ModelPK *Get the list of objects describing the PML models by set of PK parameters*

Description

This function provides the PML (Pharmacometric Modelling Language) PK parameter sets based on the specified options. They are available as a list of specific S3 classes.

Usage

```
create_ModelPK(
  CompartmentsNumber = 1,
  Absorption = "Intravenous",
  Parameterization = "Clearance",
  Saturation = FALSE,
  EliminationCpt = FALSE,
  FractionExcreted = FALSE,
  ByVector = FALSE,
  ClosedForm = TRUE,
  ...
)

get_PMLParametersSets(
  CompartmentsNumber = 1,
  Absorption = "Intravenous",
  Parameterization = "Clearance",
  Saturation = FALSE,
  EliminationCpt = FALSE,
  FractionExcreted = FALSE,
  ByVector = FALSE,
  ClosedForm = TRUE,
  ...
)
```

Arguments

`CompartmentsNumber`
: The number of compartments in the model. Supported embedded models are 1-, 2-, 3-compartments. Default is 1.

`Absorption`
: The absorption type of the model. Supported types are:

  - `Intravenous` (Default) - Dose is given in the main compartment (A1) directly.
  - `First-Order` - Dose is absorbed to the main compartment (A1) from the absorption compartment (Aa) by first-order kinetic.
  - `Gamma` - Dose is absorbed to A1 by Gamma Distributed delay kinetic.
  - `Inverse Gaussian` - Dose is absorbed to A1 by Inverse Gaussian Distributed delay kinetic.
  - `Weibull` - Dose is absorbed to A1 by Weibull Distributed delay kinetic.

`Parameterization`
: The parameterization type. Possible options are `Clearance` - Clearance parameters: Cl, Cl2 to be used and `Micro` - Micro parameters: Ke, K12, K21 to be used. Default is `Clearance`.

`Saturation`
: Logical indicating whether saturation should be considered. Default is `FALSE`.

`EliminationCpt`
: Logical indicating whether elimination compartment should be included. Default is `FALSE`.

`FractionExcreted`
: Logical indicating whether fraction excreted structural parameter should be included in urinecpt statement: `urinecpt(A0 = Cl * C, fe=Fe)`. Valid only if `EliminationCpt == TRUE`. Default is `FALSE`.

`ByVector`
: Logical indicating whether each element in vectorized argument should be treated as a separate PML structure (i.e. treated as data.frame vectors), `TRUE`, or as parameters to obtain a pool (i.e. expanded) of PML structures, `FALSE`. Default is `FALSE` (one value for a function call).

`ClosedForm`
: Logical indicating whether closed forms (cfMicro) should be used when possible. Note that closed forms are not available for the models with elimination compartment, models with saturation or absorption types other than `Intravenous` or `First-Order`. The models with interpolated covariates must use `ClosedForm == FALSE`. Default is `TRUE` (one value for a function call).

`...`
: Additional named arguments, including Structural parameters (StParm), Covariates, Dosepoints (for PK models), Thetas and Omegas. See 'Additional arguments' section.

Details

The names of `PMLStructure` are constructed by the following parts:

- Model type ('PK'),
- Compartments number
- Abbreviated absorption type:
  - 'IV' for Intravenous,
  - 'FO' for First-Order,
  - 'G' for Gamma,
  - 'W' for Weibull,
  - 'IG' for Inverse Gaussian,
- Abbreviated parameterization ('C' for Clearance and 'M' for Micro),
- Abbreviated saturation if presented ('S'),
- Abbreviated elimination if presented ('E'),
- Abbreviated fraction excreted if presented ('F').

Value

A list of PML models (`PMLModels` class instance) matching the specified options.

Additional arguments

Additional arguments (ellipsis) will be applied sequentially. They can be used to add or modify Structural parameters (StParm), Covariates, Observations, Dosepoints (for PK models); by the way it is advised to use specific functions for it (see 'See Also' section for the references). Also it is possible to modify Omegas and Thetas, but it is impossible to add them (they are parts of other structures). If `PMLStructure` argument is not specified, class instances will be modified or added in all PML structures. If `PMLStructure` argument is specified, class instances in the specified PML structure will be modified/added. Note that only one PML structure could be added to the class instance. If multiple structures should be modified, suggest to use specific functions.

See Also

Functions used for Dosepoint specification: Dosepoint(), add_Dosepoint(), modify_Dosepoint()

Functions used for StParm specification: StParm(), add_StParm(), create_ModelPD(), modify_StParm(), modify_StParmCustom(), remove_StParm()

Functions used for Observation specification: Observation(), ObservationCustom(), Sigmas(), create_ModelPD(), modify_Observation(), remove_Observation()

Functions used for Omega specification: Omega(), create_ModelPD(), modify_Omega()

Functions used for Theta specification: InitialEstimate(), Theta(), create_ModelPD(), modify_Theta()

Functions used for Covariate specification: Covariate(), add_Covariate(), create_ModelPD(), remove_Covariate()

Examples

```
# Get PK model set with default options
PMLParametersSets <- create_ModelPK()

#' # Get PK Model search with custom options:
# will create 2 PML Parameters Sets with 2 and 3 compartments,
# with Absorption First-Order and Gamma accordingly:
ModelPKSearch <-
  create_ModelPK(CompartmentsNumber = c(2, 3),
                 Parameterization = "Micro",
                 Absorption = c("First-Order", "Gamma"),
                 ByVector = TRUE,
                 ClosedForm = TRUE)


# Next example will create a set of 4 PMLParametersSets:
# a combination of models with 2 and 3 compartments and First-Order and Gamma Absorption
PMLParametersSets <-
  create_ModelPK(CompartmentsNumber = c(2, 3),
                 Absorption = c("First-Order", "Gamma"),
                 ByVector = FALSE,
                 ClosedForm = FALSE)

# Create 2 PML Parameters Sets with elimination compartment and fraction excreted
# and add zero order absorption to the main dosepoint of the PML Structure
# with infusion
PMLParametersSets <-
  create_ModelPK(CompartmentsNumber = 1,
                 Absorption = c("Intravenous", "Gamma"),
                 EliminationCpt = TRUE,
                 FractionExcreted = TRUE,
                 duration = StParm(StParmName = "Duration",
                                   OmegaStParm = Omega(State = "None")),
                                   PMLStructure = "PK1IVCEF")

# Create 4 PML Parameters Sets, then modify `Cl` structural parameter for all sets,
# with 2 initial estimates sets to be searched,
# add `tlag` as a structural parameter `Tlag` to 1 compartment First-Order PML parameters set,
```

```
# change `tvKa` Theta initial estimate,
# change `nV` Omega initial estimate,
# change `CObs` Observation sigmas,
# add structural parameter `Rate` for 1 compartment Weibull Parameters set,
# add `Weight` covariate for all structural parameters to be searched.

PMLParametersSets <-
   create_ModelPK(
     CompartmentsNumber = 1,
     Absorption = c("First-Order", "Weibull"),
     ByVector = FALSE,
     Cl = StParm(
       StParmName = "Cl",
       Type = "LogNormal2",
       ThetaStParm =
         Theta(Name = "tvCl",
               InitialEstimates =
                 InitialEstimate(c(-Inf, 0.2, Inf),
                                 c(0, 3, 10)))
     ),
     tlag = StParm(
       StParmName = "Tlag",
       State = "Searched",
       PMLStructure = "PK1FOC",
       Covariates = list(
         Age = Covariate(
           Name = "Age",
           Type = "Categorical",
           State = "Searched",
           Direction = "Backward",
           Center = "None",
           Categories = c(1, 2, 3)
         )
       )
     ),
     tvKa = Theta(Name = "tvKa", InitialEstimates = 10),
     nV = Omega(Name = "nV", InitialOmega = 0.1),
     CObs = Observation(
       ObservationName = "CObs",
       SigmasChosen = list(
         AdditiveMultiplicative = c(PropPart = 0.1, AddPart = 2),
         Proportional = 1
       )
     ),
     A1 = Dosepoint(
       DosepointName = "A1",
       rate = StParm(StParmName = "Rate"),
       PMLStructure = "PK1WC"
     ),
     Weight = Covariate(
       Name = "Weight",
       State = "Searched",
       Center = "Median"
```

```
    )
  )
```

`create_pyDarwinOptions` *Create pyDarwin Options*

Description

Generates a list of parameters to be used in a pyDarwin run.

Usage

```
create_pyDarwinOptions(
  author = "",
  project_name = NULL,
  algorithm = c("GA", "EX", "MOGA", "MOGA3", "GP", "RF", "GBRT", "PSO"),
  GA = pyDarwinOptionsGA(),
  MOGA = pyDarwinOptionsMOGA(),
  PSO = pyDarwinOptionsPSO(),
  random_seed = 11,
  num_parallel = 4,
  num_generations = 6,
  population_size = 4,
  num_opt_chains = 4,
  exhaustive_batch_size = 100,
  crash_value = 99999999,
  penalty = pyDarwinOptionsPenalty(),
  effect_limit = -1,
  downhill_period = 2,
  num_niches = 2,
  niche_radius = 2,
  local_2_bit_search = TRUE,
  final_downhill_search = TRUE,
  local_grid_search = FALSE,
  max_local_grid_search_bits = 5,
  search_omega_blocks = FALSE,
  search_omega_bands = FALSE,
  individual_omega_search = TRUE,
  search_omega_sub_matrix = FALSE,
  max_omega_sub_matrix = 4,
  model_run_timeout = 1200,
  model_run_priority_class = c("below_normal", "normal"),
  postprocess = pyDarwinOptionsPostprocess(),
  keep_key_models = TRUE,
  keep_best_models = TRUE,
```

```
  rerun_key_models = FALSE,
  rerun_front_models = TRUE,
  use_saved_models = FALSE,
  saved_models_file = "{working_dir}/models0.json",
  saved_models_readonly = FALSE,
  remove_run_dir = FALSE,
  remove_temp_dir = FALSE,
  keep_files = c("dmp.txt", "posthoc.csv"),
  keep_extensions = NULL,
  use_system_options = TRUE,
  model_cache = "darwin.MemoryModelCache",
  model_run_man = c("darwin.LocalRunManager", "darwin.GridRunManager"),
  engine_adapter = c("nlme", "nonmem"),
  skip_running = FALSE,
  working_dir = NULL,
  data_dir = NULL,
  output_dir = "{working_dir}/output",
  temp_dir = NULL,
  key_models_dir = "{working_dir}/key_models",
  non_dominated_models_dir = "{working_dir}/non_dominated_models",
  nlme_dir = "C:/Program Files/Certara/NLME_Engine",
  gcc_dir = "C:/Program Files/Certara/mingw64",
  nmfe_path = NULL,
  rscript_path = file.path(normalizePath(R.home("bin")), "Rscript"),
  generic_grid_adapter = pyDarwinOptionsGridAdapter(),
  remote_run = FALSE,
  ...
)
```

Arguments

author	Character string: The name of the author.
project_name	Character string (optional): The name of the project. If not specified, pyDarwin will set its value to the name of the parent folder of the options file.
algorithm	Character string: One of EX, GA, MOGA, MOGA3, GP, RF, GBRT, PSO. See section Details below for more information.
GA	List: Options specific to the Genetic Algorithm (GA). See pyDarwinOptionsGA(). Ignored if algorithm is not "GA".
MOGA	List: Options specific to the Multi-Objective Genetic Algorithm (MOGA or MOGA3). See pyDarwinOptionsMOGA(). Ignored if algorithm is not "MOGA" or "MOGA3".
PSO	List: Options specific to the Particle Swarm Optimization (PSO). See pyDarwinOptionsPSO(). Ignored if algorithm is not "PSO".
random_seed	Positive integer: Seed for random number generation.
num_parallel	Positive integer: Number of models to execute in parallel, i.e., how many threads to create to handle model runs. Default: 4.

num_generations
: Positive integer: Number of iterations or generations of the search algorithm to run. Not used/required for EX. Default: 6.

population_size
: Positive integer: Number of models to create in every generation. Not used/required for EX. Default: 4.

num_opt_chains
: Positive integer: Number of parallel processes to perform the "ask" step (to increase performance). Required only for GP, RF, and GBRT. Default: 4.

exhaustive_batch_size
: Positive integer: Batch size for the EX (Exhaustive Search) algorithm. Default: 100.

crash_value
: Positive real: Value of fitness or reward assigned when model output is not generated. Should be set larger than any anticipated completed model fitness. Default: 99999999.

penalty
: List: Options specific to the penalty calculation. See pyDarwinOptionsPenalty().

effect_limit
: Integer: Limits number of effects. Applicable only for NONMEM and GA/MOGA/MOGA3. If < 1, effect limit is turned off. Default: -1.

downhill_period
: Integer: How often to run the downhill step. If < 1, no periodic downhill search will be performed. Default: 2.

num_niches
: Integer: Used for GA and downhill. A penalty is assigned for each model based on the number of similar models within a niche radius. This penalty is applied only to the selection process (not to the fitness of the model). The purpose is to ensure maintaining a degree of diversity in the population. num_niches is also used to select the number of models that are entered into the downhill step for all algorithms, except EX. Default: 2.

niche_radius
: Positive real: The radius of the niches. Used to define how similar pairs of models are, for Local search and GA sharing penalty. Default: 2.

local_2_bit_search
: Logical: Whether to perform the two-bit local search. Substantially increases search robustness. Done starting from num_niches models. Ignored for MOGA and MOGA3. Default: TRUE.

final_downhill_search
: Logical: Whether to perform a local search (1-bit and 2-bit) at the end of the global search. Default: TRUE.

local_grid_search
: Logical: Whether to perform a local grid search during downhill. Default: FALSE.

max_local_grid_search_bits
: Positive integer: Maximum number of bits to explore in the local grid search. Default: 5.

search_omega_blocks
: Logical: Whether to perform search for block omegas. Used only when engine_adapter == 'nlme'. Default: FALSE.

search_omega_bands
: Logical: Whether to perform search for band omegas. Used only when engine_adapter == 'nonmem'. Default: FALSE.

`individual_omega_search`
: Logical: If TRUE, every omega search block is handled individually. If FALSE, all search blocks have the same pattern. Default: TRUE.

`search_omega_sub_matrix`
: Logical: Set to TRUE to search omega submatrix. Default: FALSE.

`max_omega_sub_matrix`
: Integer: Maximum size of sub matrix to use in search. Default: 4.

`model_run_timeout`
: Positive real: Time (seconds) after which the execution will be terminated, and the crash value assigned. Default: 1200.

`model_run_priority_class`
: Character string (Windows only): Priority class for child processes. Options are `below_normal` (default) and `normal`.

`postprocess`
: List: Options specific to postprocessing. See `pyDarwinOptionsPostprocess()`. For `algorithm = "MOGA3"`, postprocessing is required to define objectives and constraints. For `algorithm = "MOGA"` (NSGA-II), pyDarwin does not use post-processing for objective calculation.

