# Package 'mtk'

July 22, 2014

**Type** Package

**Title** Mexico ToolKit library (MTK)

**Encoding** UTF-8

**Version** 1.0

**Date** 2014-07-15

**Author** Juhui WANG Hervé Richard Hervé Monod Robert Faivre

**Maintainer** Juhui WANG <Juhui.Wang@jouy.inra.fr>

**Description** Mexico ToolKit library

**License** GPL-3

**LazyLoad** yes

**Depends** R (>= 2.15.0), base, stringr, graphics, methods, XML,sensitivity, lhs, rgl

**Suggests** MASS

**Collate** 'mtkAllGenerics.R' 'globalsMtkFuncts.R' 'mtkValue.R'
'mtkFeature.R' 'mtkLevels.R' 'mtkParameter.R' 'mtkDomain.R'
'mtkFactor.R' 'mtkExpFactors.R' 'mtkProcess.R'
'mtkExpWorkflow.R' 'mtkExperiment.R' 'mtkParsor.R'
'mtkResult.R' 'mtkDesignerResult.R' 'mtkDesigner.R'
'mtkMorrisDesigner.R' 'mtkBasicMonteCarloDesigner.R'
'mtkRandLHSDesigner.R' 'mtkNativeDesigner.R'
'mtkSobolDesigner.R' 'mtkFastDesigner.R' 'mtkEvaluatorResult.R'
'mtkEvaluator.R' 'mtkNativeEvaluator.R' 'mtkIshigamiEvaluator.R' 'mtkWWDMEvaluator.R'
'mtkSystemEvaluatorResult.R' 'mtkSystemEvaluator.R'
'mtkAnalyserResult.R' 'mtkAnalyser.R' 'mtkNativeAnalyser.R'
'mtkDefaultAnalyser.R' 'mtkRegressionAnalyser.R'
'mtkMorrisAnalyser.R' 'mtkSobolAnalyser.R' 'mtkPLMMAnalyser.R'
'mtkFastAnalyser.R' 'mtk.addons.R' 'mtkPackage.R'

# R topics documented:

mtk-package . . . 5
addProcess-methods . . . 7
ANY . . . 8
BasicMonteCarlo . . . 9
deleteProcess-methods . . . 10
extractData-methods . . . 12
Fast . . . 13
getData-methods . . . 15
getDiscreteDistributionLevels-methods . . . 17
getDiscreteDistributionType-methods . . . 17
getDiscreteDistributionWeights-methods . . . 18
getDistributionName-methods . . . 19
getDistributionNames-methods . . . 20
getDistributionNominalValue-methods . . . 21
getDistributionNominalValues-methods . . . 21
getDistributionNominalValueType-methods . . . 22
getDistributionNominalValueTypes-methods . . . 23
getDistributionParameters-methods . . . 24
getDomain-methods . . . 24
getFactorFeatures-methods . . . 25
getFactorNames-methods . . . 26
getFactors-methods . . . 27
getFeatures-methods . . . 28
getLevels-methods . . . 28
getMTKFeatures-methods . . . 29
getName-methods . . . 30
getNames-methods . . . 30
getNominalValue-methods . . . 31
getNominalValueType-methods . . . 32
getParameters-methods . . . 33
getProcess-methods . . . 34
getResult-methods . . . 35
getType-methods . . . 36
getValue-methods . . . 37
getWeights-methods . . . 38
is.finished-methods . . . 38
is.ready-methods . . . 39
Ishigami . . . 40
Ishigami.factors . . . 42
make.mtkFactor . . . 43
make.mtkFeatureList . . . 44
make.mtkParameterList . . . 45
Morris . . . 45
mtk.analyserAddons . . . 48
mtk.designerAddons . . . 50
mtk.evaluatorAddons . . . 52
mtkAnalyser . . . 54
mtkAnalyser-class . . . 55
mtkAnalyserResult . . . 57
mtkAnalyserResult-class . . . 58

mtkBasicMonteCarloDesigner . . . 59
mtkBasicMonteCarloDesigner-class . . . 60
mtkBasicMonteCarloDesignerResult . . . 62
mtkBasicMonteCarloDesignerResult-class . . . 63
mtkDefaultAnalyser . . . 64
mtkDefaultAnalyser-class . . . 65
mtkDesigner . . . 67
mtkDesigner-class . . . 68
mtkDesignerResult . . . 70
mtkDesignerResult-class . . . 70
mtkDomain . . . 71
mtkDomain-class . . . 72
mtkEvaluator . . . 74
mtkEvaluator-class . . . 75
mtkEvaluatorResult . . . 77
mtkEvaluatorResult-class . . . 77
mtkExperiment . . . 78
mtkExperiment-class . . . 80
mtkExpFactors . . . 82
mtkExpFactors-class . . . 82
mtkExpWorkflow . . . 84
mtkExpWorkflow-class . . . 85
mtkFactor . . . 88
mtkFactor-class . . . 89
mtkFastAnalyser . . . 90
mtkFastAnalyser-class . . . 91
mtkFastAnalyserResult . . . 93
mtkFastAnalyserResult-class . . . 94
mtkFastDesigner . . . 95
mtkFastDesigner-class . . . 96
mtkFastDesignerResult . . . 98
mtkFastDesignerResult-class . . . 99
mtkFeature . . . 100
mtkFeature-class . . . 101
mtkIshigamiEvaluator . . . 102
mtkIshigamiEvaluator-class . . . 103
mtkLevels . . . 105
mtkLevels-class . . . 106
mtkMorrisAnalyser . . . 107
mtkMorrisAnalyser-class . . . 108
mtkMorrisAnalyserResult . . . 110
mtkMorrisAnalyserResult-class . . . 111
mtkMorrisDesigner . . . 112
mtkMorrisDesigner-class . . . 113
mtkMorrisDesignerResult . . . 115
mtkMorrisDesignerResult-class . . . 116
mtkNativeAnalyser . . . 117
mtkNativeAnalyser-class . . . 118
mtkNativeDesigner . . . 120
mtkNativeDesigner-class . . . 121
mtkNativeEvaluator . . . 123
mtkNativeEvaluator-class . . . 125

mtkParameter . . . 128
mtkParameter-class . . . 129
mtkParsor . . . 130
mtkParsor-class . . . 131
mtkPLMMAnalyser . . . 132
mtkPLMMAnalyser-class . . . 133
mtkPLMMAnalyserResult . . . 134
mtkPLMMAnalyserResult-class . . . 135
mtkProcess . . . 136
mtkProcess-class . . . 138
mtkRandLHSDesigner . . . 139
mtkRandLHSDesigner-class . . . 140
mtkRandLHSDesignerResult . . . 141
mtkRandLHSDesignerResult-class . . . 142
mtkReadFactors-methods . . . 143
mtkRegressionAnalyser . . . 143
mtkRegressionAnalyser-class . . . 144
mtkRegressionAnalyserResult . . . 146
mtkRegressionAnalyserResult-class . . . 147
mtkResult . . . 148
mtkResult-class . . . 149
mtkSobolAnalyser . . . 150
mtkSobolAnalyser-class . . . 151
mtkSobolAnalyserResult . . . 152
mtkSobolAnalyserResult-class . . . 153
mtkSobolDesigner . . . 154
mtkSobolDesigner-class . . . 155
mtkSobolDesignerResult . . . 156
mtkSobolDesignerResult-class . . . 157
mtkSystemEvaluator . . . 158
mtkSystemEvaluator-class . . . 159
mtkSystemEvaluatorResult . . . 160
mtkSystemEvaluatorResult-class . . . 161
mtkValue . . . 162
mtkValue-class . . . 162
mtkWWDMEvaluator . . . 163
mtkWWDMEvaluator-class . . . 165
mtkWWDMEvaluatorResult . . . 167
mtkWWDMEvaluatorResult-class . . . 168
PLMM . . . 169
plot,mtkProcess-method . . . 172
print,mtkProcess-method . . . 173
Quantiles . . . 175
RandLHS . . . 175
reevaluate-methods . . . 177
Regression . . . 178
report-methods . . . 179
run-methods . . . 180
serializeOn-methods . . . 181
setDistributionParameters-methods . . . 182
setDomain-methods . . . 183
setFactors-methods . . . 184

setFeatures-methods . . . 185
setLevels-methods . . . 185
setName-methods . . . 186
setParameters-methods . . . 187
setProcess-methods . . . 188
setReady-methods . . . 189
setState-methods . . . 190
setType-methods . . . 191
setValue-methods . . . 192
setWeights-methods . . . 193
setXMLFilePath-methods . . . 194
Sobol . . . 195
summary,mtkProcess-method . . . 198
WWDM . . . 199
wwdm.climates . . . 201
WWDM.factors . . . 202

**Index** **204**

---

`mtk-package` *MTK (Mexico ToolKit) for Sensitivity Analysis and Numerical Experiments*

---

## Description

MTK is an R package for sensitivity analysis and numerical experiments . Three examples are provided:

- "Ishigami" model analysis with the "BasicMonteCarlo" and "Regression" methods.
- Using the "mtk" package from a XML file.
- "WWDM (Winter Wheat Dry Matter)" model analysis with the "Morris" methods.

To run the examples, just load the package and type respectively:

- `demo(demo1,package="mtk", ask=FALSE)`
- `demo(demo2,package="mtk", ask=FALSE)`
- `demo(demo3,package="mtk", ask=FALSE)`

The following methods and models are available for the current release:

- The "Fast" methods for experiments design and sensitivity index calculation. see `help(Fast)`.
- The "Morris" methods for experiments design and sensitivity index calculation. see `help(Morris)`.
- The "Sobol" methods for experiments design and sensitivity index calculation. see `help(Sobol)`.
- The "Monte-Carlo" methods for experiments design. see `help(BasiMonteCarlo)`.
- The "LHS" methods for experiments design. see `help(RandLHS)`.
- The "PLMM (Polynomial Linear Meta-Model)" methods for sensitivity analysis. see `help(PLMM)`.
- The "Regression" methods for sensitivity index calculation. see `help(Regression)`.
- The "Ishigami" model for model simulation. see `help(Ishigami)`.
- The "WWDM (Winter Wheat Dry Matter)" model for model simulation. see `help(WWDM)`.

## Author(s)

The Mexico Group. Contact: Juhui WANG, MIA-Jouy, Juhui.Wang@jouy.inra.fr,

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
### Example 1: Sensitivity analysis of the "Ishigami" model ###

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1",
   distribName="unif", distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
   distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))
# Build the processes:
#  1) the experimental design process with the method "Morris".
exp1.Designer <- mtkMorrisDesigner(listParameters
= list(r=20,type="oat",levels=4,grid.jump=2))

#  2) the model simulation process with the model "Ishigami".
exp1.Evaluator <- mtkIshigamiEvaluator()

#  3) the analysis process with the method "Morris".
exp1.Analyser <- mtkMorrisAnalyser()

# Build the  workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
     processesVector = c(
     design=exp1.Designer,
evaluate=exp1.Evaluator,
analyze=exp1.Analyser)
)

# Run the workflow and reports the results.
run(exp1)
print(exp1)

# Create a new process with the analysis method «Regression».
exp1.AnalyserReg <- mtkRegressionAnalyser(listParameters
=list(nboot=20)
 )

# Re-analyze the model "Ishigami" with the method "Regression":
## replace the process, run the workflow and  report the results

setProcess(exp1, exp1.AnalyserReg, "analyze")
run(exp1)
```

```
print(exp1)

### Example 2 : Sensitivity analysis from a XML file  ###

# # XML file is held in the directory of the library: "inst/extdata/"

# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"

## Find where the examples are held.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",xmlFile,sep = "")

# Create the workflow
##  Nota: If your XML file is local file for example "/var/tmp/X.xml",
##  you should create the workflow as follows:
##  workflow <- mtkExpWorkflow(xmlFilePath="/var/tmp/X.xml")

workflow <- mtkExpWorkflow(xmlFilePath=xmlFilePath)

# Run the workflow and report the results
run(workflow)
summary(workflow)
```

---

## addProcess-methods *The* addProcess *method*

---

### Description

Adds a process to the workflow.

### Usage

```
addProcess(this,p,name)
```

### Arguments

| | |
|---|---|
| this | an object of the class mtkExpWorkflow. |
| p | an object of the class mtkProcess. |
| name | a string from "design", "evaluate", or "analyze" to specify the type of process to add. |

### Value

invisble()

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Define the factors

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
distribPara=list(min=-pi, max=pi))

ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Create a workflow to manager the processes used for the analysis task

ishiReg <- mtkExpWorkflow(expFactors=ishi.factors)

# Create a designer to generate the experiments design and
# put the designer under control of the workflow

designer <- mtkNativeDesigner("BasicMonteCarlo",
            information=list(size=20))

addProcess(ishiReg, designer, name="design")

# Creates an evaluator and add it to the workflow

model <- mtkNativeEvaluator("Ishigami" )

addProcess(ishiReg, model, name="evaluate")

# Create a analyser and add it to the workflow

analyser <- mtkNativeAnalyser("Regression" )

addProcess(ishiReg, analyser, name="analyze")

# Run the workflow and reports the results

run(ishiReg)
summary(ishiReg)
```

---

`ANY` *The ANY class*

---

## Description

ANY is a data type to represent any S4 class.