`keep_key_models`
: Logical: Whether to save the best model from every generation to `key_models_dir`. Default: TRUE.

`keep_best_models`
: Logical: If `TRUE` (default), saves only "key" models that represent an improvement in fitness value compared to the previous overall best model. Models are saved to `key_models_dir`. Not applicable to Exhaustive Search (EX). Default: `TRUE`.

`rerun_key_models`
: Logical: Whether to re-run key models that lack output after the search. Default: FALSE.

`rerun_front_models`
: Logical: Similar to `rerun_key_models`, but for non-dominated models (typically from MOGA/MOGA3). Models are copied to `non_dominated_models_dir`. Default: TRUE.

`use_saved_models`
: Logical: Whether to restore saved Model Cache from file. Default: FALSE.

`saved_models_file`
: Character string: The file from which to restore Model Cache. Default: "{working_dir}/models0.json".

`saved_models_readonly`
: Logical: Do not overwrite the `saved_models_file` content. Default: FALSE.

`remove_run_dir`
: Logical: If TRUE, delete the entire model run directory, otherwise only unnecessary files. Default: FALSE.

`remove_temp_dir`
: Logical: Whether to delete the entire `temp_dir` after the search. Default: FALSE

`keep_files`
: Character vector (optional): List of exact file names to keep when cleaning up run directories. Default is `c("dmp.txt", "posthoc.csv")` when `engine_adapter` is "nlme".

`keep_extensions`
: Character vector (optional): List of file extensions (without dot) to keep. Default: NULL.

`use_system_options`
: Logical: Whether to override options with environment-specific values. Default: TRUE.

`model_cache`
: Character string: ModelCache subclass to be used. Default: "darwin.MemoryModelCache".

`model_run_man`
: Character string: ModelRunManager subclass to be used. Options: "darwin.LocalRunManager" (default), "darwin.GridRunManager".

`engine_adapter`
: Character string: ModelEngineAdapter subclass. Options: "nlme" (default), "nonmem".

`skip_running`
: Logical: If TRUE, no actual NM/NLME runs will be performed. Default: FALSE.

`working_dir`
: Character string (optional): Project's working directory.

`data_dir`
: Character string (optional): Directory for datasets.

`output_dir`
: Character string: Directory for pyDarwin output. Default: "{working_dir}/output".

`temp_dir`
: Character string (optional): Parent directory for model run subdirectories.

`key_models_dir`
: Character string: Directory where key/best models will be saved. Default: "{working_dir}/key_models".

`non_dominated_models_dir`
: Character string: Directory where non-dominated models will be saved (typically for MOGA/MOGA3). Default: "{working_dir}/non_dominated_models".

`nlme_dir`
: Character string (optional): Directory for NLME Engine installation.

`gcc_dir`
: Character string (optional): Directory for Mingw-w64 compiler.

`nmfe_path`
: Character string (optional): Path to NONMEM execution command.

`rscript_path`
: Character string (optional): Path to Rscript executable.

`generic_grid_adapter`
: List: Options for grid execution. See `pyDarwinOptionsGridAdapter()`. Used if `model_run_man == "darwin.GridRunManager"`.

`remote_run`
: Logical: Indicates if pyDarwin execution is for a remote host. Default: `FALSE`.

`...`
: Additional parameters.

Details

The algorithm parameter specifies the search algorithm. The algorithm "MOGA" and "MOGA3" are used for multi-objective optimization: "MOGA" uses NSGA-II (see the documentation at https://pymoo.org/algorithms/moo/nsga2.html?highlight=nsga%20ii), and "MOGA3" uses NSGA-III (see the documentation at https://pymoo.org/algorithms/moo/nsga3.html?highlight=nsga%20ii). For MOGA3, the objectives and constraints must be defined and returned by post-processing scripts (post_run_r_code or post_run_python_code) in a specific format:

- R scripts should return a list of two vectors: the first vector is for the objectives and the second one is for the constraints. If no constraints, the second vector should be empty.

- Python scripts should return a tuple of two lists: the first list is for the objectives and the second one is for the constraints). If no constraints, the second list should be empty.

Other algorithms include "EX" (Exhaustive), "GA" (Genetic Algorithm), "GP" (Gaussian Process), "RF" (Random Forest), "GBRT" (Gradient Boosted Random Tree), and "PSO" (Particle Swarm Optimization).

Please see pyDarwin documentation for complete details on all options.

Value

A list of pyDarwin options.

Examples

```
# Basic options with GA
ga_opts <- create_pyDarwinOptions(author = "Jane Doe", algorithm = "GA")

# Options for MOGA (NSGA-II)
# pyDarwin internally uses 2 objectives; postprocessing for objectives is not used by pyDarwin.
moga_opts_nsga2 <- create_pyDarwinOptions(
  author = "J. Doe",
  project_name = "MOGA_Test_NSGA2",
  algorithm = "MOGA", # NSGA-II
  MOGA = pyDarwinOptionsMOGA(), # Default MOGA options are suitable
  population_size = 50,
  num_generations = 100,
  engine_adapter = "nonmem",
  nmfe_path = "/opt/NONMEM/nm75/run/nmfe75"
)

# Options for MOGA3 (NSGA-III with 3 objectives, 1 constraint via R postprocessing)
moga_opts_nsga3_custom <- pyDarwinOptionsMOGA(
  objectives = 3,
  names = c("AIC", "NumEffects", "RunTime"), # Example custom names
  constraints = 1,
  partitions = 10 # Custom partitions
)
main_opts_nsga3 <- create_pyDarwinOptions(
  author = "J. Doe",
  project_name = "MOGA_Test_NSGA3",
  algorithm = "MOGA3", # NSGA-III
  MOGA = moga_opts_nsga3_custom,
  population_size = 60, # NSGA-III population size might need adjustment
  num_generations = 100,
  postprocess = pyDarwinOptionsPostprocess( # Required for MOGA3
    use_r = TRUE,
    post_run_r_code = "{project_dir}/moga3_postprocess.R"
  ),
  engine_adapter = "nonmem",
  nmfe_path = "/opt/NONMEM/nm75/run/nmfe75"
)
```

`Dosepoint` *Create a new Dosepoint object and validate it*

Description

Create a new Dosepoint object and validate it

Usage

```
Dosepoint(
  DosepointName = "A1",
  State = "Present",
  tlag = c(),
  bioavail = c(),
  duration = c(),
  rate = c(),
  PMLStructure = character()
)
```

Arguments

`DosepointName`	A character string giving the name of the Dosepoint.
`State`	A character string giving the state of the Dosepoint, must be one of "None", "Present", "Searched". Default is "Present".
`tlag`	An optional parameter for the time lag. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`bioavail`	An optional parameter for bioavailability. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`duration`	An optional parameter for the duration of infusion. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`rate`	An optional parameter for the rate of infusion. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`PMLStructure`	A character string that indicates a specific PML structure this Dosepoint definition should be associated with.

Value

A new Dosepoint object.

See Also

`list_Dosepoints()`, `add_Dosepoint()`, `modify_Dosepoint()`, `StParm()`, `Expression()`

Functions used for Dosepoint specification: `add_Dosepoint()`, `create_ModelPK()`, `modify_Dosepoint()`

Examples

```
# Using StParm objects
TlagStParm <- StParm("Tlag",
                     Type = "LogNormal",
                     ThetaStParm = Theta(Name = "tvTlag", InitialEstimates = 0.1))
FStParm <- StParm("F", ThetaStParm = Theta(Name = "tvF")) # Assuming Theta exists

dp1 <- Dosepoint(DosepointName = "GutInput",
                 State = "Present",
                 tlag = TlagStParm,
                 bioavail = FStParm)

# Using Expression objects
dp2 <- Dosepoint(DosepointName = "Infusion",
                 rate = Expression("RateVal",
                                   ContainedStParms =
                                     list(StParm("RateVal",
                                                 ThetaStParm = Theta("tvRateVal")))))

# Using a character string (will be converted to Expression internally)
dp3 <- Dosepoint(DosepointName = "Bolus",
                 bioavail = "SystemicF") # Converted to Expression("SystemicF")
```

Expression *Create an Expression object*

Description

Represents a PML expression that can include text and structural parameters (StParm). This is used for arguments like tlag, bioavail, duration, or rate in Dosepoint.

Usage

```
Expression(
  ExpressionText = character(),
  ContainedStParms = list(),
  State = "Present"
)
```

Arguments

`ExpressionText`	Character string. The primary textual representation of the expression (e.g., "1-F", "Tlag", "Rate * exp(Effect)").
`ContainedStParms`	List. A list containing any StParm objects that are referenced by or associated with this `ExpressionText`. The `State` of structural parameters depends on the `State` of expression (i.e. specification of `State` is not supported).

State — Character string. The state of the expression and the associated structural parameters, controlling its inclusion or search status in the model. Must be one of 'Present', 'None', or 'Searched'.

Value

A new object of class Expression, which is a list containing the provided arguments (ExpressionText, ContainedStParms, State).

See Also

Dosepoint(), StParm()

Examples

```
PMLParametersSets <-
get_PMLParametersSets(CompartmentsNumber = c(1, 2, 3),
                      Absorption = c("First-Order"))
# add dosepoint
PMLParametersSets <-
  add_Dosepoint(PMLParametersSets,
                DosepointName = "A1",
                bioavail = Expression("1 - Fa"),
                duration = Expression("Tlag",
                ContainedStParms = list(StParm("Tlag"))))
```

---

get_ModelTermsToMap *Get Model Terms to Map*

Description

This function retrieves the model terms that can be mapped from a set of PML models.

Usage

```
get_ModelTermsToMap(PMLParametersSets)
```

Arguments

PMLParametersSets — An object of class "PMLModels" containing PML model parameters.

Value

A list with two elements: "Required" and "Optional," representing the model terms that can be mapped.

See Also

create_ModelPK() create_ModelPD() create_CustomSpace()

Examples

```
# Load your PMLModels object
PMLParametersSets <-
  create_ModelPK(
    Absorption = c("First-Order", "Weibull"),
    CObs = Observation(
      ObservationName = "CObs",
      BQL = TRUE),
    A1 = Dosepoint(
      DosepointName = "A1",
      rate = StParm(StParmName = "Rate")),
    Weight = Covariate(
      Name = "Weight",
      Center = "Median")
)

# Get the model terms to map
terms_to_map <- get_ModelTermsToMap(PMLParametersSets)
print(terms_to_map$Required)
print(terms_to_map$Optional)
```

`InitialEstimate` *Create an object of class InitialEstimate*

Description

This function creates an object of class `InitialEstimate` that contains initial parameter estimates for a model `Theta()`. The estimates can be passed to the function as a single numeric value or as a vector of length three containing lower bound, estimate, and upper bound. If multiple sets of estimates are required, they can be passed as additional arguments, each separated by commas.

Usage

```
InitialEstimate(Initial = numeric(), ...)
```

Arguments

`Initial`	Numeric. Initial estimate for the model parameter.
`...`	Additional initial estimate(s) for the model parameter.

Value

An object of class `InitialEstimate`.

See Also

Functions used for Theta specification: `Theta()`, `create_ModelPD()`, `create_ModelPK()`, `modify_Theta()`

Examples

```
InitialEstimate(1)
InitialEstimate(c(0, 1, Inf), c(-Inf, 2, 10))
```

---

list_Covariates *List Covariates in the currrent PML set*

Description

This function lists the names of covariates in a given set of PMLParametersSets.

Usage

```
list_Covariates(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeAll	Logical. Should the names of covariates with None state or covariates inside structural parameters with None state be included or not.
IncludeCustom	Logical. Should the names of covariate, fcovariate and interpolate statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the names of covariates.

See Also

add_Covariate() remove_Covariate() Covariate()

Examples

```
PMLParametersSets <- get_PMLParametersSets()
PMLParametersSets <- add_Covariate(PMLParametersSets,
                                   Name  = "WT")
list_Covariates(PMLParametersSets)
```

list_Dosepoints *List Dosepoints in the current PML set*

Description

This function lists the names of dosepoints in a given set of PMLParametersSets.

Usage

```
list_Dosepoints(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeAll	Logical. Should the names of dosepoints with None state be included or not. Default is FALSE.
IncludeCustom	Logical. Should the names of custom dosepoint and dosepoint2 statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the names of dosepoints

See Also

modify_Dosepoint()

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(
    Absorption = c("First-Order", "Gamma"))
list_Dosepoints(PMLParametersSets)
```

list_Observations *List Observations in the current PML set*

Description

This function lists the names of Observations in a given PMLModels class instance.

Usage

```
list_Observations(
  PMLParametersSets,
  IncludeCustom = TRUE,
  ObservationsOnly = TRUE
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeCustom	Logical. Should the names of responses (observe, multi, ordinal, count, event and LL) from the PML code of custom spaces be included or not. Default is TRUE.
ObservationsOnly	Logical. If TRUE (default), only the names of observe responses are included in the PML code generated for custom spaces. Non-observed response names (such as multi, ordinal, count, event, and LL) are not included. Ignored if IncludeCustom == FALSE.

Value

A character vector containing the names of Observations

See Also

Observation() modify_Observation() remove_Observation()

Examples

```
PMLParametersSets <-
  create_ModelPK(
    Absorption = c("First-Order", "Gamma"),
    EliminationCpt = c(TRUE, FALSE))
list_Observations(PMLParametersSets)
```

---

list_Omegas *List Unique Omega Names*

Description

This function lists the unique names of Omega parameters in a given set.

Usage

```
list_Omegas(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	PMLModels class instance or an element (one PML structure) of this class or StParm class.
IncludeAll	Logical. Whether should the omega names to be inlcuded from structural parameters, covariates or omegas with a State == 'None'.
IncludeCustom	Logical. Should the names of custom ranef statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the unique names of Omega parameters.

See Also

Omega() modify_Omega()

Examples

```
PMLParametersSets <- create_ModelPK()
list_Omegas(PMLParametersSets)
```

---

list_StParms *List Structural Parameters in the currrent PML set*

Description

This function lists the names of structural parameters in a given set of PMLParametersSets.

Usage

```
list_StParms(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeAll	Logical. Should the names of structural parameters with None state be included or not. Default is FALSE.
IncludeCustom	Logical. Should the names of custom stparm statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the names of structural parameters.

See Also

add_StParm() modify_StParm()

Examples

```
PMLParametersSets <- get_PMLParametersSets()
list_StParms(PMLParametersSets)
```

`list_Thetas` *List Unique Theta Names*

Description

This function lists the unique names of Theta parameters in a given set.

Usage

```
list_Thetas(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

`PMLParametersSets`	`PMLModels` class instance or an element (one PML structure) of this class or `StParm` class.
`IncludeAll`	Logical. Whether should the Theta names to be inlcuded from structural parameters, covariates or thetas with a `State == 'None'`.
`IncludeCustom`	Logical. Should the names of custom `theta` statements (from the PML code of custom spaces) be included or not. Default is `TRUE`.

Value

A character vector containing the unique names of Theta parameters.

See Also

Theta() modify_Theta()

Examples

```
PMLParametersSets <- create_ModelPD()
list_Thetas(PMLParametersSets)
```

`modify_Dosepoint` *Modify Dosepoint in PML models*

Description

Modify Dosepoint in PML models

Usage

```
modify_Dosepoint(
  PMLParametersSets,
  DosepointName,
  tlag,
  bioavail,
  duration,
  rate,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`DosepointName`	A character string giving the name of the Dosepoint.
`tlag`	An optional parameter for the time lag. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`bioavail`	An optional parameter for bioavailability. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`duration`	An optional parameter for the duration of infusion. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`rate`	An optional parameter for the rate of infusion. Can be an `StParm` object, an `Expression` object, or a character string (which will be converted to an Expression).
`PMLStructures`	Character or character vector specifying names of PML structures in which the dosepoint statement will be modified. For the naming convention of PMLStructures, see Details section of `get_PMLParametersSets()`.

Details

This function can only be used to modify the structural parameters in the built-in models (i.e., created using either `create_ModelEmax()` or `create_ModelPK()`).

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

list_Dosepoints(), add_Dosepoint()

Functions used for Dosepoint specification: Dosepoint(), add_Dosepoint(), create_ModelPK()

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(CompartmentsNumber = c(1, 2, 3))
# update structural paramter type
PMLParametersSetsVMod <-
 modify_Dosepoint(PMLParametersSets,
            DosepointName = "A1",
            tlag = StParm(StParmName = "Tlag",
                          State = "Searched"))
```

---

modify_Observation *Modify Observation class in PML models*

Description

Modify Observation class in PML models

Usage

```
modify_Observation(
  PMLParametersSets,
  ObservationName,
  SigmasChosen,
  BQL,
  BQLValue,
  Frozen,
  ResetObs,
  Covariates,
  PMLStructures = NULL
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
ObservationName	A character string giving the name of the Observation.
SigmasChosen	a Sigmas class instance or a list specifying the chosen sigma values for different error models. 0s are treated as no values. Inside Observation class it is transormed and kept as Sigmas class. The list could contain the following error models:

- Additive The additive error sigma value.
- LogAdditive The log-additive error sigma value.
- Proportional The proportional error sigma value.
- AdditiveMultiplicative A numeric vector specifying the additive and multiplicative parts for the additive-multiplicative error model. The vector should have names `PropPart` and `AddPart`.
- MixRatio A numeric vector specifying the proportional and additive parts for the mix-ratio error model. The vector should have names `PropPart` and `AddPart`.
- Power A numeric vector specifying the standard deviation and power parts for the power error model. The vector should have names `StdevPart` and `PowerPart`.

`BQL` A logical value indicating whether the dataset contains BQL values and they should be taken into account (M3 method).

`BQLValue` An optional numeric positive value of static LLOQ. Applicable only when BQL argument is `TRUE`. Any observed value less than or equal to that LLOQ value is treated as censored.

`Frozen` A logical value indicating if the standard deviation (Stdev) is frozen.

`ResetObs` A logical value indicating if the Observation variable should be reset to 0 after observation (`doafter={A0=0;}`). Applicable for elimination compartment.

`Covariates` A list of covariates (`Covariate` instances) that should be included in the model, but not linked to any of structural parameters. Used with "Emax" PD models ('C' covariate is added automatically when creating a new model, but should be added manually when modifying the model).

`PMLStructures` Character or character vector specifying names of PML structures in which the observation will be modified. For the naming convention of PMLStructures, see Details section of `create_ModelPK()` for PK models and `create_ModelPD()` for PD models.

Details

This function can only be used to modify the structural parameters in the built-in models (i.e., created using either `create_ModelEmax()` or `create_ModelPK()`).

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

`list_Observations()`

Functions used for Observation specification: `Observation()`, `ObservationCustom()`, `Sigmas()`, `create_ModelPD()`, `create_ModelPK()`, `remove_Observation()`

Examples

```
PMLParametersSets <-
  create_ModelPK(CompartmentsNumber = c(1, 2, 3))
# update structural paramter type
PMLParametersSetsVMod <-
 modify_Observation(
   PMLParametersSets,
   ObservationName = "CObs",
   SigmasChosen = Sigmas(Proportional = 0,
                         AdditiveMultiplicative =
                           list(PropPart = 0.1, AddPart = 10)))

print(PMLParametersSetsVMod)
```

`modify_Omega` *Modify Omega Parameters in PML Models*

Description

This function allows to modify Omega parameters in a list of PML models (`PMLModels` class instance created by `get_PMLParametersSets()`).

Usage

```
modify_Omega(
  PMLParametersSets,
  Name,
  InitialOmega,
  State,
  Frozen,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`Name`	A character string specifying the name of the Omega.
`InitialOmega`	Numeric specifying the initial value of the Omega. Default value is 1.
`State`	Character specifying the presence of the Omega. Possible values are: • `None` The Omega does not exist in the specified `PMLStructures`. • `Present` The Omega exists in the specified `PMLStructures` (the default). • `Searched` The presence of the Omega is searched.
`Frozen`	A logical value indicating whether the Omega is frozen or not.

PMLStructures Character or character vector specifying names of PML structures in which the Omega will be modified. For the naming convention of PMLStructures, see Details section of create_ModelPK() for PK models and create_ModelPD() for PD models.

Details

- If the specified Omega does not exist in the PML models, a warning will be issued, and no modifications will be made.
- The current functionality does not support modifying custom omegas (ranefs) that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Omegas()

Functions used for Omega specification: Omega(), create_ModelPD(), create_ModelPK()

Examples

```
PMLParametersSets12 <- create_ModelPK(CompartmentsNumber = c(1, 2))
# Modify an Omega parameter named "nV" with new Initial Estimate and
# Frozen flag
PMLParametersSets12Mod1 <-
  modify_Omega(PMLParametersSets12,
               Name = "nV",
               InitialOmega = 0.3,
               State = "Present",
               Frozen = TRUE,
               PMLStructures = "PK1IVC")

print(PMLParametersSets12Mod1)
```

---

modify_StParm *Modify structural parameter in PML models set*

Description

Modify structural parameter in PML models set

Usage

```
modify_StParm(
  PMLParametersSets,
  StParmName,
  Type = "LogNormal",
  State = "Present",
  ThetaStParm,
  OmegaStParm,
  Covariates,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`
: A list of PML parameters sets (`PMLModels` class instance).

`StParmName`
: Character specifying the name of the structural parameter to be modified.

`Type`
: Character specifying the type of the structural parameter. Options are

  - `LogNormal` The PML statement of the structural parameter will look like the following:

  `stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`

  - `LogNormal1` The PML statement of the structural parameter will look like the following:

  `stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))`

  - `LogNormal2` The PML statement of the structural parameter will look like the following:

  `stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))`

  - `LogitNormal` The PML statement of the structural parameter will look like the following:

  `stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`

  - `Normal` The PML statement of the structural parameter will look like the following:

  `stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))`

`State`
: character string that indicates the presence of the structural parameter. Options are:

  - `None` The structural parameter does not exist in the specified `PMLStructures`.
  - `Present` The structural parameter exists in the specified `PMLStructures` (the default).
  - `Searched` The presence of the structural parameter is searched.

`ThetaStParm`
: A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + `StParmName`.

`OmegaStParm`
: An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + `StParmName`

Covariates	A list of covariates (Covariate instances) that should be included in the structural parameter statement.
PMLStructures	Character or character vector specifying names of PML structures to which the structural parameter will be added. For the naming convention of PMLStructures, see Details section of get_PMLParametersSets().

Details

This function can only be used to modify the structural parameters in the built-in models (i.e., created using either create_ModelEmax() or create_ModelPK()) or in the custom models if they are added with add_StParm().

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

Dosepoint() list_StParms()

Functions used for StParm specification: StParm(), add_StParm(), create_ModelPD(), create_ModelPK(), modify_StParmCustom(), remove_StParm()

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(CompartmentsNumber = c(1, 2, 3))
# update structural parameter type
PMLParametersSetsVMod <-
 modify_StParm(PMLParametersSets,
            StParmName = "V",
            Type = "LogitNormal")
```

---

modify_StParmCustom *Modify custom structural parameter in PML spaces*

Description

Modify custom structural parameter in PML spaces

Usage

```
modify_StParmCustom(
  PMLParametersSets,
  StParmName,
  Type,
  State,
  ThetaStParm,
```

```
  OmegaStParm,
  Covariates,
  PMLStructures = NULL
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
StParmName	Character specifying the name of the structural parameter to be added.

Type	Character specifying the type of the structural parameter. Options are
	• LogNormal The PML statement of the structural parameter will look like the following:
	`stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`
	• LogNormal1 The PML statement of the structural parameter will look like the following:
	`stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))`
	• LogNormal2 The PML statement of the structural parameter will look like the following:
	`stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))`
	• LogitNormal The PML statement of the structural parameter will look like the following:
	`stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`
	• Normal The PML statement of the structural parameter will look like the following:
	`stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))`

State	character string that indicates the presence of the structural parameter. Options are: • None The structural parameter does not exist in the specified PMLStructures. • Present The structural parameter exists in the specified PMLStructures (the default). • Searched The presence of the structural parameter is searched.
ThetaStParm	A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + StParmName.
OmegaStParm	An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + StParmName
Covariates	A list of covariates (Covariate instances) that should be included in the structural parameter statement.
PMLStructures	Character or character vector specifying names of PML structures to which the structural parameter will be added. For the naming convention of PMLStructures, see Details section of get_PMLParametersSets().

Details

This function can be applied to the custom models. It allows modification of custom structural parameters defined in the PML code of these spaces.

When modifying a custom structural parameter, the corresponding `Stparm` statement is removed from the PML code, and the updated parameter is added back as a `StParm` class using the provided arguments. Similarly, associated `fixef` and `ranef` statements related to the custom structural parameter are removed.

Please note that this function is specifically designed for modifying custom structural parameters. For non-custom parameters, use `modify_StParm()`.

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

`Dosepoint()` `list_StParms()` `modify_StParm()`

Functions used for StParm specification: `StParm()`, `add_StParm()`, `create_ModelPD()`, `create_ModelPK()`, `modify_StParm()`, `remove_StParm()`

Examples

```
# Modify the custom structural parameter 'Cl':
OneCpt_CustomCode <-
  paste0(
    "\nderiv(A1 = - Cl * C)",
    "\ndosepoint(A1)",
    "\ndosepoint2(A1, tlag = 12)",
    "\nC = A1 / V",
    "\nerror(CEps = 0.01)",
    "\nobserve(CObs = C + CEps * sqrt(1 + C^2 * (CMultStdev/sigma())^2), bql = 0.01)",
    "\nstparm(V = tvV * exp(nV))",
    "\nstparm(Cl = tvCl * exp(nCl))",
    "\nstparm(CMultStdev = tvCMultStdev)",
    "\nfixef(tvV = c(, 5, ))",
    "\nfixef(tvCl = c(, 1, ))",
    "\nfixef(tvCMultStdev = c(, 0.1, ))",
    "\nranef(diag(nV) = c(1))",
    "\nranef(diag(nCl) = c(1))\n"
  )

OneCpt_CustomCode <-
 modify_StParmCustom(
   create_CustomSpace(OneCpt_CustomCode),
                      StParmName = "Cl",
                      Type = "Normal")
```

modify_Theta *Modify Theta Parameters in PML Models*

Description

This function allows to modify Theta parameter in a list of PML models (`PMLModels` class instance created by `create_ModelPK()` or create_ModelPD).

Usage

```
modify_Theta(
  PMLParametersSets,
  Name,
  InitialEstimates,
  Frozen,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`Name`	Character specifying the name of the Theta to be modified.
`InitialEstimates`	An `InitialEstimate()` class instance or a numerical value for the initial estimate of the Theta or a numeric vector length three with its elements representing the lower bound, initial estimate.
`Frozen`	A logical value indicating whether the Theta will be estimated or not.
`PMLStructures`	Character or character vector specifying names of PML structures in which the `Theta` parameter will be modified. For the naming convention of `PMLStructures`, see Details section of `create_ModelPK()` for PK models and `create_ModelPD()` for PD models..

Details

- If the specified Theta does not exist in the PML models, a warning will be issued, and no modifications will be made. Thetas associated with structural parameters in the proportional part of MixRatio and Additive+Proportional error models can also be modified.
- The current functionality does not support modifying custom thetas (fixefs) that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

InitialEstimate()

Functions used for Theta specification: InitialEstimate(), Theta(), create_ModelPD(), create_ModelPK()

Examples

```
PMLParametersSets <- create_ModelPK(CompartmentsNumber = c(1, 2))
# Modify a Theta parameter named "tvV" with new Initial Estimates and
# Frozen flag
PMLParametersSetsMod1 <-
  modify_Theta(PMLParametersSets,
               Name = "tvV",
               Frozen = TRUE,
               InitialEstimates = 0.3)

print(PMLParametersSetsMod1)

PMLParametersSetsMod2 <-
  add_StParm(PMLParametersSets = PMLParametersSetsMod1,
             StParmName = "Duration",
             State = "Searched",
             PMLStructures = "PK2IVC",
             DosepointArgName = "duration")

PMLParametersSetsMod3 <-
  modify_Theta(PMLParametersSets = PMLParametersSetsMod2,
               Name = "tvDuration",
               InitialEstimates = c(2, 4, Inf))

print(PMLParametersSetsMod3)
```

Observation *Create an instance of Observation class.*

Description

This function creates a new instance of Observation object and validates it.

Usage

```
Observation(
  ObservationName = "CObs",
  SigmasChosen = Sigmas(Proportional = 0.1),
  BQL = FALSE,
  BQLValue = NA,
  Frozen = FALSE,
  ResetObs = FALSE,
```