### Details

S4 implements the ANY class, but does not document it.

### Examples

```
# creates a new class with "ANY"
setClass(Class="mtkProcess",

representation=representation(
name="character",
protocol="character",
site="character",
service="character",
parameters="ANY",
ready="logical",
state="logical",
result="ANY"
),

prototype=prototype(parameters=NULL, ready=FALSE,
                    state=FALSE, result=NULL)

)
```

---

## BasicMonteCarlo *The* BasicMonteCarlo *design method*

---

### Description

A native mtk design method to generate Monte Carlo samples.

### Usage

- mtkBasicMonteCarloDesigner(listParameters=NULL)
- mtkNativeDesigner(design="BasicMonteCarlo", information=NULL)

### Parameters

**size :** the sample size.

### Details

1. The mtk implementation of the Basic Monte-Carlo method includes the following classes:
   - mtkBasicMonteCarloDesigner for Basic Monte-Carlo design processes.
   - mtkBasicMonteCarloDesignerResult to store and manage the design.
2. Many ways to create a Basic Monte-Carlo designer are available in mtk, but we recommend the following class constructors: mtkBasicMonteCarloDesigner or mtkNativeDesigner.

## References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
## Experiments design with the "Basic Monte-Carlo" method for the "Ishigami" model

# Example I: by using the class constructors: mtkBasicMonteCarloDesigner()

#  1) Create a designer process based on the Basic Monte-Carlo method
MCdesign <- mtkBasicMonteCarloDesigner(listParameters = list(size=20))

#  2) Import the input factors of the "Ishigami" model
data(Ishigami.factors)

#  3) Build and run the  workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
               processesVector = c(design=MCdesign))
run(exp1)

#  4) Report and plot the design
     show(exp1)
     plot(exp1)

# Example II: by using the class constructors: mtkNativeDesigner()

#  1) Create a designer process based on the Basic Monte-Carlo method
MCdesign <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=20))

#  2) Import the input factors of the "Ishigami" model
data(Ishigami.factors)

#  3) Build and run the  workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
               processesVector = c(design=MCdesign))
run(exp1)

#  4) Print and plot the design
print(exp1)
plot(exp1)
```

---

`deleteProcess-methods`

*The* `deleteProcess` *method*

---

## Description

Deletes a process from the workflow.

## Usage

```
deleteProcess(this, name)
```

## Arguments

| | |
|---|---|
| this | an object of the class mtkExpWorkflow. |
| name | a string from "design", "evaluate", or "analyze" to specify the process to delete. |

## Value

invisble()

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create an analysis for the Ishigami model:

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))


designer <- mtkNativeDesigner("BasicMonteCarlo",
            information=list(size=20))
model <- mtkNativeEvaluator("Ishigami" )
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

ishiReg <- mtkExpWorkflow( expFactors=ishi.factors,
   processesVector=c( design=designer,
      evaluate=model,
      analyze=analyser)
   )
run(ishiReg)
summary(ishiReg)

# Delete the analysis process from the workflow and
# run only the model simulation:

deleteProcess(ishiReg, "analyze")
run(ishiReg)
```

```
summary(ishiReg)
```

---

`extractData-methods`

*The* `extractData` *method*

---

## Description

Gets the results produced by the workflow as a data.frame.

## Usage

```
extractData(this,name)
```

## Arguments

| | |
|---|---|
| `this` | an object of the class `mtkExpWorkflow`. |
| `name` | a vector of strings from "design", "evaluate", or "analyze" to specify the results to return. i.e. name =c("design") returns the experimental design produced by the designer, name=c("design", "evaluate") returns both the experimental design produced by the designer and the model simulation produced by the evaluator. |

## Value

a data.frame

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Build a workflow for sensitivity analysis with the model "Ishigami"

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))


designer <- mtkNativeDesigner("BasicMonteCarlo",
```

```
            information=list(size=20))
model <- mtkNativeEvaluator("Ishigami" )
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

ishiReg <- mtkExpWorkflow(expFactors=ishi.factors,
   processesVector=c(design = designer,
      evaluate = model,
      analyze = analyser)
   )
run(ishiReg)

# extracts the results produced by the workflow as a data.frame:

design <- extractData(ishiReg, "design")
simulation <-  extractData(ishiReg, c("design", "evaluate"))
```

---

`Fast` *The extended Fourier Amplitude Sensitivity Test for sensitivity analysis*

---

## Description

A `mtk` compliant implementation of the so-called `extended-FAST` or `e-Fast` method for experiments design and sensitivity analysis.

## Usage

- mtkFastDesigner(listParameters = NULL)
- mtkNativeDesigner(design="Fast", information=NULL)
- mtkFastAnalyser()
- mtkNativeAnalyser(analyze="Fast", information=NULL)

## Parameters used to manage the sampling method

`n`**:** (`numeric`) the number of iteration.

## Parameters used to manage the analysis method

No parameter is necessary.

## Details

1. The `mtk` implementation uses the `fast99` function of the `sensitivity` package. For further details on the arguments and the behaviour, see `help(fast99, sensitivity)`.
2. The `mtk` implementation of the `Fast` method includes the following classes:

   `mtkFastDesigner`**:** for `Fast` design processes.

   `mtkFastAnalyser`**:** for `Fast` analysis processes.

   `mtkFastDesignerResult`**:** to store and manage the design.

   `mtkFastAnalyserResult`**:** to store and manage the analysis results.
3. Many ways to create a `Fast` designer are available in `mtk`, but we recommend the following class constructors: `mtkFastDesigner` or `mtkNativeDesigner`.

4. Many ways to create a `Fast` analyser are available in `mtk`, but we recommend the following class constructors: `mtkFastAnalyser` or `mtkNativeAnalyser`.
5. The method `Fast` is usually used both to build the experiment design and to carry out the sensitivity analysis. In such case, we can use the `mtkDefaultAnalyser` instead of naming explicitly the method for sensitivity analysis (see example III in the examples section)

## References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## See Also

```
help(fast99, sensitivity)
```

## Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Example I: by using the class constructors: mtkFastDesigner() and mtkFastAnalyser()

# Input the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkFastDesigner(listParameters
     = list(n=1000))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkFastAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
    plot(exp1)

##  Example II:  by using the class constructors: mtkNativeDesigner() and mtkFastAnalyser

# Generate the factors
data(Ishigami.factors)
```

```
#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkNativeDesigner(design = "Fast",information=list(n=1000))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method
exp1.analyser <- mtkFastAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
plot(exp1)


##  Example III:  by using the class constructors: mtkFastDesigner() and mtkDefaultAnalys

# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkFastDesigner( listParameters = list(n=2000))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method
exp1.analyser <- mtkDefaultAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
plot(exp1)
```

---

`getData-methods` *The* `getData` *method*

---

## Description

Returns the results produced by the process as a data.frame.

## Usage

```
getData(this)
```

## Arguments

| | |
|---|---|
| this | an object of the class mtkProcess or its sub-classes |

## Value

a data.frame.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
## Example: Sensitivity analysis for the Ishigami model

# Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes
designer <- mtkNativeDesigner("BasicMonteCarlo",
            information=list(size=20))
model <- mtkNativeEvaluator("Ishigami" )
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

# Build the workflow and run it
ishiReg <- mtkExpWorkflow(expFactors=ishi.factors,
   processesVector=c( design=designer,
      evaluate=model,
      analyze=analyser)
   )
run(ishiReg)

# Extract as a data.frame the experiment design:
designer <- getProcess(ishiReg, "design")
expDesign <- getData(designer)
```

## getDiscreteDistributionLevels-methods
*The* `getDiscreteDistributionLevels` *method*

### Description

Returns the levels of the discrete distribution associated with the factor's domain.

### Usage

```
getDiscreteDistributionLevels(this)
```

### Arguments

| | |
|---|---|
| `this` | the underlying object of the class to proceed (`mtkFactor`). |

### Value

a list.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a discrete domain
x1 <- make.mtkFactor(name="x1", distribName="discrete",
 distribPara= list(type='categorical',
 levels = c(1,2,3,4,5), weights=rep(0.2, 5)))

# Returns the levels of the associated discrete distribution
getDiscreteDistributionLevels(x1)
```

## getDiscreteDistributionType-methods
*The* `getDiscreteDistributionType` *method*

### Description

Returns the type of the discrete distribution associated with the factor's domain.

### Usage

```
getDiscreteDistributionType(this)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class to proceed (mtkFactor). |

### Value

a string.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a discrete domain
x1 <- make.mtkFactor(name="x1", distribName="discrete",
 distribPara= list(type='categorical',
 levels = c(1,2,3,4,5), weights=rep(0.2, 5)))
# Returns the type of the associated discrete distribution
getDiscreteDistributionType(x1)
```

---

## getDiscreteDistributionWeights-methods

*The* getDiscreteDistributionWeights *method*

---

### Description

Returns the weights of the discrete distribution associated with the factor's domain.

### Usage

```
getDiscreteDistributionWeights(this)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class to proceed (mtkFactor). |

### Value

a list of numeric values.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a discrete domain
x1 <- make.mtkFactor(name="x1", distribName="discrete",
 distribPara= list(type='categorical',
 levels = c(1,2,3,4,5), weights=rep(0.2, 5)))
# Returns the weights of the associated discrete distribution
getDiscreteDistributionWeights(x1)
```

---

## getDistributionName-methods
*The* getDistributionName *method*

---

### Description

Returns the name of the distribution associated with a domain or a factor.

### Usage

```
getDistributionName(this)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class to proceed (mtkDomain or mtkFactor). |

### Value

a string.

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a domain and get the name of its distribution
d <- mtkDomain(distributionName="unif", domainNominalValue=0)
distribution <- getDistributionName(d)

# For more information, see examples for the mtkDomain or
# mtkFactor classes.
```

`getDistributionNames-methods`
*The* `getDistributionNames` *method*

## Description

Returns the names of the distributions associated with an object of the class `mtkExpFactors`.

## Usage

```
getDistributionNames(this)
```

## Arguments

| | |
|---|---|
| `this` | an object of the `mtkExpFactors` class. |

## Value

a list.

## Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

## Examples

```
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Get the names of the distributions managed by all the factors
names <- getDistributionNames(ishi.factors)
```

## getDistributionNominalValue-methods
*The* `getDistributionNominalValue` *method*

### Description

Returns the nominal value associated with the uncertainty domain of a factor.

### Usage

```
getDistributionNominalValue(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the class `mtkFactor`. |

### Value

ANY

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a a factor with a nominal value

x1 <- make.mtkFactor(name="x1", type='numeric', nominal=0.0, distribName="unif",
             distribPara=list(min=-pi, max=pi))

getDistributionNominalValue(x1)
```

## getDistributionNominalValues-methods
*The* `getDistributionNominalValues` *method*

### Description

Gets the nominal values associated with the managed factors.

### Usage

```
getDistributionNominalValues(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the class `mtkExpFactors`) |

### Value

a named list

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Return the nominal values
nValues <- getDistributionNominalValues(ishi.factors)
```

---

## getDistributionNominalValueType-methods

*The* `getDistributionNominalValueType` *method*

---

### Description

Returns the nominal value associated with the uncertainty domain of a factor.

### Usage

```
getDistributionNominalValueType(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the class `mtkFactor`. |

### Value

string

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a a factor with a nominal value

x1 <- make.mtkFactor(name="x1", type='numeric', nominal=0.0, distribName="unif",
             distribPara=list(min=-pi, max=pi))

getDistributionNominalValueType(x1)
```

---

## getDistributionNominalValueTypes-methods
*The* `getDistributionNominalValueTypes` *method*

---

### Description

Gets the nominal values associated with the managed factors.

### Usage

```
getDistributionNominalValueTypes(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the class `mtkExpFactors`) |

### Value

a named list

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Return the nominal values
nTypes <- getDistributionNominalValueTypes(ishi.factors)
```

## getDistributionParameters-methods *The* getDistributionParameters *method*

### Description

Gets the parameters of the distribution(s) associated with an object (mtkDomain, mtkFactor or mtkExpFactors).

### Usage

```
getDistributionParameters(this)
```

### Arguments

| | |
|---|---|
| this | an object of the underlying class (mtkDomain, mtkFactor or mtkExpFactors) |

### Value

a named list or a nested list

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Return the parameters of the distributions managed by all the factors as a nested list
names <- getDistributionParameters(ishi.factors)
```

## getDomain-methods *The* getDomain *method*

### Description

Returns the domain associated with the factor.

### Usage

```
getDomain(this)
```

### Arguments

| | |
|---|---|
| this | an object of the class mtkFactor . |

### Value

an object of the class mtkDomain

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))

# Return the  uncertainty domain associated with the factor
dom <- getDomain(x1)
```

---

## getFactorFeatures-methods
*The* getFactorFeatures *method*

---

### Description

Returns the features associated with the managed factors.