```
  Covariates = list(),
  PMLStructure = character()
)
```

Arguments

`ObservationName`
: A character string giving the name of the Observation.

`SigmasChosen`
: a Sigmas class instance or a list specifying the chosen sigma values for different error models. 0s are treated as no values. Inside Observation class it is transormed and kept as Sigmas class. The list could contain the following error models:
  - Additive The additive error sigma value.
  - LogAdditive The log-additive error sigma value.
  - Proportional The proportional error sigma value.
  - AdditiveMultiplicative A numeric vector specifying the additive and multiplicative parts for the additive-multiplicative error model. The vector should have names `PropPart` and `AddPart`.
  - MixRatio A numeric vector specifying the proportional and additive parts for the mix-ratio error model. The vector should have names `PropPart` and `AddPart`.
  - Power A numeric vector specifying the standard deviation and power parts for the power error model. The vector should have names `StdevPart` and `PowerPart`.

`BQL`
: A logical value indicating whether the dataset contains BQL values and they should be taken into account (M3 method).

`BQLValue`
: An optional numeric positive value of static LLOQ. Applicable only when BQL argument is `TRUE`. Any observed value less than or equal to that LLOQ value is treated as censored.

`Frozen`
: A logical value indicating if the standard deviation (Stdev) is frozen.

`ResetObs`
: A logical value indicating if the Observation variable should be reset to 0 after observation (`doafter={A0=0;}`). Applicable for elimination compartment.

`Covariates`
: A list of covariates (`Covariate` instances) that should be included in the model, but not linked to any of structural parameters. Used with "Emax" PD models ('C' covariate is added automatically when creating a new model, but should be added manually when modifying the model).

`PMLStructure`
: Character specifying the name of PML structure in which the observation should be added. For the naming convention of PMLStructures, see Details section of `get_PMLParametersSets()`.

Value

A new Observation object

See Also

Functions used for Observation specification: `ObservationCustom()`, `Sigmas()`, `create_ModelPD()`, `create_ModelPK()`, `modify_Observation()`, `remove_Observation()`

Examples

```
A0Obs <-
  Observation(ObservationName = "A0Obs",
              SigmasChosen = list(Additive = 2,
                                  Power = c(Stdev = 10, Power = 0.5)),
              Frozen = FALSE,
              ResetObs = TRUE,
              PMLStructure = "PK1FOC")

CObs <- Observation("CObs", Frozen = TRUE, PMLStructure = "2Cpt")
```

`ObservationCustom` *Create an instance of custom Observation class.*

Description

This function creates a new instance of custom Observation object and validates it. All PML responses are supported (`observe`, `multi`, `LL`, `event`, `count`, `ordinal`)

Usage

```
ObservationCustom(
  ObservationName = "CObs",
  Type = "observe",
  Statement = "",
  StatementNames = list(),
  Sigma = list(),
  Dobefore = c(),
  Doafter = c(),
  BQL = FALSE,
  BQLValue = NA,
  PMLStructure = character()
)
```

Arguments

`ObservationName`	A character string giving the name of the Observation.
`Type`	One of the following: `observe`, `multi`, `LL`, `event`, `count`, `ordinal`
`Statement`	A character string giving the RHS of response statement without `Type`.
`StatementNames`	A character vector giving the names of variables used in the `Statement`.
`Sigma`	a list specifying the chosen sigma value Should be given only if `Type == "observe"`
`Dobefore`	A character string specifying the sequence of operations to be performed before current observation event.

Doafter	A character string specifying the sequence of operations to be performed after current observation event.
BQL	A logical value indicating whether the dataset contains BQL values and they should be taken into account (M3 method).
BQLValue	An optional numeric positive value of static LLOQ. Applicable only when BQL argument is TRUE. Any observed value less than or equal to that LLOQ value is treated as censored.
PMLStructure	Character specifying the name of PML structure in which the observation should be added. For the naming convention of PMLStructures, see Details section of create_ModelPK().

Value

A new Observation object

See Also

Functions used for Observation specification: Observation(), Sigmas(), create_ModelPD(), create_ModelPK(), modify_Observation(), remove_Observation()

Omega *Create an Omega instance with validation*

Description

This function creates an Omega instance with the given parameters and validates it.

Usage

```
Omega(
  Name = character(),
  InitialOmega = 1,
  State = "Present",
  Frozen = FALSE,
  StParmName = character(),
  PMLStructure = character()
)
```

Arguments

Name	A character string specifying the name of the Omega.
InitialOmega	Numeric specifying the initial value of the Omega. Default value is 1.
State	Character specifying the presence of the Omega. Possible values are: • None The Omega does not exist in the specified PMLStructures. • Present The Omega exists in the specified PMLStructures (the default).

- Searched The presence of the Omega is searched.

Frozen	A logical value indicating whether the Omega is frozen or not.
StParmName	A character string specifying the corresponding structural parameter name.
PMLStructure	PML structure current omega belongs to.

Value

An Omega instance.

See Also

list_Omegas()

Functions used for Omega specification: create_ModelPD(), create_ModelPK(), modify_Omega()

Examples

```
nV <- Omega("nV")
```

---

output.CustomSpace *Output a Custom Space*

Description

This function generates the PML code representation of a custom space.

Usage

```
## S3 method for class 'CustomSpace'
output(x, ...)
```

Arguments

x	A CustomSpace object.
...	Additional arguments (not used).

Value

A character string containing the PML code.

output_NLMETemplate *Generate NLME Model Template from JSON Specification*

Description

This function serves as a command-line argument parser that calls an internal function (`output_NLMETemplateInternal`) to generate NLME (Non-Linear Mixed Effects) model templates. It takes a series of "key=value" pairs, processes a detailed JSON model specification, and outputs a model template file and a corresponding tokens file, which can be used for model fitting.

Usage

```
output_NLMETemplate(args)
```

Arguments

`args`	A character vector where each element is a string in the format "key=value". The function parses these strings to set up the model generation process. The following keys are expected: • `model_setup`: The file path to the input JSON file that defines the NLME model structure, parameters, covariates, and error models. • `template_path`: The destination file path for the output model template. • `tokens_path`: The destination file path for the output JSON tokens file. • `data_path`: The file path to the dataset that will be used with the model. • `author`: A string specifying the name of the author to be embedded in the output files. • `description`: A string providing a description of the model to be embedded in the output files.

Value

This function does not return a value to the R environment. It is executed for its side effect of writing two files:

1. A model template file to the location specified by `template_path`.
2. A tokens JSON file to the location specified by `tokens_path`.

See Also

The core logic is handled by the internal function `output_NLMETemplateInternal`. The final file writing is performed by `write_ModelTemplateTokens`.

Examples

```
## Not run:
# Example of the arguments that would be passed to the function.
# In a real scenario, these might come from a command-line call.

arguments <- c(
  "model_setup=./model_definition.json",
  "template_path=./template.txt",
  "tokens_path=./tokens.json",
  "data_path=./pk_data.csv",
  "author=Jane Doe",
  "description=2-compartment PK model with first-order absorption and linear elimination"
)

# Execute the function with the defined arguments
output_NLMETemplate(args = arguments)


## End(Not run)
```

`pyDarwinOptionsGA` *Create options for the Genetic Algorithm (GA) in pyDarwin.*

Description

This function allows you to set various options specific to the Genetic Algorithm (GA) in pyDarwin.

Usage

```
pyDarwinOptionsGA(
  elitist_num = 2,
  crossover_rate = 0.95,
  mutation_rate = 0.95,
  sharing_alpha = 0.1,
  selection = "tournament",
  selection_size = 2,
  crossover_operator = "cxOnePoint",
  mutate = "flipBit",
  attribute_mutation_probability = 0.1,
  niche_penalty = 20
)
```

Arguments

`elitist_num`	A positive integer specifying the number of best models from any generation to carry over, unchanged, to the next generation. Functions like the Hall of Fame in DEAP. Default: 2

`crossover_rate` A real value (between 0.0 and 1.0) specifying the fraction of mating pairs that will undergo crossover. Default: 0.95

`mutation_rate` A real value (between 0.0 and 1.0) specifying the probability that at least one bit in the genome will be "flipped", 0 to 1, or 1 to 0. Default: 0.95

`sharing_alpha` A real value specifying the parameter of the niche penalty calculation. Default: 0.1

`selection` A string specifying the selection algorithm for the GA. Currently, only "tournament" is available. Default: "tournament"

`selection_size` A positive integer specifying the number of "parents" to enter in the selection. 2 is highly recommended, experience with other values is very limited. Default: 2

`crossover_operator`
A string specifying the algorithm for crossover. Only "cxOnePoint" (single-point crossover) is available. Default: "cxOnePoint"

`mutate` A string specifying the algorithm for mutation. Currently, only "flipBit" is available. Default: "flipBit"

`attribute_mutation_probability`
A real value specifying the probability of any bit being mutated (real value between 0.0 and 1.0). Default: 0.1

`niche_penalty` A positive real value used for the calculation of the crowding penalty. The niche penalty is calculated by first finding the "distance matrix", the pair-wise Mikowski distance from the present model to all other models. The "crowding" quantity is then calculated as the sum of: (distance/niche_radius)^sharing_alpha for all other models in the generation for which the Mikowski distance is less than the niche radius. Finally, the penalty is calculated as: exp((crowding-1)*niche_penalty)-1. The objective of using a niche penalty is to maintain diversity of models, to avoid premature convergence of the search by penalizing when models are too similar to other models in the current generation. Default: 20

Value

An object of class "pyDarwinOptionsGA" containing the specified GA options.

Examples

```
# Create GA options with default values
options <- pyDarwinOptionsGA()

# Create GA options with custom values
options <-
  pyDarwinOptionsGA(elitist_num = 4,
                    crossover_rate = 0.9,
                    mutation_rate = 0.8,
                    sharing_alpha = 0.2)
```

pyDarwinOptionsGridAdapter *Grid Adapter Options for pyDarwin*

Description

This function creates a list of grid adapter options for pyDarwin, which are used to configure the interaction between pyDarwin and grid computing environments.

Usage

```
pyDarwinOptionsGridAdapter(
  python_path = "~/darwin/venv/bin/python",
  submit_search_command = paste("qsub -b y -cwd -o {project_stem}_out.txt",
    "-e {project_stem}_err.txt -N '{project_name}'"),
  submit_command = paste("qsub -b y -o {results_dir}/{run_name}.out",
    "-e {results_dir}/{run_name}.err -N {job_name}"),
  submit_job_id_re = "Your job (\\w+) \\(\".+?\"\\) has been submitted",
  poll_command = "qstat -s z",
  poll_job_id_re = "^\\s+(\\w+)",
  poll_interval = 10,
  delete_command = "qdel {project_stem}-*"
)
```

Arguments

python_path	Required. Path to Python interpreter, preferably to the instance of the interpreter located in the virtual environment where pyDarwin is deployed. The path must be available to all grid nodes that run jobs.
submit_search_command	Required. A command that submits a search job to the grid queue. This command is used for the entire search.
submit_command	Required. A command that submits individual runs to the grid queue. The actual command submitted to the queue is constructed by pyDarwin. It should not include `<python_path> -m darwin.run_model`.
submit_job_id_re	Required. A regular expression pattern to extract the job ID after submission. The job ID must be captured with the first capturing group.
poll_command	Required. A command that retrieves finished jobs from the grid controller. If the controller/setup allows to specify ids/patterns in polling commands, do it. Otherwise, all finished jobs should be polled using commands `qstat -s z`.
poll_job_id_re	Required. A regular expression pattern to find a job ID in every line of the poll_command output. Similar to submit_job_id_re.
poll_interval	Optional. How often to poll jobs (in seconds). Default is 10 seconds.
delete_command	Optional. A command that deletes all unfinished jobs related to the search when you stop it. It may delete all of them by ID (e.g., `qdel {job_ids}`) or by mask (e.g., `qdel {project_stem}-*`)..

Value

A list containing the configured grid adapter options.

Examples

```
grid_options <- pyDarwinOptionsGridAdapter(
  python_path = "~/darwin/venv/bin/python",
  submit_search_command =
   "qsub -b y -cwd -o {project_stem}_out.txt -e {project_stem}_err.txt -N '{project_name}'",
  submit_command =
   "qsub -b y -o {results_dir}/{run_name}.out -e {results_dir}/{run_name}.err -N {job_name}",
  submit_job_id_re = "Your job (\\w+) \\(\".+?\"\\) has been submitted",
  poll_command = "qstat -s z",
  poll_job_id_re = "^\\s+(\\w+)",
  poll_interval = 10,
  delete_command = "qdel {project_stem}-*"
)
```

---

pyDarwinOptionsMOGA *Create Options for the pyDarwin MOGA Block*

Description

Generates a list of specific options for the MOGA (Multi-Objective Genetic Algorithm) or MOGA3 (NSGA-III) algorithms in pyDarwin. This list is intended to be passed as the `MOGA` argument to the `create_pyDarwinOptions()` function when `algorithm` is set to "MOGA" or "MOGA3".

Usage

```
pyDarwinOptionsMOGA(
  objectives = 3,
  names = NULL,
  constraints = 0,
  partitions = 12,
  crossover = "single",
  crossover_rate = 0.95,
  mutation_rate = 0.95,
  attribute_mutation_probability = 0.1
)
```

Arguments

`objectives`	Positive integer: Number of objectives. Applicable only when the `algorithm` in `create_pyDarwinOptions()` is set to "MOGA3". If the `algorithm` is "MOGA" (implying NSGA-II), this parameter is ignored by pyDarwin, and 2 objectives (OFV and NEP) are used internally. For "MOGA3", objectives are defined by postprocessing. Default: 3 (relevant for "MOGA3").

`names` Character vector (optional): List of names for the objectives. Applicable only when the `algorithm` is "MOGA3". The length of this vector should match the `objectives` value. If `NULL`, empty, or of a different size, pyDarwin uses generic names (e.g., "f1", "f2", "f3"). These names are used for reporting in `results.csv`. Ignored if the `algorithm` is "MOGA". Default: NULL.

`constraints` Non-negative integer: Number of constraints. Applicable only when the `algorithm` is "MOGA3". Constraints must be provided by user-defined postprocessing scripts (R or Python). See https://pymoo.org/constraints/index.html. Ignored if the `algorithm` is "MOGA". Default: 0.

`partitions` Positive integer: Number of partitions for the reference directions used in NSGA-III. Applicable only when the `algorithm` is "MOGA3". See https://pymoo.org/misc/reference_directions.html. Ignored if the `algorithm` is "MOGA". Default: 12.

`crossover` Character string: Crossover algorithm. When set to "single", SinglePointCrossover is used. Otherwise, for other values like "two_point", TwoPointCrossover is typically used by pymoo. See https://pymoo.org/operators/crossover.html#Point-Crossover. Applicable for both "MOGA" and "MOGA3". Default: "single".

`crossover_rate` Numeric value between 0.0 and 1.0: The fraction of mating pairs that will undergo crossover. Applicable for both "MOGA" and "MOGA3". Default: 0.95.

`mutation_rate` Numeric value between 0.0 and 1.0: The probability that at least one bit in the genome will be "flipped" (mutated). Applicable for both "MOGA" and "MOGA3". Default: 0.95.

`attribute_mutation_probability` Numeric value between 0.0 and 1.0: The probability of any individual bit (attribute) in the genome being mutated. Applicable for both "MOGA" and "MOGA3". Default: 0.1.

Details

This function defines parameters for the `MOGA` settings block in pyDarwin. The relevance of certain parameters (`objectives`, `names`, `constraints`, `partitions`) depends on whether the `algorithm` in `create_pyDarwinOptions()` is set to "MOGA" (for NSGA-II) or "MOGA3" (for NSGA-III).

- If `algorithm = "MOGA"` (NSGA-II): pyDarwin internally uses 2 objectives (OFV and NEP). The `objectives`, `names`, `constraints`, and `partitions` parameters from this MOGA options block are ignored by pyDarwin. Postprocessing scripts are not used by pyDarwin to define objectives.
- If `algorithm = "MOGA3"` (NSGA-III): The `objectives`, `names`, `constraints`, and `partitions` parameters from this MOGA options block are utilized by pyDarwin. User-supplied postprocessing scripts (R or Python) are **required** to calculate and return the values for the objectives and constraints. (R: list of two vectors; Python: tuple of two lists).

Common parameters like `crossover`, `crossover_rate`, `mutation_rate`, and `attribute_mutation_probability` apply to both "MOGA" and "MOGA3" variants.

Value

A list containing MOGA-specific options.

Examples