### Usage

```
getFactorFeatures(this)
```

### Arguments

| | |
|---|---|
| this | an object of the mtkExpFactors class |

### Value

a named list.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))

# Define a list of features and associate it with the factor x1
features <- make.mtkFeatureList(list(pre=5, post=60))
setFeatures(x1, features)

x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Get the features of the managed factors as a list
factors <- getFactorFeatures(ishi.factors)
```

---

## getFactorNames-methods

*The* `getFactorNames` *method*

---

### Description

Returns the name of the managed factors.

### Usage

```
getFactorNames(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the class `mtkExpFactors`. |

### Value

a list of strings

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
```

```
      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Get the names of the factors managed by all the factors
factors <- getFactorNames(ishi.factors)
```

---

`getFactors-methods` *The* `getFactors` *method*

---

## Description

Retunrs the managed factors.

## Usage

```
getFactors(this)
```

## Arguments

| | |
|---|---|
| `this` | the underlying object of the class `mtkExpFactors`. |

## Value

a list of objects from the class `mtkFactor`.

## Author(s)

Juhui WANG, MIA-jouy, INRA

## Examples

```
# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors()

# Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
       distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
       distribPara=list(min=-pi, max=pi))
# Assign and return the factors to the mtkExpFactors' object

setFactors(ishi.factors, list(x1,x2,x3))
getFactors(ishi.factors)
```

## getFeatures-methods *The* getFeatures *method*

### Description

Returns the features associated with the underlying factor.

### Usage

```
getFeatures(this)
```

### Arguments

| | |
|---|---|
| this | an object of the mtkFactor class |

### Value

a named list.

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))

# Define a list of features and associate it with the factor
features <- make.mtkFeatureList(list(pre=5, post=60))
setFeatures(x1, features)

# Return the features associated with the factor
fl <- getFeatures(x1)
```

## getLevels-methods *The* getLevels *method*

### Description

Returns the levels associated with a discrete domain.

### Usage

```
getLevels(this)
```

### Arguments

| | |
|---|---|
| this | an object of the class mtkDomain or mtkLevels . |

### Value

a list

### Examples

```
 l <- mtkLevels(type='categorical', levels=seq(1:10), weight=rep(0.1, 10))
getLevels(l)
```

---

`getMTKFeatures-methods`
*The* `getMTKFeatures` *method*

---

### Description

Returns the features associated with the underlying factor as a list of `mtkFeature` objects.

### Usage

```
getMTKFeatures(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the `mtkFactor` class |

### Value

a list of objects of the class `mtkFeature`

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))

# Define a list of features and associate it with the factor
features <- make.mtkFeatureList(list(pre=5, post=60))
setFeatures(x1, features)

# Return the features associated with the factor
fl <- getMTKFeatures(x1)
```

## getName-methods *The* getName *method*

### Description

Returns the name of the object or a process.

### Usage

```
getName(this)
```

### Arguments

| | |
|---|---|
| this | the underlying object to proceed. |

### Value

a string

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### Examples

```
# just a method to access to the name of the underlying object or process

# Create an object of the 'mtkFeature' class.

f <- mtkFeature(name="x", type="double", val=0.0)

getName(f) # gives 'x'
```

## getNames-methods *The* getNames *method*

### Description

Returns the name of the factors managed by an object of class mtkExpFactors.

### Usage

```
getNames(this)
```

### Arguments

| | |
|---|---|
| this | an object of the class mtkExpFactors. |

### Value

a list of strings

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Get the names of the factors managed by all the factors
factors <- getNames(ishi.factors)
```

---

## getNominalValue-methods

*The* `getNominalValue` *method*

---

### Description

Returns the nominal value associated with the uncertainty domain of a factor.

### Usage

```
getNominalValue(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the class `mtkDomain`. |

### Value

ANY

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a domain and get the type of its nominal value
d <- mtkDomain(distributionName="unif", domainNominalValue=0.0)
mv <- getNominalValue(d)

# For more information, see examples for the mtkDomain or
# mtkFactor classes.
```

---

## getNominalValueType-methods

*The* `getNominalValueType` *method*

---

### Description

Returns the data type of the nominal value associated with the uncertainty domain of a factor.

### Usage

```
getNominalValueType(this)
```

### Arguments

| | |
|---|---|
| `this` | an object of the class `mtkDomain`. |

### Value

a string

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a domain and get the type of its nominal value
d <- mtkDomain(distributionName="unif", domainNominalValue=0.0)
valueType <- getNominalValueType(d)

# For more information, see examples for the mtkDomain or
# mtkFactor classes.
```

getParameters-methods
*The* getParameters *method*

## Description

Returns the vector of parameters and converts it to a named list.

## Usage

```
getParameters(this)
```

## Arguments

| | |
|---|---|
| this | the underlying object of class mtkProcess or its sub-classes. |

## Value

a named list in which each element corresponds to a parameter. The vector of parameters is converted into a named list such as (name of parameter 1 = value of parameter 1, name of parameter 2 = value of parameter 2, ...).

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a native designer avec the method "Morris"
# implemented in the package "mtk"

designer <- mtkNativeDesigner(design="Morris", information=list(size=20))

# Return the parameters as named list
getParameters(designer)
```

`getProcess-methods` *The* `getProcess` *method*

## Description

Gets a process from the workflow.

## Usage

```
getProcess(this,name)
```

## Arguments

| | |
|---|---|
| `this` | the underlying object of class `mtkExpWorkflow`. |
| `name` | a string from "design", "evaluate", or "analyze" to specify the process to fetch. |

## Value

an object of the class `mtkProcess`.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Build a workflow to do the sensitivity analysis for the model "Ishigami"
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))


designer <- mtkNativeDesigner("BasicMonteCarlo",
            information=list(size=20))
model <- mtkNativeEvaluator("Ishigami" )
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

ishiReg <- mtkExpWorkflow(expFactors=ishi.factors,
   processesVector=c( design=designer,
      evaluate=model,
      analyze=analyser)
```

```
   )
run(ishiReg)

# Extract the process "design" or "evaluate" from the workflow for other uses

designer <- getProcess(ishiReg, "design")
evaluator <- getProcess(ishiReg, "evaluate")
```

---

`getResult-methods` *The* `getResult` *method*

---

## Description

Returns the results produced by the process as an object of the class `mtkResult` or its sub-classes.

## Usage

```
getResult(this)
```

## Arguments

| | |
|---|---|
| `this` | the underlying object of class `mtkProcess` or its sub-classes |

## Details

1. Sub-class of the class `mtkProcess` returns objects of different sub-class of the class `mtkResult`. For instance, an object of the class `mtkDesigner` returns an object of the class `mtkDesignerResult`.
2. To fetch the results as a data.frame, please use the method `getData()`.

## Value

an object of the class `mtkResult`.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
```

```
     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
       distribPara=list(min=-pi, max=pi))

factors <- mtkExpFactors(list(x1,x2,x3))

# Builds the processes:
#   1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
      information=list(r=20,type="oat",levels=4,grid.jump=2))

#   2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method.
#      Here, it is the Morris method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the  workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and report the results.
run(exp1)

# Extracts the results produced by the analysis process as an objet of the class mtkAnaly

  getResult(getProcess(exp1, "analyze"))
```

---

`getType-methods` *The* `getType` *method*

---

## Description

Returns a string indicating the data type associated with the underlying object.

## Usage

```
getType(this)
```

## Arguments

| | |
|---|---|
| `this` | an object of the underlying class. |

## Value

a string

## Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))

# Return the data-type associated with the factor
t <- getType(x1)

# Create an object of the 'mtkFeature' class.

f <- mtkFeature(name="x", type="double", val=0.0)

# Return the data-type associated with the feature

getType(f) # gives 'double'
```

---

## getValue-methods *The* getValue *method*

---

### Description

Returns the name and the value managed by an object of the underlying class.

### Usage

```
getValue(this)
```

### Arguments

| | |
|---|---|
| this | an object of the class mtkValue or its sub-classes. |

### Value

a named variable

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create an object of the 'mtkValue'
v <- mtkValue(name="x", type="string", val="2.2")

# Fetch the value of the object as a named variable: x = "2.2"

getValue(v)
```

## getWeights-methods *The* getWeights *method*

### Description

Returns the weights of the discrete distribution associated with the factor's domain.

### Usage

```
getWeights(this)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class to proceed (mtkLevels and mtkDomain). |

### Value

a list of numeric values.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a discrete domain
x1 <- mtkDomain(distributionName="discrete", domainNominalValue=0,
 distributionParameters=list(type='categorical',
 levels = c(1,2,3,4,5), weights=rep(0.2, 5)))
# Returns the weights of the associated discrete distribution
getWeights(x1)
```

## is.finished-methods *The* is.finished *method*

### Description

Tests if the process has run and the results produced by the process are available.

### Usage

```
is.finished(this)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class mtkProcess |

### Value

TRUE or FALSE.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Build a workflow to do the sensitivity analysis for the model "Ishigami"
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))


designer <- mtkNativeDesigner("BasicMonteCarlo",
            information=list(size=20))

ishiReg <- mtkExpWorkflow(expFactors=ishi.factors,
   processesVector=c(design=designer)
   )
run(ishiReg)

# Extract the process "design" and test if it is correctly executed.

designer <- getProcess(ishiReg, "design")
is.finished(designer)
```

---

## is.ready-methods *The* `is.ready` *method*

---

### Description

Tests if the process is ready to run.

### Usage

```
is.ready(this)
```

### Arguments

| | |
|---|---|
| `this` | the underlying object of the class `mtkProcess` |

### Value

TRUE or FALSE.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
## This method is usually used only for the package's core programming!!!

# creates an experimental design with the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
     distribPara=list(min=-pi, max=pi))

factors <- mtkExpFactors(list(x1,x2,x3))

# Build the designer:

exp1.designer <- mtkNativeDesigner(design="Morris",
      information=list(r=20,type="oat",levels=4,grid.jump=2))

# Test if the process is ready to run

is.ready(exp1.designer)
```

---

## `Ishigami` *The* `Ishigami` *model*

---

### Description

The `Ishigami` model is an example evaluator implemented in the native `mtk`. It corresponds to the `Ishigami` function described in Saltelli et al., 2000. The behavior of the model is influenced by three factors `x1, x2, x3`.

### Usage

- mtkIshigamiEvaluator()
- mtkNativeEvaluator(model="Ishigami")
- mtkEvaluator(protocol = "R", site = "mtk", service = "Ishigami")

### Details

1. The implementation of the `Ishigami` model includes the object `Ishigami.factors` on the input factors and the class `mtkIshigamiEvaluator` to run the simulations.
2. In `mtk`, there are a few ways to build an evaluator of the `Ishigami` model, but we usually recommend the following class constructors: `mtkIshigamiEvaluator`, `mtkNativeEvaluator`.

### References

1. T. Ishigami and T. Homma (1990). An importance quantification technique in uncertainty analysis for computer models, *In:* International Symposium on Uncertainity Modelling and Analysis (ISUMA'90) (1990).
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
3. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### See Also

`help(Ishigami.factors),help(ishigami.fun, sensitivity)`

### Examples

```
### Run simulations of the "Ishigami" model
### for a random sample of input combinations

## Example I: by using the class constructor: mtkIshigamiEvaluator()

#
# Input the factors used in the "Ishigami" model
data(Ishigami.factors)

# Build the workflow:
#   1) specify the design process
exp1.designer <- mtkNativeDesigner(design = "BasicMonteCarlo",
 information = list(size=20) )

#   2) specify the evaluation process;
exp1.evaluator <- mtkIshigamiEvaluator()

#   3) specify the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
               processesVector = c(design=exp1.designer,
               evaluate=exp1.evaluator) )
# Run the workflow and report the results.
run(exp1)
```

```
print(exp1)

## Example II: by using the class constructor: mtkNativeEvaluator()

# Generate the Ishigami input factors
data(Ishigami.factors)

# Build the workflow:
#   1) specify the design process
exp1.designer <- mtkNativeDesigner(design = "BasicMonteCarlo",
information = list(size=20) )

#   2) specify the evaluation process;
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) specify the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
               processesVector = c(design=exp1.designer, evaluate=exp1.evaluator) )

# Run the workflow and report the results.
run(exp1)
print(exp1)

## Example III: by using the generic class constructor: mtkEvaluator()

# Generate the Ishigami input factors
data(Ishigami.factors)

# Build the workflow:
#   1) specify the design process
exp1.designer <- mtkNativeDesigner(
design = "BasicMonteCarlo", information = list(size=20) )

#   2) specify the evaluation process;
exp1.evaluator <- mtkEvaluator(protocol = "R", site = "mtk", service = "Ishigami")

#   3) specify the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
               processesVector = c(design=exp1.designer, evaluate=exp1.evaluator) )
# Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

## Ishigami.factors *Input factors of the* Ishigami *model*

---

### Description

The names and uncertainty distributions of the 3 input factors `x1, x2, x3` involved in the `Ishigami` function which is usually used as a model example for uncertainty and sensitivity analysis methods.

### Usage

```
data(Ishigami.factors)
```

### Format

an object of class mtkExpFactors.

### References

Saltelli, A., Chan, K., & Scott, E. M. (Eds.). (2000). Sensitivity analysis (Vol. 134). New York: Wiley.

### See Also

help(Ishigami), help(ishigami.fun,sensitivity)

### Examples

```
# The code used to generate the Ishigami.factors is as follows:

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))

Ishigami.factors <- mtkExpFactors(list(x1,x2,x3))

# To import the Ishigami.factors, just use the following line
data(Ishigami.factors)
```

---

## make.mtkFactor *The* make.mtkFactor *function*

---

### Description

Creates a new input factor and specifies its uncertainty distribution.

### Usage

```
make.mtkFactor(name="unkown", id="unkown", unit="", type="",
 nominal=NA, distribName='unknown', distribPara=list(), features=list())
```

### Arguments

| | |
|---|---|
| name | the name of the input factor. |
| id | the name of the factor in the simulation code, if different from name (optional). |
| unit | the measurement unit of the factor values (optional). This can be used in graphics or reports, for example. |
| type | the data-type of the factor's values (optional). |
| nominal | the nominal value of the factor. |
| distribName | the name of the probability distribution describing the factor's uncertainty. |
| distribPara | the list of distribution parameters. |
| features | the list of factor's features. |

### Details

The `distribName` argument must use the R terminology, for example `norm` for the normal distribution or `unif` for the uniform one; see `help(distributions)`.

### Value

an object of class `mtkFactor`.

### Author(s)

Juhui WANG, MIA-jouy, INRA, Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod

### Examples

```
# Define a new continuous factor
make.mtkFactor("A", distribName="unif", distribPara=list(min=0,max=1))
# Define a new discrete factor
make.mtkFactor("D", distribName="discrete", distribPara =
list(type='categorical', levels=c('a','b','c'),
weights=rep(0.33,3))
)
```

---

## `make.mtkFeatureList`

*The* `make.mtkFeatureList` *function*

---

### Description

Creates a list of `mtkFeature` elements from a simple named list.

### Usage

```
make.mtkFeatureList(x=list())
```

### Arguments

| | |
|---|---|
| `x` | a named list. |

### Value

a list of objects from the class `mtkFeature`.

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a list of mtkFeature for the Features: min, max, shape.
make.mtkFeatureList(list(min=-1,max=+1,shape="square"))
```

---

`make.mtkParameterList` *The* `make.mtkParameterList` *function*

---

### Description

Creates a list of `mtkParameter` elements from a simple named list.

### Usage

```
make.mtkParameterList(x=list())
```

### Arguments

| | |
|---|---|
| `x` | a named list. |

### Value

a list of objects from the class `mtkParameter`.

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a list of mtkParameter from a named list for the parameters: min, max, shape.
make.mtkParameterList(list(min=-1,max=+1,shape="hello"))
```

---

`Morris` *The* `Morris` *method*

---

### Description

A `mtk` compliant implementation of the `morris` method for experiments design and sensitivity analysis.

### Usage

- mtkMorrisDesigner(listParameters = NULL)
- mtkNativeDesigner(design="Morris", information=NULL)
- mtkMorrisAnalyser(listParameters = NULL)
- mtkNativeAnalyser(analyze="Morris", information=NULL)

### Parameters

`r`: the number of trajectories or a pair `(r1,r2)` if the version due to Campolongo et al. (2007) is used.

`type`: the type of design (either `oat` or `simplex`).

`levels`: the number of levels per factor (if `type = "oat"`).

`grid.jump`: the length of the steps within the trajectories (if `type = "oat"`).

`scale.factor`: a numeric value, the homothety factor of the (isometric) simplexes (if `type = "simplex"`).

`scale`: logical. If TRUE, the input design of experiments is scaled before computing the elementary effects so that all factors vary within the range [0,1].

`shrink`: a scalar or a vector of scalars between 0 and 1, specifying shrinkage to be used on the probabilities before calculating the quantiles.

### Details

1. The `mtk` implementation uses the `morris` function of the `sensitivity` package. For further details on the arguments and the behavior, see `help(morris, sensitivity)`.
2. The `mtk` implementation of the `Morris` method includes the following classes:

   `mtkMorrisDesigner`: for the `Morris` design processes.

   `mtkMorrisAnalyser`: for `Morris` analysis processes.

   `mtkMorrisDesignerResult`: to store and manage the design.

   `mtkMorrisAnalyserResult`: to store and manage the analysis results.
3. Many ways to create a `Morris` designer are available in `mtk`, but we recommend the following class constructors: `mtkMorrisDesigner` or `mtkNativeDesigner`.
4. Many ways to create a `Morris` analyser are available in `mtk`, but we recommend the following class constructors: `mtkMorrisAnalyser` or `mtkNativeAnalyser`.
5. The method `Morris` is usually used both to build the experiment design and to carry out the sensitivity analysis. In such case, we can use the `mtkDefaultAnalyser` instead of naming explicitly the method for sensitivity analysis (see example III in the examples section)

### References

1. Campolongo, F., J. Cariboni, and A. Saltelli, 2007. An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. Saltelli A., Chan K.and Scott E. M., 2000. Sensitivity Analysis. Wiley, New York
3. Pujol G., 2009, Simplex-based screening designs for estimating metamodels, Reliability Engineering and System Safety 94, 1156–1160.

### See Also

`help(morris, sensitivity)`

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Morris" method

# Example I: by using the class constructors: mtkMorrisDesigner() and mtkMorrisAnalyser()

# Generate the factors
```

```
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkMorrisDesigner(
listParameters = list(r=20, type="oat",
    levels=4, grid.jump=2)
)

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
            processesVector = c(design=exp1.designer,
           evaluate=exp1.evaluator,
         analyze=exp1.analyser)
         )

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
plot(exp1)
#       plot3d.morris(extractData(exp1, name="analyze"))

##  Example II:  by using the class constructors: mtkNativeDesigner() and mtkMorrisAnalys

# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkNativeDesigner(design = "Morris",
       information = list(r=20, type="oat",
        levels=4, grid.jump=2)
         )

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method
exp1.analyser <- mtkMorrisAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
     evaluate=exp1.evaluator,
     analyze=exp1.analyser)
       )
```

```
#  Run the workflow and reports the results.
run(exp1)
print(exp1)


##  Example III:  by using the class constructors: mtkMorrisDesigner() and mtkDefaultAnal

# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkMorrisDesigner( listParameters =
list(r=20, type="oat",
levels=4, grid.jump=2))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method
exp1.analyser <- mtkDefaultAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
     evaluate=exp1.evaluator,
     analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
```

---

`mtk.analyserAddons` *The* `mtk.analyserAddons` *function*

---

## Description

A function used to extend the "mtk" package with new analysis methods programmed as R functions. The `mtk.analyserAddons` function takes a R file as input and converts it into a `mtk` compliant class which can be seamlessly integrated into the `mtk` package.

## Usage

```
mtk.analyserAddons(where = NULL, library = NULL,
  authors = NULL, name = NULL,
                           main = NULL,
                           summary = NULL,
                           plot = NULL,
                           print = NULL)
```

### Arguments

| | |
|---|---|
| where | NULL or a file holding the R function to convert. |
| library | NULL or the name of the library if the R function to convert is held in a library. |
| authors | NULL or information about the authors of the R function. |
| name | a string to name the method when used with the "mtk" package. |
| main | the R function which implements the method. |
| summary | NULL or a subversion of the summary function provided with the method. |
| plot | NULL or a reprogrammed version of the plot function provided with the method. |
| print | NULL or a reprogrammed version of the print function provided with the method. |

### Details

The new method must be programmed according to the following syntax:

`main <- function(X, Y, ...)` where `X` is a data.frame holding the experiment design, and `Y` is a data.frame holding the results produced by the model simulation.

The function `main` returns a named list with two elements: `main` and `information`. The element `main` holds the result of the sensitivity analysis and the element `information` is optional, may be used to give supplementary information about the analysis process and the produced results.

Furthermore, in order to report the analysis results more precisely, users can redefine the generic functions: `summary (object, ...)`,`plot(x,y, ...)`,`print(x, ...)`.

### Value

invisble()

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# an example implementation of the method "Regression"
# called here "RegressionTest" is held  in the file
#  "inst/extdata/regressionSI.R"

rFile <- "regressionSI.R"
rFile <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",rFile,sep = "")


# to convert the method "RegressionTest" to S4 classes
# compliant with the "mtk" package. The generated "mtk" compliant class
```

```
# is called "mtkXXXAnalyser.R" where XXX corresponds to the name of the method.

mtk.analyserAddons(where=rFile, authors="H. Monod,INRA",
 name="RegressionTest",
   main="regressionSI", print="print.regressionSI",
       plot="plot.regressionSI")

# To use the method "RegressionTest" with the package "mtk",
# just source the generated new files

source("mtkRegressionTestAnalyser.R")

## Use the method "RegressionTest" to do sensitivity analysis

# 1) Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

#  2) Create a workflow with the "Ishigami" model and analyze it with the new method
ishiReg <- mtkExperiment(expFactors=ishi.factors,
design="BasicMonteCarlo",designInfo=list(size=20),
model="Ishigami",
analyze="RegressionTest",
)
#  3) Run the workflow and report the results
run(ishiReg)
summary(ishiReg)
```

---

`mtk.designerAddons` *The* `mtk.designerAddons` *function*

---

## Description

A function used to extend the `mtk` package with new design methods programmed as R functions. The `mtk.designerAddons` function takes a R file as input and converts it into a `mtk` compliant class which can be seamlessly integrated into the `mtk` package.

## Usage

```
mtk.designerAddons(where = NULL, library = NULL,
  authors = NULL, name = NULL,
  main = NULL, summary = NULL,
  plot = NULL, print = NULL)
```

## Arguments

| | |
|---|---|
| `where` | NULL or the file containing the R functions to convert into native `mtk` methods. |
| `library` | NULL or the name of the package if the R function to convert is included in a package. |

| | |
|---|---|
| authors | NULL or information about the authors of the R function. |
| name | a string to name the method when used with the mtk package. |
| main | the name of the R function implementing the designer. |
| summary | NULL or a special version of the summary function provided in the file where. |
| plot | NULL or a special version of the plot function provided in the file where. |
| print | NULL or a special version of the print function provided in the file where. |

## Details

The main function must have the following syntax:

main <- function(factors, distribNames, distribParameters, ...)

where factors is either a number or a list of strings giving the names of the n input factors, distribNames is a list of string giving the names of the n probability distributions that describe the factors' uncertainty, and distribParameters is a list of n lists specifying the distribution parameters associated with the uncertainty domains.

The R function main returns a named list with two elements: the element main is a data.frame containing the generated experiment design and the element information is an optional list that may be used to provide complementary information about the design process and results.

Furthermore, in order to give more advanced data reporting mechanism with the new method, users can redefine the generic functions:

summary(object, ...),plot(x,y, ...),print(x, ...)

## Value

invisible()

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# an example implementation of the method "MC" is held  in the file
# "inst/extdata/montecarloDesigner.R"

rFile <- "montecarloDesigner.R"
rFile <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",rFile,sep = "")

# to convert this special version of the method "MC"
# to S4 classes compliant with the "mtk" package. The generated "mtk" compliant class
# is called "mtkXXXDesigner.R" where XXX corresponds to the name of the method.
mtk.designerAddons(where=rFile, authors="H. Monod,INRA", name="MC",
 main="basicMonteCarlo")
```

```
# to use the method "MC" with the package "mtk",
# just source the generated new files

source("mtkMCDesigner.R")

## Use the "mtkMCDesigner" with the "mtk" package in a seamless way:

# 1) Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# 2) Specify a new workflow with the new method
ishiReg <- mtkExperiment(expFactors=ishi.factors,design="MC",
model="Ishigami", analyze="Regression",
    designInfo=list(size=20))

# 3) Run the workflow and report the results
run(ishiReg)
summary(ishiReg)
```

---

`mtk.evaluatorAddons`

*The* `mtk.evaluatorAddons` *function*

---

## Description

A function used to extend the "mtk" package with new models programmed as R functions. The `mtk.evaluatorAddons` function takes a R file as input and converts it into a `mtk` compliant class which can be seamlessly integrated into the `mtk` package.

## Usage

```
mtk.evaluatorAddons(where = NULL, library = NULL,
authors = NULL, name = NULL, main = NULL,
summary = NULL, plot = NULL,
print = NULL)
```

## Arguments

| | |
|---|---|
| `where` | NULL or a file holding the R function to convert. |
| `library` | NULL or the name of the library if the R function to convert is held in a library. |
| `authors` | NULL or information about the authors of the R function. |
| `name` | a string to name the model when used with the "mtk" package. |
| `main` | the R function which implements the model. |
| `summary` | NULL or a special version of the "summary" function provided with the model. |

| | |
|---|---|
| plot | NULL or a special version of the "plot" function provided with the model. |
| print | NULL or a special version of the "print" function provided with the model. |

## Details

The new model must be programmed according to the following syntax:

`main <- function(X, ...)` where `X` is a data.frame holding the experiment design used to run the model simulation.

The function `main` returns a named list with two elements: `main` and `information`. The element `main` holds the result of the model simulation and the element `information` is optional, may be used to give supplementary information about the simulation process and its results.