```
# MOGA options, defaults are generally suitable for algorithm = "MOGA3"
# if postprocessing handles 3 objectives.
moga_block_for_moga3 <- pyDarwinOptionsMOGA(
  objectives = 3,
  names = c("Objective1", "Objective2", "Objective3"),
  constraints = 1,
  partitions = 10
)

# MOGA options where specific settings are for NSGA-II (algorithm = "MOGA")
# Note: objectives, names, constraints, partitions would be ignored by pyDarwin.
moga_block_for_moga <- pyDarwinOptionsMOGA(
  crossover = "two_point",
  crossover_rate = 0.92
)
```

---

pyDarwinOptionsPenalty

Create pyDarwin Penalty Options

Description

Generates a list of penalty parameters to be used in pyDarwin create_pyDarwinOptions function.

Usage

```
pyDarwinOptionsPenalty(
  theta = 10,
  omega = 10,
  sigma = 10,
  convergence = 100,
  covariance = 100,
  correlation = 100,
  condition_number = 100,
  non_influential_tokens = 1e-05
)
```

Arguments

theta	numeric: Penalty added to fitness/reward for each estimated THETA. A value of 3.84 corresponds to a hypothesis test with 1 df and $p < 0.05$ (for nested models), and a value of 2 for 1 df corresponds to the Akaike information criterion. Default: 10
omega	numeric: Penalty added to fitness/reward for each estimated OMEGA element. Default: 10

`sigma` numeric: Penalty added to fitness/reward for each estimated SIGMA element. Default: 10

`convergence` numeric: Penalty added to fitness/reward for failing to converge. Default: 100

`covariance` numeric: Penalty added to fitness/reward for failing the covariance step (real number). If a successful covariance step is important, this can be set to a large value (e.g., 100), otherwise, set to 0. Default: 100

`correlation` numeric: Penalty added to fitness/reward if any off-diagonal element of the correlation matrix of estimates has an absolute value > 0.95 (real number). This penalty will be added if the covariance step fails or is not requested. Default: 100

`condition_number`
numeric: Penalty added if the covariance step fails or is not requested, e.g., PRINT=E is not included in $COV. Additionally, if the covariance is successful and the condition number of the covariance matrix is > 1000, then this penalty is added to the fitness/reward. Default: 100

`non_influential_tokens`
numeric: Penalty added to fitness/reward if any tokens do not influence the control file (relevant for nested tokens). Should be very small (e.g., 0.0001), as the purpose is only for the model with non-influential tokens to be slightly worse than the same model without the non-influential token(s) to break a tie. Default: 0.00001

Value

A list of penalty options in pyDarwin optimization process.

Examples

```
# Create penalty options with default values
penalty_options <- pyDarwinOptionsPenalty()
# Create penalty options with custom values
penalty_options_custom <-
  pyDarwinOptionsPenalty(theta = 3.84,
                         omega = 8,
                         sigma = 6,
                         convergence = 50,
                         covariance = 80,
                         correlation = 60,
                         condition_number = 70,
                         non_influential_tokens = 0.0001)
```

`pyDarwinOptionsPostprocess`
Create pyDarwin Postprocess Options

Description

Generates a list of postprocessing options to be used in pyDarwin optimization process.

Usage

```
pyDarwinOptionsPostprocess(
  use_r = FALSE,
  post_run_r_code = "{project_dir}/simplefunc.R",
  r_timeout = 30,
  use_python = FALSE,
  post_run_python_code = "{project_dir}/simplefunc.py"
)
```

Arguments

use_r	Logical: Whether to use R for postprocessing. If set to TRUE, R will be used to execute the post-processing script specified in post_run_r_code. Default: FALSE.
post_run_r_code	Character: The file path to the R script that contains post-processing code. This script will be executed after the pyDarwin optimization process finishes. For NSGA-III (MOGA with 3 objectives), the R script must return a list containing two vectors: the first for objectives, the second for constraints (empty vector if no constraints). For other cases, it should return a vector containing a penalty value and a text string. Default: "{project_dir}/simplefunc.R".
r_timeout	Numeric: The time limit (in seconds) for the execution of the post-processing R script. If the R script takes longer to execute than this timeout value, it will be terminated. Default: 30.
use_python	Logical: Whether to use Python for postprocessing. If set to TRUE, Python will be used to execute the post-processing script specified in post_run_python_code. Default: FALSE.
post_run_python_code	Character: The file path to the Python script that contains post-processing code. The script must contain a function post_process(run_dir_path) or post_process2(model_run_obje For NSGA-III (MOGA with 3 objectives), this function must return a tuple of two lists: the first for objectives, the second for constraints (empty list if no constraints). For other cases, it should return a tuple containing a penalty value and a text string. Default: "{project_dir}/simplefunc.py".

Value

A list of postprocessing options in pyDarwin optimization process.

Examples

```
# Create postprocess options with default values
postprocess_options <- pyDarwinOptionsPostprocess()
```

```
# Create postprocess options with custom values
postprocess_options_custom <-
  pyDarwinOptionsPostprocess(use_r = TRUE,
                             post_run_r_code = "{project_dir}/postprocess.R",
                             r_timeout = 60,
                             use_python = TRUE,
                             post_run_python_code = "{project_dir}/postprocess.py")
```

`pyDarwinOptionsPSO` *Create options for the Particle Swarm Optimization (PSO) in pyDarwin.*

Description

This function allows you to set various options specific to the Particle Swarm Optimization (PSO) in pyDarwin.

Usage

```
pyDarwinOptionsPSO(
  inertia = 0.4,
  cognitive = 0.5,
  social = 0.5,
  neighbor_num = 20,
  p_norm = 2,
  break_on_no_change = 5
)
```

Arguments

`inertia`	A real value specifying the particle coordination movement as it relates to the previous velocity (commonly denoted as w). Default: 0.4
`cognitive`	A real value specifying the particle coordination movement as it relates to its own best known position (commonly denoted as c1). Default: 0.5
`social`	A real value specifying the particle coordination movement as it relates to the current best known position across all particles (commonly denoted as c2). Default: 0.5
`neighbor_num`	A positive integer specifying the number of neighbors that any particle interacts with to determine the social component of the velocity of the next step. A smaller number of neighbors results in a more thorough search (as the neighborhoods tend to move more independently, allowing the swarm to cover a larger section of the total search space) but will converge more slowly. Default: 20
`p_norm`	A positive integer specifying the Minkowski p-norm to use. A value of 1 is the sum-of-absolute values (or L1 distance) while 2 is the Euclidean (or L2) distance. Default: 2
`break_on_no_change`	A positive integer specifying the number of iterations used to determine whether the optimization has converged. Default: 5

Value

An object containing the specified options for the Particle Swarm Optimization (PSO) algorithm.

Examples

```
# Create PSO options with default values
options <- pyDarwinOptionsPSO()

# Create PSO options with custom values
options <- pyDarwinOptionsPSO(inertia = 0.2,
                              cognitive = 0.8,
                              social = 0.7,
                              neighbor_num = 10)
```

---

reconnect_pyDarwinJob *Reconnect, Monitor, and Retrieve Results from a Remote pyDarwin Job*

Description

This function reconnects to a pyDarwin job previously launched in the background on a remote host. It monitors the job until completion (if a PID is available), then downloads and processes the results.

Usage

```
reconnect_pyDarwinJob(
  LocalDirectoryPath = ".",
  LocalJobInfoFilePath = NULL,
  OriginalOptionsPath = NULL,
  Password = NULL,
  KeyPath = NULL,
  MonitoringInterval = 30,
  verbose = getOption("verbose", default = FALSE)
)
```

Arguments

`LocalDirectoryPath`

Character string: The base local directory associated with the pyDarwin job. This directory is used to: 1. Locate the job information file (if `LocalJobInfoFilePath` is NULL), expected as `{ProjectName}_remote_job_info.json`. 2. Locate the original `options.json` file (if `OriginalOptionsPath` is NULL). 3. Serve as the base location for downloading results into a subdirectory (e.g., `{LocalDirectoryPath}/{ProjectNar`

`LocalJobInfoFilePath`
: Character string (optional): Explicit path to the local JSON file containing information about the remote job (e.g., as created by `RunPyDarwinRemote(Wait = FALSE)`). If NULL (default), the path is constructed using `LocalDirectoryPath` and `ProjectName` (derived from `OriginalOptionsPath`).

`OriginalOptionsPath`
: Character string (optional): Explicit path to the original local `options.json` file that was used when the job was first launched. This is needed to correctly parse results (e.g., `engine_adapter`) and to derive `ProjectName` if not available from `LocalJobInfoFilePath`. If NULL (default), the function attempts to find `options.json` within `LocalDirectoryPath`. If not found, the operation will stop.

`Password`
: Character string. The password for SSH authentication. Defaults to `""`, which is appropriate when using key-based authentication. Using keys is strongly recommended over embedding passwords in scripts.

`KeyPath`
: Character string. The path to your private SSH key file. Defaults to the path stored in the `SSH_PRIVATE_KEY_PATH` environment variable.

`MonitoringInterval`
: Numeric. The interval in seconds between status checks when monitoring a running job (`Wait = TRUE`).

`verbose`
: Logical: Passed to helper functions for verbose output during SSH connection and file downloads. Default: `getOption("verbose", default = FALSE)`.

Details

This function requires a job information JSON file (typically created by `RunPyDarwinRemote` when `Wait = FALSE`) to obtain details like the remote host, user, remote project directory, and optionally the remote process ID (PID).

The `ProjectName` is crucial. It's primarily derived from the `project_name` field in the original options file (located via `OriginalOptionsPath` or within `LocalDirectoryPath`). If not present in the options file, a fallback derivation uses the parent directory name of the options file. This `ProjectName` is then used to find the job info file (as `{ProjectName}_remote_job_info.json` in `LocalDirectoryPath`) if `LocalJobInfoFilePath` is not directly provided. If the job info file itself contains a `ProjectName`, that value may take precedence. Downloaded results are organized locally using this determined `ProjectName`.

If `RemoteJobPID` is available in the job info file, the function will actively monitor the process. If the PID is not available, it will skip active monitoring and proceed directly to attempt downloading any available results.

Value

A list containing parsed results similar to `RunPyDarwinRemote(Wait = TRUE)` (i.e., `results` data.frame, `FinalResultFile`, `FinalControlFile`, `DownloadedResultsDir`, `DownloadedItems`), or the content of the downloaded `messages.txt` as a character vector if primary result files are not found or parsed successfully. If essential information (like job info or options file) is missing, the function will stop.

See Also

run_pyDarwin(), run_pyDarwinRemote()

Examples

```
## Not run:
# Assuming 'my_project_job_info.json' and 'options.json' exist in '~/darwin_runs/my_project_run'
# and 'my_project_job_info.json' was created by a previous RunPyDarwinRemote(Wait=FALSE) call.

# Example 1: Specifying only the local directory path
# ProjectName will be derived from options.json within that path.
# Job info file will be sought as {ProjectName}_remote_job_info.json.
try({
  reconnect_pyDarwinJob(
    LocalDirectoryPath = "~/darwin_runs/my_project_run"
  )
})

# Example 2: Specifying paths explicitly
try({
  reconnect_pyDarwinJob(
    LocalDirectoryPath = "~/darwin_runs/my_project_run", # Still used for downloads
   LocalJobInfoFilePath = "~/darwin_runs/my_project_run/my_project_remote_job_info.json",
    OriginalOptionsPath = "~/darwin_runs/my_project_run/options.json",
    KeyPath = "~/.ssh/id_rsa_remote_server"
  )
})

## End(Not run)
```

---

remove_Covariate *Remove Covariate from PML models*

Description

Remove Covariate from PML models

Usage

```
remove_Covariate(
  PMLParametersSets,
  Name,
  StParmNames = NULL,
  PMLStructures = NULL
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
Name	Character specifying the name of the covariate to be removed.
StParmNames	Character or character vector specifying names of structural parameters from which the covariate will be removed. Can be set to NULL or not specified, for such case the covariate will be removed from all structural parameters.
PMLStructures	Character or character vector specifying names of PML structures from which the covariate will be removed. For the naming convention of PMLStructures, see Details section of see details section of get_PMLParametersSets().

Details

The current functionality does not support removing custom covariates that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Covariates()

Functions used for Covariate specification: Covariate(), add_Covariate(), create_ModelPD(), create_ModelPK()

Examples

```
PMLParametersSets <- get_PMLParametersSets()

PMLParametersSetsWT <-
 add_Covariate(PMLParametersSets,
               Name = "WT",
               Type = "Continuous",
               State = "Present",
               Direction = "Forward",
               Center = 70)

PMLParametersSetsVonly <-
 remove_Covariate(PMLParametersSets = PMLParametersSetsWT,
               Name = "WT",
               StParmNames = "Cl")
```

`remove_Observation` *Remove Observation from PML models*

Description

Remove Observation from PML models

Usage

```
remove_Observation(PMLParametersSets, ObservationName, PMLStructures = NULL)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`ObservationName`	A character string giving the name of the Observation.
`PMLStructures`	Character or character vector specifying names of PML structures from which the observation will be removed. For the naming convention of PMLStructures, see Details section of `create_ModelPK()` for PK models and `create_ModelPD()` for PD models.

Details

The current functionality does not support modifying custom observations that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

`list_Observations()`

Functions used for Observation specification: `Observation()`, `ObservationCustom()`, `Sigmas()`, `create_ModelPD()`, `create_ModelPK()`, `modify_Observation()`

Examples

```
PMLParametersSets <-
  create_ModelPK(
    CompartmentsNumber = c(2, 3),
    Parameterization = "Micro",
    Absorption = c("First-Order", "Gamma"),
    ByVector = TRUE,
    ClosedForm = TRUE,
    EliminationCpt = TRUE)
```

```
remove_Observation(PMLParametersSets,
                   ObservationName = "A0Obs",
                   PMLStructures = "PK3GME")
```

`remove_StParm` *Remove structural parameter from PML models*

Description

Remove structural parameter from PML models

Usage

```
remove_StParm(PMLParametersSets, StParmName, PMLStructures = NULL)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`StParmName`	character specifying the name for the structural parameter to be removed.
`PMLStructures`	Character or character vector specifying names of PML structures from which the structural parameter will be removed. For the naming convention of PML-Structures, see Details section of `get_PMLParametersSets()`.