Furthermore, users can redefine the following generic functions to report the results more precisely:

`summary (object, ...),plot(x,y, ...),print(x, ...).`

## Value

invisble()

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# an example implementation of the model "WW" is held
# in the file "inst/extdata/wwdm.R"

rFile <- "wwdm.R"
rFile <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",rFile,sep = "")

# to covert the model "WW" to a S4 classes compliant with the "mtk" package.
# The generated "mtk" compliant class is called "mtkXXXEvaluator.R" where XXX corresponds
# to the name of the model.

mtk.evaluatorAddons(where=rFile, authors="H. Monod,INRA", name="WW", main="wwdm.simule")

# to use the model evaluator "WW" with the package "mtk",
# just source the generated new files

source("mtkWWEvaluator.R")

## Use the "mtkWWEvaluator" with the "mtk" package in a seamless way:

# 1) Define the factors

Eb <- make.mtkFactor(name="Eb", distribName="unif",
```

```
       nominal=1.85, distribPara=list(min=0.9, max=2.8))
Eimax <- make.mtkFactor(name="Eimax", distribName="unif",
       nominal=0.94, distribPara=list(min=0.9, max=0.99))
K <- make.mtkFactor(name="K", distribName="unif", nominal=0.7,
       distribPara=list(min=0.6, max=0.8))
Lmax <- make.mtkFactor(name="Lmax", distribName="unif", nominal=7.5,
       distribPara=list(min=3, max=12))
A <- make.mtkFactor(name="A", distribName="unif", nominal=0.0065,
       distribPara=list(min=0.0035, max=0.01))
B <- make.mtkFactor(name="B", distribName="unif", nominal=0.00205,
       distribPara=list(min=0.0011, max=0.0025))
TI <- make.mtkFactor(name="TI", distribName="unif", nominal=900,
       distribPara=list(min=700, max=1100))

WW.factors <- mtkExpFactors(list(Eb,Eimax,K,Lmax,A,B,TI))

# 2) Build a workflow  for the "WW" model

exp <- mtkExperiment(expFactors=WW.factors,
design="Morris",designInfo=list(type="oat",
r=10, levels=5, grid.jump=3),
model="WW", modelInfo=list(year=3),
analyze="Morris", analyzeInfo=list(type="oat",
r=10, levels=5, grid.jump=3))

## 3) Run the workflow and reports the results

run(exp)
summary(exp)
```

---

`mtkAnalyser` *The constructor of the class* `mtkAnalyser`

---

## Description

The constructor

## Usage

```
 mtkAnalyser(protocol = "R", site = "mtk", service = "",
parameters= NULL, parametersList = NULL, ready = TRUE,
state = FALSE, result = NULL)
```

## Arguments

| | |
|---|---|
| `protocol` | a string from "http", "system", "R" respectively representing if the process is implemented remotely, locally or as R function. |
| `site` | the site where the process is implemented if remotely or the package where the process is implemented if as a R function. |
| `service` | a string corresponding to the name of the method implemented in the package "mtk" or the service that implements the process if remotely. |

`parameters` a vector of [`mtkParameter`] representing the parameters necessary to run the process.

`parametersList` a named list containing the parameters necessary to run the process. It gives another way to specify the parameters.

`ready` a logical to indicate if the process is ready to run.

`state` a logical to indicate if the process finished running and the results are available.

`result` an object of a class derived from [`mtkAnalyserResult`] to hold the results produced by the analyser.

### Value

an object of the `mtkAnalyser` class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Creates an analyser avec the method "Morris" implemented
# in the package "mtk"
analyser <- mtkAnalyser(service="Morris",
parametersList=list(nboot=20))
```

---

## `mtkAnalyser-class` *The* `mtkAnalyser` *class*

---

### Description

The `mtkAnalyser` class is a sub-class of the class `mtkProcess` used to manage the sensitivity analysis process. It provides all the slots and methods defined in the class `mtkProcess`.

### Class Hierarchy

**Parent classes :** `mtkProcess`

**Direct Known Subclasses :** `mtkNativeAnalyser`,`mtkMorrisAnalyser`, etc.

### Constructor

`mtkAnalyser` signature(protocol="R", site="mtk", service="", parameters=NULL, parametersList=NULL, ready=TRUE, state=FALSE, result=NULL)

## Slots

`name`: (`character`) a string to name the processing type. Here, it always takes "analyze".

`protocol`: (`character`) a string to name the protocol used to run the process: http, system, R, etc.

`site`: (`character`) a string to indicate where the service is located.

`service`: (`character`) a string to name the method or the service (if remotely) to invoke.

`parameters`: (`vector`) a vector of `mtkParameter` containing the parameters to pass while calling the service.

`ready`: (`logical`) a logical to tell if the process is ready to run.

`state`: (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`: (`ANY`) NULL or an object of the class `mtkAnalyserResult` to hold the results produced by the process

## Methods

`setName` signature(this = "mtkAnalyser", name = "character"): Not used, just inherited from the parent class.

`setParameters` signature(this = "mtkAnalyser", f = "vector"): Assigns a new vector of parameters to the process.

`getParameters` signature(this = "mtkAnalyser"): Returns the parameters as a named list.

`is.ready` signature( = "mtkAnalyser"): Tests if the process is ready to run.

`setReady` signature(this = "mtkAnalyser", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkAnalyser"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkAnalyser", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkAnalyser"): Returns the results produced by the process as a `mtkAnalyserResult`.

`getData` signature(this = "mtkAnalyser"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkAnalyser"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkAnalyser", context= "mtkExpWorkflow"): Runs the sensitivity analysis on the model defined in the context.

`summary` signature(object = "mtkAnalyser"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkAnalyser"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkAnalyser"): Builds a plot of the results produced by the process.

`report` signature(this = "mtkAnalyser"): Reports the results produced by the process.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Creates an analyser avec the method "Morris"
# implemented in the package "mtk".

analyser <- mtkAnalyser(service="Morris",
 parametersList=list(nboot=20))
```

---

# mtkAnalyserResult *The constructor of the class* mtkAnalyserResult

---

## Description

The constructor

## Usage

```
mtkAnalyserResult(main = data.frame(), information = list())
```

## Arguments

| | |
|---|---|
| main | a data.frame to hold the results produced with the analyser. |
| information | a named list containing optional information about the managed data and process. |

## Value

an object of the mtkAnalyserResult class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Construct an object of the \code{mtkAnalyserResult} class
# from a data.frame.
data <- data.frame()
result <- mtkAnalyserResult(main=data,
information = list(method="Morris", model="Ishigami"))
```

---

## mtkAnalyserResult-class

*The* mtkAnalyserResult *class*

---

### Description

A class to manage the results produced by the sensitivity analysis process.

### Class Hierarchy

**Parent classes :** mtkResult

**Direct Known Subclasses :** mtkMorrisAnalyserResult, mtkPLMMAnalyserResult, etc.

### Constructor

{mtkAnalyserResult} signature(main = data.frame(), information = list())

### Slots

main**:** (data.frame) a data.frame to hold the analysis results produced with the analyser.

information**:** (list) a named list containing optional information about the managed data and process.

### Methods

summary signature(object = "mtkAnalyserResult"): Provides a summary of the analysis results produced with the analyser.

print signature(x = "mtkAnalyserResult"): Prints a report of the analysis results produced with the analyser.

plot signature(x = "mtkAnalyserResult"): Plots the analysis results produced with the analyser.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Construct an object of the \code{mtkAnalyserResult} class
# from a data.frame.
data <- data.frame()
result <- mtkAnalyserResult(main=data, information
= list(method="Morris", model="Ishigami"))
```

---

## mtkBasicMonteCarloDesigner

*The constructor of the class* `mtkBasicMonteCarloDesigner`

---

### Description

The constructor

### Usage

```
mtkBasicMonteCarloDesigner(mtkParameters = NULL,
 listParameters = NULL)
```

### Arguments

`mtkParameters`
: a vector of `mtkParameter` representing the parameters necessary to run the process.

`listParameters`
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the `mtkBasicMonteCarloDesigner` class

### Details

See the `BasicMonteCarlo` method with `help(BasicMonteCarlo)`

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# see examples with help(BasicMonteCarlo)
```

---

`mtkBasicMonteCarloDesigner-class`
*The* `mtkBasicMonteCarloDesigner` *class*

---

### Description

The `mtkBasicMonteCarloDesigner` class is a sub-class of the class `mtkDesigner`. It implements the `BasicMonteCarlo` method for experiments design and provides all the slots and methods defined in the class `mtkDesigner`.

### Class Hierarchy

**Parent classes :** `mtkDesigner`

**Direct Known Subclasses :**

### Constructor

**mtkBasicMonteCarloDesigner** signature(mtkParameters = NULL, listParameters = NULL)

### Slots

`name`**:** (`character`) always takes the string "design".

`protocol`**:** (`character`) always takes the string "R".

`site`**:** (`character`) always takes the string "mtk".

`service`**:** (`character`) always takes the string "BasicMonteCarlo".

`parameters`**:** (`vector`) a vector of [`mtkParameter`] containing the parameters to pass while calling the service.

`ready`**:** (`logical`) a logical to tell if the process is ready to run.

`state`**:** (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`**:** (`ANY`) a data holder to hold the results produced by the process

### Methods

**setName** signature(this = "mtkBasicMonteCarloDesigner", name = "character"): Method inherited from the parent class.

**setParameters** signature(this = "mtkBasicMonteCarloDesigner", f = "vector"): Assigns new parameters to the process.

**getParameters** signature(this = "mtkBasicMonteCarloDesigner"): Returns the parameters as a named list.

**is.ready** signature( = "mtkBasicMonteCarloDesigner"): Tests if the process is ready to run.

**setReady** signature(this = "mtkBasicMonteCarloDesigner", switch = "logical"): Makes the process ready to run.

**is.ready** signature( = "mtkBasicMonteCarloDesigner"): Tests if the results produced by the process are available.

**setReady** signature(this = "mtkBasicMonteCarloDesigner", switch = "logical"): Marks the process as already executed.

**getResult** signature(this = "mtkBasicMonteCarloDesigner"): Returns the results produced by the process as a [mtkBasicMonteCarloDesignerResult].

**getData** signature(this = "mtkBasicMonteCarloDesigner"): Returns the results produced by the process as a data.frame.

**serializeOn** signature(this = "mtkBasicMonteCarloDesigner"): Returns all data managed by the process as a named list.

**run** signature(this = "mtkBasicMonteCarloDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

**summary** signature(object = "mtkBasicMonteCarloDesigner"): Provides a summary of the results produced by the process.

**print** signature(x = "mtkBasicMonteCarloDesigner"): Prints a report of the results produced by the process.

**plot** signature(x = "mtkBasicMonteCarloDesigner"): Plots the results produced by the process.

**report** signature(this = "mtkBasicMonteCarloDesigner"): Reports the results produced by the process.

## Details

See the `BasicMonteCarlo` method with `help(BasicMonteCarlo)`

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# See examples from  the BasicMontecarlo method:  help(basicMonteCarlo)
```

`mtkBasicMonteCarloDesignerResult`
*The constructor of class* `mtkBasicMonteCarloDesignerResult`

## Description

The constructor

## Usage

```
mtkBasicMonteCarloDesignerResult(main,information=NULL)
```

## Arguments

| | |
|---|---|
| `main` | a data.frame holding the experimental design produced by the designer. |
| `information` | a named list containing the information about the managed data and the underlying process. |

## Value

an object of the `mtkBasicMonteCarloDesignerResult` class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples with help(BasicMonteCarlo)
```

mtkBasicMonteCarloDesignerResult-class
*The* mtkBasicMonteCarloDesignerResult *class*

## Description

A class to collect the experimental design produced by the designer implementing the method BasicMonteCarlo.

## Class Hierarchy

**Parent classes :** mtkDesignerResult

**Direct Known Subclasses :**

## Constructor

mtkBasicMonteCarloDesignerResult signature(main,information=NULL)

## Slots

main**:** (data.frame) a data-frame holding the experimental design.

information**:** (list) a named list containing optional information about the managed data or the underlying process.

## Methods

summary signature(object = "mtkBasicMonteCarloDesignerResult"): Provides a summary of the experimental design produced by the designer.

print signature(x = "mtkBasicMonteCarloDesignerResult"): Prints a report of the experimental design produced by the designer.

plot signature(x = "mtkBasicMonteCarloDesignerResult"): Plots the experimental design produced by the designer.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples with help(BasicMonteCarlo)
```

`mtkDefaultAnalyser` *The constructor of the class* `mtkDefaultAnalyser`

## Description

This class is used when both the experimental design and the sensitivity analysis are fulfilled with the same method.

## Usage

```
mtkDefaultAnalyser()
```

## Value

an object of the `mtkDefaultAnalyser` class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# creates a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
     distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
#   1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
      information=list(r=20,type="oat",levels=4,grid.jump=2))

#   2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method.
#      Here, it is the "Morris" method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the  workflow with the processes defined previously.
```

```
exp1 <- mtkExpWorkflow(expFactors=factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

`mtkDefaultAnalyser-class`
*The* `mtkDefaultAnalyser` *class*

---

## Description

The `mtkDefaultAnalyser` class is a sub-class of the class `mtkAnalyser`. It provides all the slots and methods defined in the class `mtkAnalyser`. The `mtkDefaultAnalyser` class is used when the method used for the sensitivity analysis is the same as the method used for the experiment design.

## Class Hierarchy

**Parent classes :** `mtkAnalyser`

**Direct Known Subclasses :**

## Constructor

`mtkDefaultAnalyser` signature()

## Slots

`name`**:** (`character`) always takes the string "analyze".

`protocol`**:** (`character`) a string to name the protocol used to run the process: http, system, R, etc.

`site`**:** (`character`) a string to indicate where the service is located.

`service`**:** (`character`) a string to name the service to invoke.

`parameters`**:** (`vector`) a vector of [`mtkParameter`] containing the parameters to pass while calling the service.

`ready`**:** (`logical`) a logical to tell if the process is ready to run.

`state`**:** (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`**:** (`ANY`) a data holder to hold the results produced by the process

## Methods

**setName** signature(this = "mtkDefaultAnalyser", name = "character"): Not used, method inherited from the parent class.

**setParameters** signature(this = "mtkDefaultAnalyser", f = "vector"): Assigns new parameters to the process.

**getParameters** signature(this = "mtkDefaultAnalyser"): Returns the parameters as a named list.

**is.ready** signature( = "mtkDefaultAnalyser"): Tests if the process is ready to run.

**setReady** signature(this = "mtkDefaultAnalyser", switch = "logical"): Makes the process ready to run.

**is.ready** signature( = "mtkDefaultAnalyser"): Tests if the results produced by the process are available.

**setReady** signature(this = "mtkDefaultAnalyser", switch = "logical"): Marks the process as already executed.

**getResult** signature(this = "mtkDefaultAnalyser"): Returns the results produced by the process as a `mtkAnalyserResult`.

**getData** signature(this = "mtkDefaultAnalyser"): Returns the results produced by the process as a data.frame.

**serializeOn** signature(this = "mtkDefaultAnalyser"): Returns all data managed by the process as a named list.

**run** signature(this = "mtkDefaultAnalyser", context= "mtkExpWorkflow"): Runs the sensitivity analysis defined in the context.

**summary** signature(object = "mtkDefaultAnalyser"): Provides a summary of the results produced by the process.

**print** signature(x = "mtkDefaultAnalyser"): Prints a report of the results produced by the process.

**plot** signature(x = "mtkDefaultAnalyser"): Reports graphically the results produced by the process.

**report** signature(this = "mtkDefaultAnalyser"): Reports the results produced by the process.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))
# Build the processes:
#   1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design = "Morris",
 information=list(r=20,type="oat",levels=4,grid.jump=2))

#   2) the model simulation process with the model "Ishigami".
```

```
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method.
#      Here, it is the "Morris" method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the  workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

`mtkDesigner` *The constructor of the class* `mtkDesigner`

---

## Description

The constructor

## Usage

```
mtkDesigner(protocol = "R", site = "mtk", service = "",
  parameters = NULL, parametersList = NULL, ready = TRUE,
  state = FALSE, result = NULL)
```

## Arguments

| | |
|---|---|
| `protocol` | (`character`) a string from "http", "system", "R" respectively representing if the process is implemented remotely, locally or as R function. |
| `site` | (`character`) a string to indicate where the service is located. |
| `service` | (`character`) a string to name the method or the service (if remotely) to invoke. |
| `parameters` | a vector of [`mtkParameter`] representing the parameters necessary to run the process. |
| `parametersList` | a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters. |
| `ready` | a logical to indicate if the process is ready to run. |
| `state` | a logical to indicate if the process finished running and the results are available. |
| `result` | an object of a class derived from [`mtkDesignerResult`] to hold the results produced by the designer. |

## Value

an object of the `mtkDesigner` class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a designer with the method "Morris"
# implemented in the package "mtk"
designer <- mtkDesigner(service="Morris",
parametersList=list(nboot=20))
```

---

# mtkDesigner-class *The* mtkDesigner *class*

---

## Description

The mtkDesigner class is a sub-class of the class mtkProcess used to manage the experiments design task. It provides all the slots and methods defined in the class mtkProcess.

## Class Hierarchy

**Parent classes :** mtkProcess

**Direct Known Subclasses :** mtkNativeDesigner,mtkMorrisDesigner, etc.

## Constructor

**mtkDesigner** signature(protocol = "R", site = "mtk", service = "", parameters = NULL, parametersList = NULL, ready = TRUE, state = FALSE, result = NULL)

## Slots

name**:** (character) always takes the string "design".

protocol**:** (character) a string to name the protocol used to run the process: http, system, R, etc.

**site** the site where the process is implemented if remotely or the package where the process is implemented if as a R function.

**service** a string corresponding to the name of the method implemented in the package "mtk" or the service that implements the process if remotely.

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder from the class mtkDesignerResult to hold the results produced by the process.

## Methods

**setName** signature(this = "mtkDesigner", name = "character"): Not used, method inherited from the parent class.

**setParameters** signature(this = "mtkDesigner", f = "vector"): Assigns new parameters to the process.

**getParameters** signature(this = "mtkDesigner"): Returns the parameters as a named list.

**is.ready** signature( = "mtkDesigner"): Tests if the process is ready to run.

**setReady** signature(this = "mtkDesigner", switch = "logical"): Makes the process ready to run.

**is.ready** signature( = "mtkDesigner"): Tests if the results produced by the process are available.

**setReady** signature(this = "mtkDesigner", switch = "logical"): Marks the process as already executed.

**getResult** signature(this = "mtkDesigner"): Returns the results produced by the process as mtkDesignerResult.

**getData** signature(this = "mtkDesigner"): Returns the results as a data.frame.

**serializeOn** signature(this = "mtkDesigner"): Returns all data managed by the process as a named list.

**run** signature(this = "mtkDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

**summary** signature(object = "mtkDesigner"): Provides a summary of the results produced by the process.

**print** signature(x = "mtkDesigner"): Prints a report of the results produced by the process.

**plot** signature(x = "mtkDesigner"): Reports graphically the results produced by the process.

**report** signature(this = "mtkDesigner"): Reports the results produced by the process.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a designer with the method "Morris"
# implemented in the package "mtk"
designer <- mtkDesigner(service="Morris",
 parametersList=list(nboot=20))
```

## mtkDesignerResult *The constructor of the class* mtkDesignerResult

### Description

The constructor

### Usage

```
mtkDesignerResult(main=data.frame(),information=list())
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the experimental design produced by the designer. |
| information | a named list containing the information about the experiments design. |

### Value

an object of the mtkDesignerResult class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Construct an object of the \code{mtkDesignerResult}
# class from a data-frame.
data <- data.frame()
expDesign <- mtkDesignerResult(main=data,
   information = list(sampling="Fast"))
```

## mtkDesignerResult-class *The* mtkDesignerResult *class*

### Description

A class to collect the experimental design produced by an experiments design process.

### Class Hierarchy

**Parent classes :** mtkResult

**Direct Known Subclasses :** mtkSobolDesignerResult, mtkMorrisDesignerResult, etc.

### Constructor

mtkDesignerResult signature(main=data.frame(),information=list())

### Slots

main: (data.frame) a data.frame holding the experimental design produced by the process.

information: (list) a named list containing optional information about the experiments design.

### Methods

summary signature(object = "mtkDesignerResult"): Provides a summary of the experimental design produced by the design process.

print signature(x = "mtkDesignerResult"): Prints a report of the experimental design produced by the design process.

plot signature(x = "mtkDesignerResult"): Plots the experimental design produced by the design process.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Construct an object of the mtkDesignerResult class from a data-frame.
data <- data.frame()
expDesign <- mtkDesignerResult(main=data,
   information = list(sampling="Fast"))
```

---

## mtkDomain *The constructor of the class* mtkDomain

---

### Description

The constructor of the class mtkDomain.

### Usage

```
mtkDomain(distributionName="unknown",domainNominalValue=0,
    distributionParameters=list())
```

### Arguments

distributionName
: a string corresponding to the distribution name associated with the domain.

domainNominalValue
: an object of the mtkValue class or information allowing to create an object of the mtkValue class, used to hold the nominal value of the domain.

distributionParameters
: a list to hold the parameters of the distribution associated with the domain.

### Value

an object of the mtkDomain class

### Examples

```
# creates a  new domain with a continue distribution
d <- mtkDomain(distributionName="unif", domainNominalValue=0,
distributionParameters = list(max=3, min=0))

# creates a  new domain with a discrete distribution
d <- mtkDomain(distributionName="discrete", domainNominalValue=3,
distributionParameters = list(type='categorical',
  levels = c(1,2,3,4,5), weights=rep(0.2, 5)))
```

---

## mtkDomain-class *The* mtkDomain *class*

---

### Description

The mtkDomain class is a class used to manage the uncertainty domain associated with a factor.

### Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :**

### Constructor

**mtkDomain** signature(distributionName = "unknown", domainNominalValue = 0, distributionParameters = list())

### Slots

distributionName**:** (character) a string representing the distribution law.

nominalValue**:** (mtkValue) the nominal value of the domain.

levels**:** (mtkLevels) an object of mtkLevels class.

distributionParameters**:** (list) a list of mtkParameter objects.

## Methods

`initialize` signature(.Object = "mtkDomain"): The initializer of the class `mtkDomain`.

`getDistributionName` signature(this = "mtkDomain"): Returns the distribution's name.

`getNominalValue` signature(this = "mtkDomain"): Returns the the nominal value.

`getNominalValueType` signature(this = "mtkDomain"): Returns the value type of the nominal value .

`getDiscreteDistributionType` signature(this = "mtkDomain"): Returns the type of the discrete distribution.

`getLevels` signature(this="mtkDomain"): Fetches the the levels managed by the domain.

`getWeights` signature(this="mtkDomain"): Fetches the the weights managed by the domain.

`getDistributionParameters` signature(this = "mtkDomain"): Fetches the parameters of the distributions associated with the domain.

`setLevels` signature(this="mtkDomain", levels = "vector"): Affects a new level to the domain where `levels` is a named list like `list(type='categorical', levels=c(1,2,3,4,5), weights`

`setLevels` signature(this="mtkDomain", levels = "mtkLevels"): Affects a new level to the domain where `levels` is an object from the class `mtkLevels`.

`setDistributionParameters` signature(this = "mtkDomain", aDistParamList="list"): Affects a new list of parameters to the domain. For continue distributions, `aDistParamList` may be a list of objects of the class `mtkParameter` or a named list like `list(max=5, min=1)`., For discrete distributions, `aDistParamList` may be a named list containing an object of the class `mtkLevels` or a named list like `list(type='categorical',levels = c(1,2,3,4,5), weig` from which we can build an object of the class `mtkLevels`.

`print` signature(x = "mtkDomain"): Prints the data managed by the domain.

`show` signature(object = "mtkDomain"): Displays the underlying object of the class `mtkDomain`.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a  new domain with a continue distribution
d <- mtkDomain(distributionName="unif", domainNominalValue=0,
distributionParameters = list(max=3, min=0))

# Create a  new domain with a discrete distribution
d <- mtkDomain(distributionName="discrete", domainNominalValue=3,
 distributionParameters = list(type='categorical',
 levels = c(1,2,3,4,5), weights=rep(0.2, 5)))
# Change the levels to list(type='categorical', levels = c('a','b','c','d'), weights=rep(
setLevels(d, list(type='categorical', levels = c('a','b','c','d'), weights=rep(0.25, 4)))
```

mtkEvaluator *The constructor of the class* mtkEvaluator

## Description

The constructor

## Usage

```
mtkEvaluator(protocol = "R", site = "mtk", service = "",
  parameters = NULL, parametersList = NULL, ready = TRUE,
   state = FALSE, result = NULL)
```

## Arguments

| | |
|---|---|
| protocol | a string from "http", "system", "R" respectively representing if the process is implemented remotety, locally or as R function. |
| site | the site where the process is implemented if remotely or the package where the process is implemented if as a R function. |
| service | a string corresponding to the name of the method implemented in the package "mtk" or the service that implements the process if remotely. |
| parameters | a vector of [mtkParameter] representing the parameters necessary to run the process. |
| parametersList | a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters. |
| ready | a logical to indicate if the process is ready to run. |
| state | a logical to indicate if the process finished running and the results are available. |
| result | an object of the class [mtkEvaluatorResult] to hold the results produced by the Evaluator. |

## Value

an object of the mtkEvaluator class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Create an evaluator with the model "Ishigami" implemented in the package "mtk".

  evaluator1 <- mtkEvaluator(service="Ishigami")

# Create an evaluator avec the model "WWDM"  implemented in the package "mtk"
  evaluator2 <- mtkEvaluator(service="WWDM",
  parametersList=list(year=3, tout=FALSE))
```

---

## mtkEvaluator-class *The* mtkEvaluator *class*

---

### Description

The mtkEvaluator class is a sub-class of the class mtkProcess used to manage the model simulation. It provides all the slots and methods defined in the class mtkProcess.

### Class Hierarchy

**Parent classes :** mtkProcess

**Direct Known Subclasses :** mtkNativeEvaluator,mtkWWDMEvaluator, etc.

### Constructor

**mtkEvaluator** signature(protocol = "R", site = "mtk", service = "", parameters = NULL, parametersList = NULL, ready = TRUE, state = FALSE, result = NULL)

### Slots

name**:** (character) always takes the string "evaluate".

protocol**:** (character) a string to name the protocol used to run the process: http, system, R, etc.

site**:** (character) a string to indicate where the service is located.

service**:** (character) a string to name the service to invoke.

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

### Methods

setName signature(this = "mtkEvaluator", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkEvaluator", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkEvaluator"): Returns the parameters as a named list.

`is.ready` signature( = "mtkEvaluator"): Tests if the process is ready to run.

`setReady` signature(this = "mtkEvaluator", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkEvaluator"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkEvaluator", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkEvaluator"): Returns the results produced by the process as a [`mtkEvaluatorResult`].