Details

Please make sure that structural parameter to be removed is not essential for the model. Usually the user does not need to remove any structural parameter. The only case is related to structural parameters in `Dosepoint()`.

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

`Dosepoint()` `list_StParms()`

Functions used for StParm specification: `StParm()`, `add_StParm()`, `create_ModelPD()`, `create_ModelPK()`, `modify_StParm()`, `modify_StParmCustom()`

Examples

```
PMLParametersSets <- get_PMLParametersSets(CompartmentsNumber = c(1, 2))

PMLParametersSetsDuration <-
 add_StParm(PMLParametersSets,
            StParmName = "Duration",
            State = "Searched",
            DosepointArgName = "duration")

PMLParametersSetsDuration1CptOnly <-
 remove_StParm(PMLParametersSetsDuration,
               StParmName = "Duration",
               PMLStructures = "PK2IVC")
```

`run_pyDarwin` *Run pyDarwin Model Search*

Description

This function executes a pyDarwin model search by calling the specified Python interpreter and the `darwin.run_search` module.

Usage

```
run_pyDarwin(
  InterpreterPath,
  Flags = c("-u", "-m"),
  DirectoryPath = ".",
  TemplatePath = "template.txt",
  TokensPath = "tokens.json",
  OptionsPath = "options.json",
  Wait = TRUE
)
```

Arguments

`InterpreterPath`	Character string. The full path to the Python interpreter executable (e.g., `python.exe` or `python`).
`Flags`	Character vector. Optional flags passed directly to the Python interpreter. Defaults to `c("-u", "-m")`. `-u` forces unbuffered binary stdout and stderr streams. `-m` runs a library module as a script and is essential for calling `darwin.run_search`.
`DirectoryPath`	Character string. Optional path to the directory containing the `template.txt`, `tokens.json`, and `options.json` files. If provided, this path is used to locate these files, overriding any directory information in the `TemplatePath`, `TokensPath`, and `OptionsPath` arguments (a warning will be issued). Defaults to the current R working directory.

TemplatePath	Character string. Path to the pyDarwin template file (typically template.txt). If DirectoryPath is specified, only the basename of TemplatePath is used, combined with DirectoryPath.
TokensPath	Character string. Path to the pyDarwin tokens JSON file (typically tokens.json). If DirectoryPath is specified, only the basename of TokensPath is used, combined with DirectoryPath.
OptionsPath	Character string. Path to the pyDarwin options JSON file (typically options.json). This file defines run settings like working_dir, output_dir, etc. If DirectoryPath is specified, only the basename of OptionsPath is used, combined with DirectoryPath.
Wait	Logical. If TRUE (default), R waits for the pyDarwin process to complete before proceeding. If FALSE, R launches the pyDarwin process in the background and returns immediately. See the 'Background Execution' section for important details when using Wait = FALSE.

Value

- If Wait = TRUE: A list containing the search results read from the output_dir (specified in options.json). This typically includes:
  - results: A data frame (results.csv).
  - FinalResultFile: Character vector containing lines from the final model's result file (e.g., .lst, .txt).
  - FinalControlFile: Character vector containing lines from the final model's control file (e.g., .mod, .mmdl).

  If result files are not found, warnings are issued. If no results are found but messages.txt exists, its content might be returned with a warning. If the Python call fails (non-zero exit code), the function stops with an error.
- If Wait = FALSE: A character string giving the full path to the main pyDarwin log file (messages.txt) within the working_dir.

Background Execution (Wait = FALSE)

When Wait is set to FALSE, the pyDarwin process is launched in the background, and the R function returns immediately. This allows R to continue processing while pyDarwin runs.

- **Output Redirection:** Standard output (stdout) and standard error (stderr) from the Python process are redirected to files named stdout.log and stderr.log respectively, within the working_dir. These files are crucial for diagnosing issues if the background process fails or behaves unexpectedly.
- **Primary Log:** The main pyDarwin log file (messages.txt, located in the working_dir) remains the primary source for detailed run information, but the stdout.log and stderr.log capture console output and errors directly.

See Also

run_pyDarwinRemote(), reconnect_pyDarwinJob()

Examples

```
## Not run:
# Example: Running pyDarwin and waiting for results
# Assuming python is in the PATH and input files are in 'my_project'

results <- run_pyDarwin(
  InterpreterPath = "python",
  DirectoryPath = "my_project",
  Wait = TRUE
)
print(results$results)

# Example: Launching pyDarwin in the background

log_file_path <- run_pyDarwin(
  InterpreterPath = "python",
  DirectoryPath = "my_project",
  Wait = FALSE
)

## End(Not run)
```

---

run_pyDarwinRemote *Run pyDarwin on a Remote Host via SSH*

Description

Establishes an SSH connection, prepares and uploads project files, executes pyDarwin in the background, and can optionally monitor the job and download results upon completion.

Usage

```
run_pyDarwinRemote(
  Host,
  User,
  Password = "",
  KeyPath = Sys.getenv("SSH_PRIVATE_KEY_PATH"),
  SshFlags = character(0),
  LocalTemplatePath,
  LocalTokensPath,
  LocalOptionsPath,
  LocalDirectoryPath = ".",
  RemoteBaseDir = "~/.rdarwin/",
  RemoteInterpreterPath = NULL,
  UseLocalLicense = FALSE,
  Wait = TRUE,
  Flags = c("-u", "-m"),
```

```
  MonitoringInterval = 30
)
```

Arguments

Host	Character string. The hostname or IP address of the remote server.
User	Character string. The username for the SSH connection.
Password	Character string. The password for SSH authentication. Defaults to "", which is appropriate when using key-based authentication. Using keys is strongly recommended over embedding passwords in scripts.
KeyPath	Character string. The path to your private SSH key file. Defaults to the path stored in the SSH_PRIVATE_KEY_PATH environment variable.
SshFlags	Character vector. Additional flags to pass to the underlying ssh::ssh_connect function.
LocalTemplatePath	Character string. The path to the pyDarwin template file. If not provided, defaults to "template.txt" within LocalDirectoryPath.
LocalTokensPath	Character string. The path to the pyDarwin tokens JSON file. If not provided, defaults to "tokens.json" within LocalDirectoryPath.
LocalOptionsPath	Character string. The path to the pyDarwin options JSON file. If not provided, defaults to "options.json" within LocalDirectoryPath.
LocalDirectoryPath	Character string or NULL. The path to the local project directory that contains the pyDarwin input files. Defaults to the current working directory (.). If NULL is provided, the directory containing LocalOptionsPath is used as the project directory.
RemoteBaseDir	Character string. The base directory on the remote host under which a new project-specific directory will be created.
RemoteInterpreterPath	Character string or NULL. The full path to the Python interpreter on the remote host (e.g., /usr/bin/python3). If NULL, the function attempts to find a suitable Python interpreter automatically.
UseLocalLicense	Logical. If TRUE, attempts to transfer local Certara license files to the remote host.
Wait	Logical. If TRUE (the default), the function will monitor the remote job's progress and download the results upon completion. If FALSE,
Flags	Character vector. Command-line flags to pass to the pyDarwin Python module.
MonitoringInterval	Numeric. The interval in seconds between status checks when monitoring a running job (Wait = TRUE).

Details

This function automates the entire remote execution workflow. It creates a unique project directory on the remote host to ensure run isolation.

Value

The return value depends on the `Wait` parameter:

`If Wait = TRUE` On successful completion, a list containing the parsed results, similar to `run_pyDarwin()`. This may include data frames like `$results` and character vectors like `$FinalResultFile`.

`If Wait = FALSE` An invisible list containing information needed to reconnect to the job later using `reconnect_pyDarwinJob()`. The list includes:

- `LocalJobInfoFile`: Path to the local JSON file with job details.
- `RemoteProjectDir`: The directory on the remote host.
- `RemoteJobPID`: The Process ID of the job on the remote host.
- `Host`, `User`, `ProjectName`

The function throws an error if a critical step fails.

See Also

`create_pyDarwinOptions()`, `reconnect_pyDarwinJob()`

Examples

```
## Not run:
# Example of launching a remote job and waiting for the results
remote_results <- run_pyDarwinRemote(
  Host = "cluster.mycompany.com",
  User = "myuser",
  KeyPath = "~/.ssh/id_rsa_cluster",
  LocalDirectoryPath = "path/to/my/CovariateSearchProject"
)

# Example of launching a job in the background
job_info <- run_pyDarwinRemote(
  Host = "cluster.mycompany.com",
  User = "myuser",
  KeyPath = "~/.ssh/id_rsa_cluster",
  LocalDirectoryPath = "path/to/my/CovariateSearchProject",
  Wait = FALSE
)

# You can later use job_info to reconnect to the job
# final_results <- reconnect_pyDarwinJob(JobInfo = job_info)

## End(Not run)
```

`Sigmas` *Create an instance of Sigmas class.*

Description

This function creates a new instance of different error models object to be applied. 0s are treated as no values.

Usage

```
Sigmas(
  Additive = 0,
  LogAdditive = 0,
  Proportional = 0.1,
  AdditiveMultiplicative = list(PropPart = 0, AddPart = 0),
  MixRatio = list(PropPart = 0, AddPart = 0),
  Power = list(PowerPart = 0, StdevPart = 0),
  ObservationName = ""
)
```

Arguments

`Additive`	The additive error sigma value.
`LogAdditive`	The log-additive error sigma value.
`Proportional`	The proportional error sigma value.
`AdditiveMultiplicative`	A list specifying the additive and multiplicative parts for the additive-multiplicative error model. The list should have elements `PropPart` and `AddPart`. Alternatively the proportional part (`PropPart`) could be presented as `StParm`, see `StParm()`.
`MixRatio`	A list specifying the proportional and additive parts for the mix-ratio error model. The list should have elements `PropPart` and `AddPart`. Alternatively the proportional part (`PropPart`) could be presented as `StParm`, see `StParm()`.
`Power`	A numeric vector specifying the standard deviation and power parts for the power error model. The vector should have names `StdevPart` and `PowerPart`.
`ObservationName`	A character string giving the name of the Observation.

Value

A Sigmas class instance.

See Also

Functions used for Observation specification: `Observation()`, `ObservationCustom()`, `create_ModelPD()`, `create_ModelPK()`, `modify_Observation()`, `remove_Observation()`

Examples

```
RSE_CObs <-
  Observation(SigmasChosen =
    Sigmas(MixRatio = list(PropPart = 2,
                           AddPart = 0.01),
           Proportional = 0))
models <-
  create_ModelPK(CompartmentsNumber = 2,
                 CObs = RSE_CObs)
print(models)
```

`specify_EngineParams` *Specify Engine Parameters for NLME Model Execution*

Description

Defines optional engine parameters to control the estimation or simulation process in Phoenix NLME. This function generates a **single character string** containing space-separated `name=value` pairs for non-default settings. Parameters are included in the output string according to the order in the function signature.

Usage

```
specify_EngineParams(
  sort = FALSE,
  ODE = c("MatrixExponent", "DVERK", "DOPRI5", "AutoDetect", "Stiff", "LSODE"),
  rtolODE = 1e-06,
  atolODE = 1e-06,
  maxStepsODE = 50000L,
  numIterations = 1000L,
 method = c("FOCE-ELS", "QRPEM", "Laplacian", "Naive-Pooled", "FOCE-LB", "IT2S-EM",
    "FO"),
  stdErr = c("Sandwich", "Hessian", "Fisher-Score", "Auto-Detect", "None"),
  isCentralDiffStdErr = TRUE,
  stepSizeStdErr = 0.01,
  logTransform = NULL,
  numIntegratePtsAGQ = 1L,
  numIterNonParametric = 0L,
  fastOptimization = FALSE,
  numIterMAPNP = 0L,
  numRepPCWRES = 0L,
  stepSizeLinearize = 0.002,
  numDigitLaplacian = 7L,
  numDigitBlup = 13L,
  gradTolOuter = 2e-04,
  stepTolOuter = 1e-04,
```

```
  gradTolInner = 1.71e-05,
  stepTolInner = 7.07e-08,
  refDeltaLagl = 0.001,
  mapAssist = 0L,
  iSample = 300L,
  iAcceptRatio = 0.1,
 impDist = c("Normal", "DoubleExponential", "Direct", "T", "Mixture-2", "Mixture-3"),
  tDOF = 4L,
  numSampleSIR = 10L,
  numBurnIn = 0L,
  freezeOmega = FALSE,
  MCPEM = FALSE,
  runAllIterations = FALSE,
  scramble = c("Owen", "Tezuka-Faur", "None"),
  emTolType = 0L,
  emConvLen = 10L,
  emConvCritVal = 5,
  stepSizePartialDeriv = 1e-05,
  numTimeStepPartialDeriv = 20L
)
```

Arguments

`sort`	Logical; Specifies whether to sort the input data by subject and time. **Default:** `FALSE`. (Note: NLME/RsNLME may default to `TRUE` if model has no reset info). Included in output only if set to `TRUE`.
`ODE`	Character; Specifies the ODE solver. Options: `"MatrixExponent"`, `"DVERK"`, `"DOPRI5"`, `"AutoDetect"`, `"Stiff"`, `"LSODE"`. **Default:** `"MatrixExponent"`. `"AutoDetect"` and `"LSODE"` use LSODA. `"Stiff"` is LSODE configured for stiff systems.
`rtolODE`	Numeric; Relative tolerance for the ODE solver. **Default:** `1e-6`. *(Not applicable if* `ODE = "MatrixExponent"`*).*
`atolODE`	Numeric; Absolute tolerance for the ODE solver. **Default:** `1e-6`. *(Not applicable if* `ODE = "MatrixExponent"`*).*
`maxStepsODE`	Integer; Maximum number of steps for the ODE solver. **Default:** `50000L`. *(Not applicable if* `ODE = "MatrixExponent"`*).*
`numIterations`	Integer; Maximum number of estimation iterations (max: 10000). **Default:** `1000L`.
`method`	Character; Estimation method. Options: `"FOCE-ELS"`, `"QRPEM"`, `"Laplacian"`, `"Naive-Pooled"`, `"FOCE-LB"`, `"IT2S-EM"`, `"FO"`. **Default:** `"FOCE-ELS"`. (Note: NLME/RsNLME default depends on model type). Only `"Naive-Pooled"` is valid for individual models.
`stdErr`	Character; Standard error computation method. Options: `"Sandwich"`, `"Hessian"`, `"Fisher-Score"`, `"Auto-Detect"`, `"None"`. **Default:** `"Sandwich"` (for most methods), `"Fisher-Score"` (if `method="QRPEM"`), `"None"` (if `method="IT2S-EM"`). The default applied depends on the chosen `method`. See `Certara.RsNLME::engineParams` documentation for full details.