`getData` signature(this = "mtkEvaluator"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkEvaluator"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkEvaluator", context= "mtkExpWorkflow"): Runs the model with the experimental design defined in the context.

`summary` signature(object = "mtkEvaluator"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkEvaluator"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkEvaluator"): Plots the results produced by the process.

`report` signature(this = "mtkEvaluator"): Reports the results produced by the process.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create an evaluator with the model "Ishigami"
# implemented in the package "mtk".
   evaluator1 <- mtkEvaluator(service="Ishigami")

# Create an evaluator with the model "WWDM"
# implemented in the package "mtk"
 evaluator2 <- mtkEvaluator(service="WWDM",
  parametersList=list(year=3, tout=FALSE))
```

## mtkEvaluatorResult *The constructor of the class* mtkEvaluatorResult

### Description

The constructor

### Usage

```
mtkEvaluatorResult(main=data.frame(), information=list())
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the data produced by the model simulation.. |
| information | a named list containing the information about the managed data or process. |

### Value

an object of the mtkEvaluatorResult class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Construct an object of the \code{mtkEvaluatorResult}
# class from a data-frame.
data <- data.frame()
simulation <- mtkEvaluatorResult(main=data,
information = list(model="Ishigami"))
```

## mtkEvaluatorResult-class *The* mtkEvaluatorResult *class*

### Description

A class to collect the results of the simulation produced with a model.

### Class Hierarchy

**Parent classes :** mtkResult

**Direct Known Subclasses :** mtkWWDMEvaluatorResult, etc.

### Constructor

mtkEvaluatorResult signature(main=data.frame(),information=list())

### Slots

main: (data.frame) a data.frame holding the data produced by the model simulation.

information: (list) a named list containing information about the managed data and process.

### Methods

summary signature(object = "mtkEvaluatorResult"): Provides a summary of the data produced with the model simulation.

print signature(x = "mtkEvaluatorResult"): Prints a report of the data produced with the model simulation.

plot signature(x = "mtkEvaluatorResult"): Plots the data produced with the model simulation.

### See Also

help(morris, sensitivity) and help(Regression)

### Examples

```
## See examples from help(mtkAnalyserResult)
```

---

## mtkExperiment *The constructor of the class* mtkExperiment

---

### Description

A simple way to build a workflow for interactive use.

### Usage

```
mtkExperiment(expFactors,
design=NULL, designInfo=NULL,
model=NULL, modelInfo=NULL,
analyze=NULL, analyzeInfo=NULL,
XY=NULL)
```

### Arguments

| | |
|---|---|
| expFactors | (mtkExpFactors) an object of the mtkExpFactors class. |
| design | (NULL or character) the name of the method used to build the experiment design. NULL means that the experiment design is produced off-line and should be imported through the parameter XY$X. |

| | |
|---|---|
| designInfo | (list) a named list to specify the parameters used to generate the experiments design. |
| model | (NULL or character) the name of the model to simulate. NULL means that the simulation is produced off-line and should be imported through the parameter XY$Y. |
| modelInfo | (list) a named list to specify the parameters used to manage the model simulation. |
| analyze | (NULL or character) the name of the method used to compute the sensitivity index. |
| analyzeInfo | (list) a named list to specify the parameters used to carry out the analyses. |
| XY | (NULL or list) a named list with two elements X and Y: X allows importing the experiment design produced off-line and Y allows importing the model simulation produced off-line. |

## Value

an object of the mtkExperiment class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Compute the sensitivity index with the method "Regression"
# over the model "Ishigami" according to an experiment design
# generated with the method "BasicMonteCarlo"

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

ishiReg <- mtkExperiment(expFactors=ishi.factors,
design="BasicMonteCarlo", designInfo=list(size=20),
model="Ishigami",
analyze="Regression", analyzeInfo=list(nboot=20))

run(ishiReg)
summary(ishiReg)
```

mtkExperiment-class

*The* mtkExperiment *class*

## Description

The class mtkExperiment is a sub-class of the class mtkExpWorkflow. It provides more facilities and more flexible use for interactive manipulation of the workflow. Different behaviors may be expected by appropriately combining the parameters: design – the method used for the experiment design; model – the model used for the simulation; analyze – the method used for calculating the sensitivity index; XY – argument used to provide with data produced off-line;

For example, 1) if the experiment design is produced off-line, it will be imported with the help of the parameter "XY$X" ; 2) if the model simulation is produced off-line, it will be imported through the parameter "XY$Y";

## Class Hierarchy

**Parent classes :** mtkExpWorkflow

**Direct Known Subclasses :**

## Constructor

mtkExperiment signature(expFactors, design=NULL, designInfo=NULL, model=NULL, modelInfo=NULL, analyze=NULL, analyzeInfo=NULL, XY=NULL)

## Slots

expFactors**:** (mtkExpFactors) an object of the mtkExpFactors class.

processesVector**:** (vector) a vector of objects from the class mtkProcess or its subclasses.

## Methods

addProcess signature(this = "mtkExperiment", p = "mtkProcess", name = "character"): Adds a process to the workflow.

deleteProcess signature(this = "mtkExperiment", name = "character"): Deletes a process from the workflow.

setProcess signature(this = "mtkExperiment", p = "mtkProcess", name = "character"): Replaces a process into the workflow.

getProcess signature(this = "mtkExperiment", name = "character"): Gets a process from the workflow.

extractData signature(this = "mtkExperiment", name = "list"): Returns the results produced by the workflow as a data.frame. According to the processes specified with the argument "name", we can fetch the results produced by the process "design", "evaluate" or "analyze". i.e. name=c("design") gives the experimental design produced by the process "design" and name=c("design","evaluate") gives both the experimental design and the model simulation, etc.

`reevaluate` signature(this = "mtkExperiment", name = "character"): Re-evaluate the processes of the workflow to know if they should be re-run. This should be done after changing a process of the workflow. According to the order "design", evaluate", "analyze", only the processes after the one given by the argument "name" will be re-evaluated.

`run` signature(this = "mtkExperiment", context= "missing"): Runs the ExpWorkflow.

`serializeOn` signature(this = "mtkExperiment"): Returns all data managed by the workflow as a named list.

`summary` signature(object = "mtkExperiment"): Provides a summary of the results produced by the workflow.

`print` signature(x = "mtkExperiment"): Prints a report of the results produced by the workflow.

`plot` signature(x = "mtkExperiment"): Plots the results produced by the workflow.

`report` signature(this = "mtkExperiment"): Reports the results produced by the workflow.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Compute the sensitivity index with the method "Regression"
# over the model "Ishigami" according to an experiment design
# generated with the method "BasicMonteCarlo"

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

exp <- mtkExperiment(
factors,
design = 'BasicMonteCarlo',
designInfo=list(size=20),
model = 'Ishigami',
analyze = 'Regression',
analyzeInfo = list(ntboot=20)
)
run(exp)
summary(exp)
```

## mtkExpFactors *The constructor of the class* mtkExpFactors

### Description

This class is used to define the input factors for a simulation experiment.

### Usage

```
mtkExpFactors(expFactorsList=list())
```

### Arguments

expFactorsList
: a list of mtkFactor objects.

### Value

an object of the mtkExpFactors class

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create an object of the class mtkExpFactor
x1 <- make.mtkFactor(name="x1", distribName="unif",
             distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))
```

## mtkExpFactors-class *The* mtkExpFactors *class*

### Description

The mtkExpFactors class is a class used to manage the factors involved in a sensitivity analysis.

### Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :**

### Constructor

`mtkExpFactors` signature(expFactorsList=list())

### Slots

`expFactorsList:` (`list`) a list of `mtkFactor` objects.

### Methods

**initialize** signature(.Object="mtkExpFactors") : The initializer.

**setFactors** signature(this="mtkExpFactors",aFactList="list"): Assigns a new list of `mtkFactor` objects.

**getFactors** signature(this="mtkExpFactors"): Returns the factors as a list of `mtkFactor` objects.

**getNames** signature(this = "mtkExpFactors"): Returns the names of the managed factors.

**getFactorNames** signature(this = "mtkExpFactors"): Returns the names of the managed factors as the method `getNames`.

**getDistributionNames** signature(this="mtkExpFactors"): Gets a list of mtkExpFactors names.

**getDistributionParameters** signature(this="mtkExpFactors"): Gets the parameters.

**getFeatures** signature(this = "mtkExpFactors"): Returns the features associated with the managed factors.

**getDistributionNominalValues** signature(this = "mtkExpFactors"): Returns the nominal values associated with the distributions of the managed factors.

**getDistributionNominalValueTypes** signature(this = "mtkExpFactors"): Returns the data type of the nominal value associated with the managed factors.

**[[** signature( x = "mtkExpFactors", i="ANY" ): Extracts or replaces parts of an object of the class `mtkExpFactors`.

**[** signature( x = "mtkExpFactors", i="ANY" ): Extracts or replaces parts of an object of class `mtkExpFactors`.

**$** signature(x = "mtkExpFactors"): Extracts or replaces parts of an object of the class.

`print` signature(x = "mtkExpFactors"): Prints information about the managed factors.

`show` signature(object = "mtkExpFactors"): Displays the underlying object of the class `mtkExpFactors`.

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define the factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
       distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
       distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))
```

mtkExpWorkflow *The constructor of the class* mtkExpWorkflow

## Description

The class mtkExpWorkflow is used to manage the processes involved in a sensitivity analysis. We can construct a workflow in two ways: either from pre-defined factors and processes or from a XML file.

## Usage

```
mtkExpWorkflow(
expFactors = NULL,
processesVector = NULL,
xmlFilePath = NULL
)
```

## Arguments

| | |
|---|---|
| expFactors | (mtkExpFactors) an object of the mtkExpFactors class. |
| processesVector | (vector) a vector of objects from the class mtkProcess or its sub-classes. |
| xmlFilePath | (character) a string holding the name of the XML file and its path. |

## Value

an object of the mtkExpWorkflow class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
##############
# Example 1: Construct a workflow
# from the factors and the processes
##############

x1 <- make.mtkFactor(name="x1", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
```

```
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))


designer <- mtkNativeDesigner("BasicMonteCarlo",
            information=list(size=20))
model <- mtkNativeEvaluator("Ishigami" )
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

ishiReg <- mtkExpWorkflow(expFactors=ishi.factors,
   processesVector=c( design=designer,
      evaluate=model,
      analyze=analyser)
   )
run(ishiReg)
summary(ishiReg)

##############
######## Example 2: Construct a workflow from a XML file
##############
# Create a workflow from XML file
## Nota: If your XML file is a local file
## for example /var/tmp/X.xml", you should
## create the workflow as follows:
## workflow <- mtkExpWorkflow(
##   xmlFilePath="/var/tmp/X.xml"
## )


xmlFile <- "WWDM_morris.xml"

## If WWDM_morris.xml is a local file, the next line is not necessary.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",xmlFile,sep = "")

workflow <- mtkExpWorkflow(xmlFilePath=xmlFilePath)

# Run the workflow and report the results
run(workflow)
summary(workflow)
```

---

`mtkExpWorkflow-class`

*The* `mtkExpWorkflow` *class*

---

## Description

The `mtkExpWorkflow` class is used to coordinate the processes involved in a sensitivity analysis. It controls the state of the processes and coordinates their chaining.

### Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :**

### Constructor

`mtkExpWorkflow` signature(expFactors=NULL, processesVector=NULL, xmlFilePath=NULL)

### Slots

`expFactors`**:** (`mtkExpFactors`) an object of the `mtkExpFactors` class.

`processesVector`**:** (`vector`) a vector of objects from the class `mtkProcess` or its subclasses.

### Methods

`addProcess` signature(this = "mtkExpWorkflow", p = "mtkProcess", name = "character"): Adds a process to the workflow.

`deleteProcess` signature(this = "mtkExpWorkflow", name = "character"): Deletes a process from the workflow.

`setProcess` signature(this = "mtkExpWorkflow", p = "mtkProcess", name = "character"): Replaces a process into the workflow.

`getProcess` signature(this = "mtkExpWorkflow", name = "character"): Gets a process from the workflow.

`extractData` signature(this = "mtkExpWorkflow", name = "list"): Returns the results produced by the workflow as a data.frame. According to the processes specified with the argument "name", we can fetch the results produced by the process "design", "evaluate" or "analyze". i.e. name=c("design") gives the experimental design produced by the process "design" and name=c("design","evaluate") gives both the experimental design and the model simulation, etc.

`reevaluate` signature(this = "mtkExpWorkflow", name = "character"): Re-evaluate the processes of the workflow to know if they should be re-run. This should be done after changing a process of the workflow. According to the order "design", evaluate", "analyze", only the processes after the one given by the argument "name" will be re-evaluated.

`run` signature(this = "mtkExpWorkflow", context= "missing"): Runs the workflow.

`serializeOn` signature(this = "mtkExpWorkflow"): Returns all data managed by the workflow as a named list.

`summary` signature(object = "mtkExpWorkflow"): Provides a summary of the results produced by the workflow.

`print` signature(x = "mtkExpWorkflow"): Prints a report of the results produced by the workflow.

`plot` signature(x = "mtkExpWorkflow"): Plots the results produced by the workflow.