`isCentralDiffStdErr`
: Logical; Use central (`TRUE`) or forward (`FALSE`) difference for numerical standard error calculations. **Default:** `TRUE`.

`stepSizeStdErr`
: Numeric; Relative step size for numerical Hessian computation for standard errors. Must be positive. **Default:** `0.01`. (Note: NLME/RsNLME default differs for Naive-Pooled method).

`logTransform`
: Logical or NULL; Controls log-transformation behavior for models with log-additive residual error (e.g., C*exp(epsilon)). **Default:** `NULL`. If set to `TRUE` or `FALSE` (i.e., not `NULL`), the `logTransform=VALUE` pair is included in the output string. The NLME engine interprets this parameter based on the model structure.

  - For models with a **single log-additive residual error**:
    - `NULL` or `TRUE`: Enables fitting in the log-domain (LTBS approach).
    - `FALSE`: The log-additive error is treated as proportional during fitting.
  - For models with **multiple residual errors where at least one is log-additive**: The log-additive error(s) are treated as proportional during fitting, regardless of the `logTransform` value.
  - For models **without log-additive residual errors**: This setting is generally ignored by the engine concerning special log-additive handling.

  Note: This function includes `logTransform` in the output string if it's not `NULL`. The ultimate applicability and interpretation are handled by the NLME engine based on the detailed model structure.

`numIntegratePtsAGQ`
: Integer; Number of quadrature points per dimension for Adaptive Gaussian Quadrature (AGQ). 1 means no AGQ. >1 enables AGQ. **Default:** `1L`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`numIterNonParametric`
: Integer; Controls non-parametric (NP) optimization. 0: Disable. 1: NONMEM-style NP. >1: Evolutionary NP algorithm generations. **Default:** `0L`. *(Population models,* `method != "Naive-Pooled"` *only).*

`fastOptimization`
: Logical; Use Automatic Differentiation (`TRUE`) or Finite Difference (`FALSE`) for optimizing random effects (etas). **Default:** `FALSE`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`numIterMAPNP`
: Integer; Number of preliminary Naive-Pooled iterations. **Default:** `0L`. *(Population models,* `method != "Naive-Pooled"` *only).*

`numRepPCWRES`
: Integer; Replicates for PCWRES (max 10000). 0 disables calculation. **Default:** `0L`. *(Population models,* `method != "Naive-Pooled"` *only).*

`stepSizeLinearize`
: Numeric; Relative step size for numerical differentiation during linearization. Must be positive. **Default:** `0.002`. *(Population models? Check NLME docs).*

`numDigitLaplacian`
: Integer; Optimization accuracy (NDIGIT) for the outer loop (`"FOCE-ELS"`/`"Laplacian"`). Positive integer. **Default:** `7L`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`numDigitBlup`
: Integer; Optimization accuracy (NDIGIT) for the inner loop (etas) or for `"Naive-Pooled"`. Positive integer. **Default:** `13L`. *(Population models or* `method = "Naive-Pooled"`*).*

`gradTolOuter` Numeric; Max gradient tolerance, outer loop. Non-negative. **Default:** `2e-4`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`stepTolOuter` Numeric; Max step tolerance, outer loop. Non-negative. **Default:** `1e-4`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`gradTolInner` Numeric; Max gradient tolerance, inner loop (etas). Non-negative. **Default:** `1.71e-5`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`stepTolInner` Numeric; Max step tolerance, inner loop (etas). Non-negative. **Default:** `7.07e-8`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`refDeltaLagl` Numeric; Log-likelihood change tolerance. Non-negative. **Default:** `1e-3`. *(Population models,* `method = "FOCE-ELS"` *or* `"Laplacian"` *only).*

`mapAssist` Integer; Periodicity for MAP assistance. 0 disables. **Default:** `0L`. *(Population models,* `method = "QRPEM"` *only).*

`iSample` Integer; Sample points. Positive integer. **Default:** `300L`. *(Population models,* `method = "QRPEM"` *only).*

`iAcceptRatio` Numeric; Acceptance ratio for covariance scaling. Positive. **Default:** `0.1`. *(Population models,* `method = "QRPEM"` *only).*

`impDist` Character; Importance sampling distribution. Options: `"Normal"`, etc. **Default:** `"Normal"`. *(Population models,* `method = "QRPEM"` *only).*

`tDOF` Integer; Degrees of freedom for T distribution importance sampling (3-30). **Default:** `4L`. *(Population models,* `method = "QRPEM"` *and* `impDist = "T"` *only).*

`numSampleSIR` Integer; Samples per eta per subject for SIR. Positive integer. **Default:** `10L`. *(Population models,* `method = "QRPEM"` *only).*

`numBurnIn` Integer; Burn-in iterations. **Default:** `0L`. *(Population models,* `method = "QRPEM"` *only).*

`freezeOmega` Logical; Freeze Omega during burn-in. **Default:** `FALSE`. *(Population models,* `method = "QRPEM"` *only).*

`MCPEM` Logical; Use Monte-Carlo (`TRUE`) or Quasi-Random (`FALSE`) sampling. **Default:** `FALSE`. *(Population models,* `method = "QRPEM"` *only).*

`runAllIterations` Logical; Force execution of all iterations. **Default:** `FALSE`. *(Population models,* `method = "QRPEM"` *only).*

`scramble` Character; Quasi-random scrambling. Options: `"Owen"`, etc. **Default:** `"Owen"`. *(Population models,* `method = "QRPEM"` *only).*

`emTolType` Integer; QRPEM convergence check type (0-3). **Default:** `0L`. *(Population models,* `method = "QRPEM"` *only).*

`emConvLen` Integer; Iterations for QRPEM convergence check window. Positive. **Default:** `10L`. *(Used when* `emTolType` *> 0, QRPEM only).*

`emConvCritVal` Numeric; Critical value for QRPEM convergence check. Positive. **Default:** `5.0`. *(Used when* `emTolType` *> 0, QRPEM only).*

`stepSizePartialDeriv` Numeric; Step size for numerical partial derivatives. Positive. **Default:** `1e-5`. *(Individual models only).*

`numTimeStepPartialDeriv` Integer; Time steps for outputting partial derivatives. Positive integer. **Default:** `20L`. *(Individual models only).*

Details

This function allows customization of the NLME engine settings. Parameters are validated based on type, range, and applicability (detailed in parameter descriptions). Only parameters explicitly set to a value different from their default *for the specified context* (e.g., method-specific defaults for `stdErr`) are included in the output string. Values are returned as character strings.

Important Note on Defaults: Uses fixed defaults as specified in the argument list for comparison *unless otherwise noted* (e.g., `stdErr`). The actual default applied by NLME might differ based on model context (population vs individual, presence of reset info, discontinuities, BQL data).

Parameter Applicability & Warnings: The function checks for common cases where provided parameters might be ignored by the NLME engine based on the selected `method` or other settings. Warnings are issued in such cases. It assumes population context unless `method="Naive-Pooled"`.

Value

A **single character string** containing space-separated `name=value` pairs, ordered according to the function signature. Includes parameters if specified with non-default values (using method-specific defaults for `stdErr`).

See Also

`write_ModelTemplateTokens()`, `specify_SimParams()`

Examples

```
# Default settings
EstArgs_def <- specify_EngineParams()
print(EstArgs_def)

# Setting sort = TRUE
EstArgs_sort_true <- specify_EngineParams(sort = TRUE)
print(EstArgs_sort_true)

# QRPEM method with several custom settings
EstArgs_qrpem_str <-
  specify_EngineParams(
    sort = TRUE, # Explicitly non-default
    ODE = "DVERK",
    rtolODE = 1e-5,
    numIterations = 500,
    method = "QRPEM",
    isCentralDiffStdErr = FALSE,
    numIterMAPNP = 3,
    iSample = 350,
    impDist = "Mixture-2",
    scramble = "Tezuka-Faur",
    numBurnIn = 10,
    freezeOmega = TRUE
  )
print(EstArgs_qrpem_str)
```

`specify_SimParams` *Specify Engine Parameters for Model Simulation*

Description

Use to define engine parameters for model simulation. Generates a single character string containing space-separated `name=value` pairs.

Usage

```
specify_SimParams(
  numReplicates = 100L,
  seed = 1234L,
  sort = FALSE,
  ODE = c("MatrixExponent", "DVERK", "DOPRI5", "AutoDetect", "Stiff"),
  rtolODE = 1e-06,
  atolODE = 1e-06,
  maxStepsODE = 50000L
)
```

Arguments

`numReplicates`	Integer; Number of replicates (simulations) to generate. Must be positive. **Always included in output.**
`seed`	Integer; Seed for the random number generator used during simulation. **Always included in output.**
`sort`	Logical; Specifies whether to sort the input data by subject and time before simulation. **Default:** `FALSE`. **Always included in output.**
`ODE`	Character; Specifies the ODE solver. Options: `"MatrixExponent"`, `"DVERK"`, `"DOPRI5"`, `"AutoDetect"`, `"Stiff"`. **Default:** `"MatrixExponent"`.
`rtolODE`	Numeric; Relative tolerance for the ODE solver. **Default:** `1e-6`. *(Not applicable if* `ODE = "MatrixExponent"`*).*
`atolODE`	Numeric; Absolute tolerance for the ODE solver. **Default:** `1e-6`. *(Not applicable if* `ODE = "MatrixExponent"`*).*
`maxStepsODE`	Integer; Maximum number of steps for the ODE solver. **Default:** `50000L`. *(Not applicable if* `ODE = "MatrixExponent"`*).*

Details

This function allows customization of the NLME engine settings specific to simulation runs. Parameters are validated based on type and range.

The parameters `numReplicates`, `seed`, and `sort` are **always** included in the output string. Other parameters (`ODE`, `rtolODE`, `atolODE`, `maxStepsODE`) are included only if their specified value differs from the function's default value *and* they are applicable (ODE tolerances are ignored if `ODE="MatrixExponent"`).

Values are returned as character strings. The order of parameters in the output string matches the order in the function definition.

Value

A **single character string** containing space-separated `name=value` pairs, ordered according to the function signature. Includes `numReplicates`, `seed`, `sort` always, and other parameters if specified with non-default, applicable values.

See Also

`write_ModelTemplateTokens()`, `specify_EngineParams()`, `Table()`

Examples

```
# Default settings (includes numReplicates, seed, sort)
SimArgs1 <- specify_SimParams()
print(SimArgs1)

# Custom settings
SimArgs2 <-
  specify_SimParams(
    numReplicates = 50,
    seed = 9876,
    sort = TRUE, # Non-default
    ODE = "DVERK", # Non-default
    rtolODE = 1e-5 # Non-default and applicable
  )
print(SimArgs2)

# Custom settings where ODE tolerances are ignored
SimArgs3 <-
  specify_SimParams(
    numReplicates = 20,
    ODE = "MatrixExponent", # Default, but tolerances are now inapplicable
    rtolODE = 1e-4 # Non-default, but ignored
   )
print(SimArgs3)
```

---

stop_pyDarwin *Stop pyDarwin Model Search*

Description

This function stops a pyDarwin model search.

Usage

```
stop_pyDarwin(
  InterpreterPath,
  Flags = c("-u", "-m"),
  ForceStop = FALSE,
  DirectoryPath = "."
)
```

Arguments

InterpreterPath	Path to the Python interpreter executable.
Flags	Flags to pass to the Python interpreter. Refer to Python documentation for details. Note that `-m` is essential (runs library module as a script and terminates option list).
ForceStop	Logical. If `TRUE`, `-f` flag is added to force stop search immediately.- Default is `FALSE`.
DirectoryPath	the `DirectoryPath` argument of `run_pyDarwin()` or the parent folder of options file passed to that function. Default is current working directory.

Value

Returned code of `system2()` call.

Examples

```
## Not run:
stop_pyDarwin(
  InterpreterPath = "~/darwin/venv/bin/python",
  DirectoryPath = "~/project_folder")

## End(Not run)
```

---

stop_pyDarwinRemote *Stop a Remote pyDarwin Job*

Description

Attempts to gracefully stop a running pyDarwin job on a remote host by creating a 'stop.darwin' file in the remote project directory.

Usage

```
stop_pyDarwinRemote(
  LocalDirectoryPath,
  LocalJobInfoFilePath = NULL,
  OriginalOptionsPath = NULL,
  Password = NULL,
  KeyPath = NULL,
  verbose = getOption("verbose", default = FALSE)
)
```

Arguments

LocalDirectoryPath	Character string: The base local directory associated with the pyDarwin job. This directory is used to locate the job information file and the original options file.
LocalJobInfoFilePath	Character string (optional): Explicit path to the local JSON file containing remote job information. If NULL (default), it's constructed using `LocalDirectoryPath` and `ProjectName` (derived from `OriginalOptionsPath`).
OriginalOptionsPath	Character string (optional): Explicit path to the original local `options.json` file. If NULL (default), it's sought in `LocalDirectoryPath`. This is used to derive `ProjectName` if needed.
Password	Character string. The password for SSH authentication. Defaults to `""`, which is appropriate when using key-based authentication. Using keys is strongly recommended over embedding passwords in scripts.
KeyPath	Character string. The path to your private SSH key file. Defaults to the path stored in the `SSH_PRIVATE_KEY_PATH` environment variable.
verbose	Logical: Passed to `ssh::ssh_connect` for verbose SSH output. Default: `getOption("verbose", default = FALSE)`.

Value

Invisibly returns `TRUE` if the stop signal file was successfully created (or already existed), and `FALSE` if there was an SSH error or an error creating the file. This only signals the intent to stop; it does not confirm the pyDarwin process has actually stopped.

Examples

```
## Not run:
# Assuming 'my_project_remote_job_info.json' and 'options.json'
# exist in '~/darwin_runs/my_project_run'.
try({
  stop_pyDarwinRemote(
    LocalDirectoryPath = "~/darwin_runs/my_project_run",
    KeyPath = "~/.ssh/id_rsa_remote"
  )
```

```
})

# Explicitly providing paths
try({
 stop_pyDarwinRemote(
   LocalDirectoryPath = "~/darwin_runs/my_project_run", # Still used as a base if needed
  LocalJobInfoFilePath = "~/darwin_runs/my_project_run/my_project_remote_job_info.json",
   OriginalOptionsPath = "~/darwin_runs/my_project_run/options.json"
 )
})

## End(Not run)
```

`StParm` *Create an instance of a Structural parameter.*

Description

This function creates a new instance of a Structural parameter.

Usage

```
StParm(
  StParmName = character(),
  Type = "LogNormal",
  State = "Present",
  ThetaStParm = list(),
  OmegaStParm = list(),
  Covariates = list(),
  PMLStructure = character()
)
```

Arguments

`StParmName`	Character specifying the name of the structural parameter.
`Type`	Character specifying the type of the structural parameter. Options are

- `LogNormal` The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- `LogNormal1` The PML statement of the structural parameter will look like the following:

```
stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))
```

- `LogNormal2` The PML statement of the structural parameter will look like the following:

```
stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))
```

- `LogitNormal` The PML statement of the structural parameter will look like the following:

```
stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- `Normal` The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))
```

`State` character string that indicates the presence of the structural parameter. Options are:

- `None` The structural parameter does not exist in the specified `PMLStructures`.
- `Present` The structural parameter exists in the specified `PMLStructures` (the default).
- `Searched` The presence of the structural parameter is searched.

`ThetaStParm` A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + `StParmName`.

`OmegaStParm` An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + `StParmName`

`Covariates` A list of covariates (`Covariate` instances) that should be included in the structural parameter statement.

`PMLStructure` Character specifying the name of PML structure for which current parameter should be attributed. For the naming convention of PMLStructures, see Details section of `create_ModelPK()` for PK models and `create_ModelPD()` for PD models.

Value

An instance of a structural parameter.

See Also

Functions used for StParm specification: `add_StParm()`, `create_ModelPD()`, `create_ModelPK()`, `modify_StParm()`, `modify_StParmCustom()`, `remove_StParm()`

Examples

```
# Create a Structural parameter instance with default values
V <- StParm(StParmName = "V")

# Create a Structural parameter with Normal type:
V2 <- StParm("V2",
       Type = "Normal",
       ThetaStParm = Theta(Name = "tvV2", InitialEstimates = 0.1))

# Create a Structural parameter instance with covariates:
Cl <- StParm(
  StParmName = "Cl",
  Covariates = Covariate(
    Name = "Period",
```

```
    Type = "Occasion",
    State = "Searched",
    Categories = c(1,2),
    Omegas = list(Omega(Name = "nPeriodx1", 2),
                  Omega(Name = "nPeriodx2", 3))),
  PMLStructure = "1CFOE")
```

`Table` *Class initializer for NLME tables*

Description

Creates Table class object used to specify triggers and columns for tables output.

Usage

```
Table(
  Name = "table01.csv",
  TimesList = numeric(0),
  CovrSet = "",
  WhenDose = "",
  WhenObs = "",
  VariablesList = "",
  KeepSource = FALSE,
  TimeAfterDose = FALSE,
  IRES = FALSE,
  Weight = FALSE,
  IWRES = FALSE,
  Mode = "all",
  ForSimulation = FALSE
)
```

Arguments

`Name`	Character; Name of the generated table.
`TimesList`	Numeric; Time values for simulation. Applicable for time-based models only. Ignored when `keepSource=TRUE`.
`CovrSet`	Character; Vector of covariate names. Simulation point is added when the covariate value is set.
`WhenDose`	Character or logical; Vector of dosing compartment names. Alternatively if `WhenDose == TRUE`, triggers are added for all dosepoints for each PMLParametersSet separately; that approach is useful when different models in the set have different dosing compartments. Simulation point is added when the dose value is set.
`WhenObs`	Character; String of observed variables names. Simulation point is added when the observation value is set.

`VariablesList`	Character; List of variables from the model for simulation.
`KeepSource`	Logical; Set to `TRUE` to keep the number of rows appearing in the table the same as the number of rows in the input dataset.
`TimeAfterDose`	Set to `TRUE` to output time after dose.
`IRES`	Logical; Set to `TRUE` to output individual residuals. Valid only if `WhenObs` is specified and `ForSimulation==FALSE`.
`Weight`	Logical; Set to `TRUE` to output the weight of current observation. Valid only if `WhenObs` is specified and `ForSimulation==FALSE`.
`IWRES`	Logical; Set to `TRUE` to output individual weighted residuals. Valid only if `WhenObs` is specified and `ForSimulation==FALSE`.
`Mode`	Character; The mode of output. Options are `all` (default), `unique`, `first`. Only applicable to non time-based models for the case where only `CovrSet` is defined or the case where only `CovrSet` and `VariablesList` are defined. Since current version supports time-based models only, this argument is not applicable and won't change the output.
`ForSimulation`	Logical; Set to `TRUE` if the table should be generated during simulation, otherwise the table will be generated after fitting.

Details

If the table has a flag `ForSimulation==TRUE`, it will be ignored and won't be generated during estimation stage. Simulation stage should be added for simulation table generation. Tables with `ForSimulation==FALSE` will be ignored during simulation stage.

Value

A Table class used to store custom table information.

Examples

```
table01 <-
  Table(Name = "table01.csv",
      TimesList = seq(1,3,1),
      CovrSet = "WT",
      WhenDose = "A1",
      WhenObs = "CObs",
      VariablesList = "C",
      KeepSource = FALSE,
      TimeAfterDose = TRUE,
      IRES = TRUE,
      Weight = TRUE,
      IWRES = TRUE,
      ForSimulation = FALSE)
```

Theta *Create a new Theta instance with validation.*

Description

Create a new Theta instance with validation.

Usage

```
Theta(
  Name = character(),
  InitialEstimates = 1,
  State = "Present",
  Frozen = FALSE,
  StParmName = character(),
  PMLStructure = character()
)
```

Arguments

Name	A character string representing the name of the Theta instance.
InitialEstimates	An InitialEstimate() class instance or a numerical value for the initial estimate of the Theta or a numeric vector length three with its elements representing the lower bound, initial estimate.
State	Character specifying the presence of the Theta. Possible values are: • None The Theta does not exist in the specified PMLStructure. • Present The Theta exists in the specified PMLStructure (the default) • Searched The presence of the Theta is searched.
Frozen	A logical value indicating whether the Theta will be estimated or not.
StParmName	A character specifying the corresponding structural parameter name. Used for the Name of current Theta construction if it is not specified as 'tv' + StParmName.
PMLStructure	PML structure current theta belongs to

Value

A Theta instance.

See Also

InitialEstimate() StParm()

Functions used for Theta specification: InitialEstimate(), create_ModelPD(), create_ModelPK(), modify_Theta()

Examples

```
# Create a new Theta instance with a name 'tvV' and initial value 2 (no bounds)
theta <- Theta(Name = "tvV", InitialEstimates = 2)
```

`write_ModelTemplateTokens` *Prints NLME metamodel template file and token json file using given options, filepaths and data*

Description

This function generates and writes the model template and tokens files based on the provided inputs.

Usage

```
write_ModelTemplateTokens(
  TemplateFilePath = "template.txt",
  TokensFilePath = "tokens.json",
  Description = "",
  Author = "",
  DataFilePath,
  DataMapping = NULL,
  ColDef = "",
  PMLParametersSets,
  EstArgs = specify_EngineParams(),
  SimArgs = "",
  Tables = list(),
  AppendixRows = "",
  OmegaSearchBlocks = list()
)
```

Arguments

`TemplateFilePath`	TemplateFilePath NLME template file path to be written (usually txt).
`TokensFilePath`	json file path to be written (usually json).
`Description`	A problem name to be outputted in Description section.
`Author`	The author information for the model to be outputted in Author section.
`DataFilePath`	A data file path used by NLME.
`DataMapping`	Mapping of model terms to data column names, which can be: • **A named character vector**: Used when `PMLParametersSets` contains a single parameter set. Maps model terms to data columns. • **A named list of named character vectors**: Used when `PMLParametersSets` contains multiple parameter sets. Each element corresponds to a parameter set, with list names matching the names of `PMLParametersSets`.

`ColDef` A character string specifying additional column definitions in NLME column definition format. See Phoenix NLME documentation for details. https://onlinehelp.certara.com/phoenix/8.6/index.html#t=Phoenix_UserDocs%2FPML%2FColumn_mappings.htm

`PMLParametersSets` A list of PML parameters sets (`PMLModels` class instance).

`EstArgs` Estimation arguments for the model template. Please use `specify_EngineParams` to specify the arguments passed to NLME.

`SimArgs` Simulation arguments for the model template. Please use `specify_SimParams` to specify the arguments passed to NLME.

`Tables` A list of `Table` class instances specifying properties of the tables to be generated after fitting or during simulation.

`AppendixRows` Additional rows to include in the model template appendix in NLME column definition format. See Phoenix NLME documentation for details. https://onlinehelp.certara.com/phoenix/8.6/index.html#t=Phoenix_UserDocs%2FPML%2FColumn_mappings.htm

`OmegaSearchBlocks` A list of character vectors representing omega names to try to build block omegas.

Details

Mapping Details:
Basic Mapping: Maps a model variable name (e.g., `CObs`) to a column name in your data file (e.g., `"Concentration"`).
Shorthand Mapping: If an element is unnamed, the Model Term is assumed to be the same as the Data Column Term. Example: `c(ID = "Subject", "Age", Weight = "WT")` is equivalent to `c(ID = "Subject", Age = "Age", Weight = "WT")`.
Multiple ID/Grouping Levels: Model terms matching the pattern `ID[0-9]?` (i.e., `ID`, `ID0`, `ID1`, `ID2`, `ID3`, `ID4`), case-insensitive, are automatically recognized by `Certara.RsNLME` as NLME sort keys/grouping levels. You can map up to 5 such levels. The function uses these to structure the model execution.
Covariates: If not explicitly mapped, the function attempts to map them using data column names that match covariate names in the model.
Mapping a List: When using a list, each vector must map terms specific to its parameter set, and the list length must equal the number of parameter sets.
Special Terms:

- Terms `<DosepointName>_Duration` or `<DosepointName>_Rate` could be used to map rate/duration columns for the corresponding dosepoints (e.g., `A1_Rate = "InfRate"`). Term `<ObservationName>BQL` could be used to map a BQL flag column for the corresponding observation (e.g., `CObsBQL = "ConcBQL"`).

- The generic `AMT` term can be used to map the dose amount column; the function will automatically associate it with the primary absorption compartment (e.g., `A1` for zero-order/bolus, `Aa` for first-order) for each parameter set.

- Generic `Duration` or `Rate` terms can be mapped (e.g., `Rate = "InfRate"`); the function will associate them with the dose mapped via `AMT`. If a specific mapping like `A1_Rate` exists, it overrides the generic `Rate` mapping for that dosepoint (`A1`).

Value

A list containing statements written to template and tokens files.

See Also

`specify_EngineParams()`, `specify_SimParams()`, `Table()`

Examples

```
# Write model template and tokens files
PMLParametersSets <- create_ModelPK(CompartmentsNumber = c(1,2))
# write test data frame
TempFolder <- tempdir()
TemplateFilePath <- file.path(TempFolder, "template.txt")
TokensFilePath <- file.path(TempFolder, "tokens.json")
DataFilePath <- file.path(TempFolder, "Data.csv")
# Ensure data file has columns matching the DataMapping values
write.csv(data.frame(Subject = 'id_1', # Column for ID
                     StudyDay = 1,      # Column for ID1
                     time = 0,         # Column for time
                     DoseAmt = 100,     # Column for AMT
                     Concentration = 10.5, # Column for CObs
                     SubjectAge = 45,   # Column for Age
                     Weight = 70,       # Column for Weight
                     ConcBQL = 0),      # Column for CObsBQL
                     DataFilePath, row.names = FALSE) # Use row.names=FALSE

write_ModelTemplateTokens(
  TemplateFilePath = TemplateFilePath,
  TokensFilePath = TokensFilePath,
  Description = "1-2Cpts try with Multi-ID and Shorthand",
  Author = "Certara",
  DataFilePath = DataFilePath,
  DataMapping = c(ID = "Subject",   # Map ID model term to Subject column
                  ID1 = "StudyDay", # Map ID1 model term to StudyDay column
                  time = "time",    # Map time model term to time column
                  CObs = "Concentration", # Map CObs to Concentration
                  AMT = "DoseAmt",  # Map generic AMT to DoseAmt
                  "SubjectAge",     # Shorthand: Map Age model term to SubjectAge column
                  Weight = "Weight",# Map Weight model term to Weight column
                  CObsBQL = "ConcBQL"), # Map BQL flag
  ColDef = "",
  PMLParametersSets = PMLParametersSets,
  EstArgs = specify_EngineParams(method = "QRPEM"),
  SimArgs = specify_SimParams(numReplicates = 1000L),
  Tables = list(Table(Name = "simtable1.csv",
```

```
                      KeepSource = TRUE,
                      VariablesList = "C",
                      ForSimulation = TRUE)),
  OmegaSearchBlocks = list(c("nCl", "nV"), c("nCl2", "nV2")))

# Multiple parameter sets
PMLParametersSets <- create_ModelPK(Absorption =c("Intravenous", "Weibull"))
DataMapping <- list(
  c(ID = "Subject", time = "time", Aa = "DoseAmt", CObs = "Concentration"),
  c(ID = "Subject", time = "time", A1 = "DoseAmt", CObs = "Concentration")
)

names(DataMapping) <- names(PMLParametersSets)

write_ModelTemplateTokens(
  TemplateFilePath = TemplateFilePath,
  TokensFilePath = TokensFilePath,
  Description = "1 Cpt Weibull and First-Order",
  Author = "Certara",
  DataFilePath = DataFilePath,
  DataMapping = DataMapping,
  PMLParametersSets = PMLParametersSets)
```

`write_pyDarwinOptions` *Write pyDarwin options to a JSON file.*

Description

This function takes a list of pyDarwin options and writes them to a JSON file in the specified format. The options can be generated using the `create_pyDarwinOptions` function or customized manually. The resulting JSON file can be used as input for a pyDarwin model search.

Usage

```
write_pyDarwinOptions(
  pyDarwinOptions = create_pyDarwinOptions(),
  file = "options.json",
  pretty = TRUE,
  digits = NA,
  auto_unbox = TRUE
)
```

Arguments

`pyDarwinOptions`
: A list containing the pyDarwin options to be written to the JSON file. Default is the result of calling `create_pyDarwinOptions()` with default arguments.

file	Character: The path to the JSON file where the options will be written. Default is a file named "options.json" in the current working directory.
pretty	adds indentation whitespace to JSON output. Can be TRUE/FALSE or a number specifying the number of spaces to indent (default is 2). Use a negative number for tabs instead of spaces.
digits	max number of decimal digits to print for numeric values. Use I() to specify significant digits. Use NA for max precision.
auto_unbox	automatically unbox() all atomic vectors of length 1. It is usually safer to avoid this and instead use the unbox() function to unbox individual elements. An exception is that objects of class AsIs (i.e. wrapped in I()) are not automatically unboxed. This is a way to mark single values as length-1 arrays.

Value

None (invisible NULL).

Examples

```
# Write pyDarwin options to a JSON file
Options <-
  create_pyDarwinOptions(author = "John Doe",
                         algorithm = "GA",
                         population_size = 10)
write_pyDarwinOptions(Options,
                      file = file.path(tempdir(), "options.json"))
```

Index