`report` signature(this = "mtkExpWorkflow"): Reports the results produced by the workflow.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
##############
# Example 1: Construct a workflow
# from the factors and the processes
##############

# Specify the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Define the processes
designer <- mtkNativeDesigner("BasicMonteCarlo",
            information=list(size=20))
model <- mtkNativeEvaluator("Ishigami" )
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

# Build the workflow
ishiReg <- mtkExpWorkflow( expFactors=ishi.factors,
   processesVector=c( design=designer,
      evaluate=model,
      analyze=analyser)
   )

# Run the workflow and report the results
run(ishiReg)
summary(ishiReg)

##############
######## Example 2: Construct a workflow from a XML file
##############
# # XML file is held in the directory of the library: "inst/extdata/"

# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"
## find where the examples are held.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",xmlFile,sep = "")

# Create the workflow from the XML
## Nota: If your XML file is local
## file for example /var/tmp/X.xml", you should
## create the workflow as follows:
## workflow <- mtkExpWorkflow(
```

```
## xmlFilePath = "/var/tmp/X.xml"
## )

workflow <- mtkExpWorkflow(xmlFilePath=xmlFilePath)

# Run the workflow and report the results
run(workflow)
summary(workflow)
```

---

`mtkFactor` *The constructor of the class* `mtkFactor`

---

## Description

The constructor of the class `mtkFactor`. See also the function `make.mtkFactor`

## Usage

```
mtkFactor(name="unkown", id="unkown", unit="", type="numeric",
 domain=mtkDomain(), featureList=list())
```

## Arguments

| | |
|---|---|
| `name` | a string to name the factor. |
| `id` | a string giving the id of the factor in the code. |
| `unit` | a string giving the measurement unit of the factor levels. |
| `type` | a string giving the data type of the factor levels. |
| `domain` | an object of the class `mtkDomain` giving the uncertainty domain associated with the factor. |
| `featureList` | a list giving the uncertainty domain associated with the factor. It may be a list of objects from the class `mtkDomain` or a named list defining the features. |

## Value

an object of the `mtkFactor` class

## Author(s)

Juhui WANG, MIA-jouy, INRA

## Examples

```
# Create an object of the class mtkExpFactor
x1 <- make.mtkFactor(name="x1", distribName="unif",
             distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
```

mtkFactor-class *The* mtkFactor *class*

## Description

The class used to manage an input factor and its uncertainty distribution.

## Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :**

## Constructor

mtkFactor signature(name="unkown", id="unkown", unit="", type="numeric", domain=mtkDomain(), featureList=list())

## Slots

name: the name of the input factor.

id: the name of the factor in the simulation code, if different from name.

unit: the measurement units of the factor values. This can be used in graphics or reports, for example.

type: the data type of the factor's values.

domain: the mtkDomain object that describes the factor's uncertainty.

featureList: the list of features that may be associated with the factor.

## Methods

initialize signature(.Object = "mtkFactor"): The initializer of the class mtkFactor.

getName signature(this="mtkFactor"): Fetches the name of the factor.

getType signature(this = "mtkFactor"): Returns the data type of the factor's levels.

getDomain signature(this="mtkFactor"): Fetches the domain associated with the factor. It returns an object of the class mtkDomain.

getDistributionName signature(this="mtkFactor"): Fetches the name of the distribution associated with the uncertainty domain.

getDistributionNominalValue signature(this="mtkFactor"): Fetches the nominal value of the distribution associated with the uncertainty domain.

getDistributionNominalValueType signature(this="mtkFactor"): Fetches the data type associated with the uncertainty domain.

getDiscreteDistributionType signature(this="mtkFactor"): Returns the discrete distribution type.

getDiscreteDistributionLevels signature(this="mtkFactor"): Returns the levels managed by a discrete distribution.

getDiscreteDistributionWeights signature(this="mtkFactor"): Returns the weights managed by a discrete distribution.

`getDistributionParameters` signature(this="mtkFactor"): The getDistributionParameters method.

`getFeatures` signature(this="mtkFactor"): Returns the features as a named list.

`getMTKFeatures` signature(this="mtkFactor"): Returns the features as a vector of objects from the class `mtkFeature`.

`setName` signature(this = "mtkFactor", name = "character"): Gives a new name to the factor.

`setDomain` signature(this = "mtkFactor", domain = "mtkDomain"): Associates a new domain with the factor.

`setType` signature(this = "mtkFactor", type = "character"): Names explicitly the data type managed by the factor.

`setFeatures` signature(this="mtkFactor",aFList="list): Gives new features to the factor. `aFList` may be a vector of objects from the class `mtkFeature` or a named list from which we can build a list of features.

`print` signature(x = "mtkFactor"): Prints the data managed by the factor.

`show` signature(object = "mtkFactor"): Displays the underlying object of the class `mtkFactor`.

## Author(s)

Juhui WANG and Hervé Monod, MIA-jouy, INRA, Hervé Richard, BioSP, INRA

## Examples

```
# Manage a factor x1 with  a mtkFactor object.

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
getName(x1)
getDomain(x1)
getDistributionName(x1)
getType(x1)
setType(x1, "double")
getType(x1); # 'double'
```

---

# mtkFastAnalyser *The constructor of the class* `mtkFastAnalyser`

---

## Description

The constructor

## Usage

```
mtkFastAnalyser(mtkParameters = NULL, listParameters = NULL)
```

## Arguments

`mtkParameters`
: a vector of [`mtkParameter`] representing the parameters necessary to run the process.

`listParameters`
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the mtkFastAnalyser class

### References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### See Also

```
help(fast, sensitivity)
```

### Examples

```
##  Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkFastDesigner(listParameters
     = list(n=1000))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkFastAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
```

---

## mtkFastAnalyser-class

*The* mtkFastAnalyser *class*

---

### Description

The mtkFastAnalyser class is a sub-class of the class mtkAnalyser. It implements the sensitivity analysis method 'Fast' and provides all the slots and methods defined in the class mtkAnalyser.

### Class Hierarchy

**Parent classes :** mtkAnalyser

**Direct Known Subclasses :**

### Constructor

mtkFastAnalyser signature(mtkParameters = NULL, listParameters = NULL)

### Slots

name: (character): always takes the string "analyze".

protocol: (character): always takes the string "R".

site: (character): always takes the string "mtk".

service: (character): always takes the string "Fast".

parameters: (vector): a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready: (logical): a logical to tell if the process is ready to run.

state: (logical): a logical to tell if the results produced by the process are available and ready to be consumed.

result: (ANY): a data holder to hold the results produced by the process

### Methods

setName signature(this = "mtkFastAnalyser", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkFastAnalyser", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkFastAnalyser"): Returns the parameters as a named list.

is.ready signature( = "mtkFastAnalyser"): Tests if the process is ready to run.

setReady signature(this = "mtkFastAnalyser", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkFastAnalyser"): Tests if the results produced by the process are available.

setReady signature(this = "mtkFastAnalyser", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkFastAnalyser"): Returns the results produced by the process as a [mtkAnalyserResult].

getData signature(this = "mtkFastAnalyser"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkFastAnalyser"): Returns all data managed by the process as a named list.

run signature(this = "mtkFastAnalyser", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

summary signature(object = "mtkFastAnalyser"): Provides a summary of the results produced by the process.

print signature(x = "mtkFastAnalyser"): Prints a report of the results produced by the process.

plot signature(x = "mtkFastAnalyser"): Plots the results produced by the process.

report signature(this = "mtkFastAnalyser"): Reports the results produced by the process.

### References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### See Also

```
help(fast, sensitivity)
```

### Examples

```
##  Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkFastDesigner(listParameters
     = list(n=1000))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkFastAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
```

---

## mtkFastAnalyserResult

*The constructor of the class* `mtkFastAnalyserResult`

---

### Description

The constructor

### Usage

```
mtkFastAnalyserResult(main,information=NULL)
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the results of the sensitivity analysis produced by the analyser. |
| information | a named list containing the information about the managed data. |

### Value

an object of the mtkFastAnalyserResult class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# See examples from the help of the method: help(Fast)
```

---

## mtkFastAnalyserResult-class
*The* mtkFastAnalyserResult *class*

---

### Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Fast.

### Class Hierarchy

**Parent classes :** mtkAnalyserResult

**Direct Known Subclasses :**

### Constructor

mtkFastAnalyserResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkFastAnalyserResult"): Provides a summary of the results produced by the analyser.

print signature(x = "mtkFastAnalyserResult"): Prints a report of the results produced by the analyser.

plot signature(x = "mtkFastAnalyserResult"): Plots the results produced by the analyser.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# See examples from the help of the method: help(Fast)
```

---

## mtkFastDesigner *The constructor of the class* mtkFastDesigner

---

### Description

The constructor

### Usage

```
mtkFastDesigner(mtkParameters = NULL, listParameters = NULL)
```

### Arguments

mtkParameters
: a vector of [mtkParameter] representing the parameters necessary to run the process.

listParameters
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the mtkFastDesigner class

### See Also

```
help(fast, sensitivity)
```

### Examples

```
##  Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkFastDesigner(listParameters
     = list(n=1000))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkFastAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
```

---

## mtkFastDesigner-class

*The* `mtkFastDesigner` *class*

---

### Description

The `mtkFastDesigner` class is a sub-class of the class `mtkDesigner`. It implements the sampling method `Fast` and provides all the slots and methods defined in the class `mtkDesigner`.

### Class Hierarchy

**Parent classes :** `mtkDesigner`

**Direct Known Subclasses :**

### Constructor

`mtkFastDesigner` signature(mtkParameters = NULL, listParameters = NULL)

## Slots

name: (character) always takes the string "design".

protocol: (character) always takes the string "R".

site: (character) always takes the string "mtk".

service: (character) always takes the string "Fast".

parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready: (logical) a logical to tell if the process is ready to run.

state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result: (ANY) a data holder to hold the results produced by the process

## Methods

setName signature(this = "mtkFastDesigner", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkFastDesigner", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkFastDesigner"): Returns the parameters as a named list.

is.ready signature( = "mtkFastDesigner"): Tests if the process is ready to run.

setReady signature(this = "mtkFastDesigner", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkFastDesigner"): Tests if the results produced by the process are available.

setReady signature(this = "mtkFastDesigner", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkFastDesigner"): Returns the results produced by the process as a [mtkDesignerResult].

getData signature(this = "mtkFastDesigner"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkFastDesigner"): Returns all data managed by the process as a named list.

run signature(this = "mtkFastDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

summary signature(object = "mtkFastDesigner"): Provides a summary of the results produced by the process.

print signature(x = "mtkFastDesigner"): Prints a report of the results produced by the process.

plot signature(x = "mtkFastDesigner"): Plots the results produced by the process.

report signature(this = "mtkFastDesigner"): Reports the results produced by the process.

## References

1. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York.
2. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## See Also

```
help(fast, sensitivity)
```

## Examples

```
##  Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkFastDesigner(listParameters
     = list(n=1000))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkFastAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
```

---

# mtkFastDesignerResult

*The constructor of the class* `mtkFastDesignerResult`

---

## Description

The constructor

## Usage

```
mtkFastDesignerResult(main,information=NULL)
```

## Arguments

| | |
|---|---|
| `main` | a data.frame holding the experimental design produced by the designer. |
| `information` | a named list containing the information about the managed data. |

### Value

an object of the mtkFastDesignerResult class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# See examples from the help of the method: help(Fast)
```

---

## mtkFastDesignerResult-class

*The* mtkFastDesignerResult *class*

---

### Description

A class to collect the experimental design produced by the designer implementing the method Fast.

### Class Hierarchy

**Parent classes :** mtkDesignerResult

**Direct Known Subclasses :**

### Constructor

mtkFastDesignerResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkFastDesignerResult"): Provides a summary of the experimental design produced by the designer.

print signature(x = "mtkFastDesignerResult"): Prints a report of the experimental design produced by the designer.

plot signature(x = "mtkFastDesignerResult"): Plots the experimental design produced by the designer.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# See examples from the help of the method: help(Fast)
```

---

## mtkFeature *The constructor of the class* mtkFeature

---

### Description

The constructor of the class mtkFeature. See also make.mtkFeatureList.

### Usage

```
mtkFeature(name='unknown', type='logical', val=NULL)
```

### Arguments

| | |
|---|---|
| name | (character) the name of the feature. |
| type | (character) the data type managed by the feature such as 'numeric', 'double', 'logical', etc.. |
| val | (ANY) the value of the feature. |

### Value

an object of the mtkFeature class

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# creates a feature "he"
f <- mtkFeature(name='he', type ='character', val = 'pekin')

# We usually use the 'make.mtkFeatureList()' function to define
# a list of 'mtkFeature' instead of the constructor
# of the 'mtkFeature' class

flist <- make.mtkFeatureList(list(min=-1,max=+1,shape="hello"))
```

mtkFeature-class *The* mtkFeature *class*

## Description

The mtkFeature class is a class used to manage the features associated with a factor.

## Class Hierarchy

**Parent classes :** mtkValue

**Direct Known Subclasses :**

## Constructor

mtkFeature signature(name='unknown', type='logical', val=NULL)

make.mtkFeatureList signature(x=list())

## Slots

name**:** (character) the name of the feature.

type**:** (character) the type of value managed by the feature.

val**:** (ANY) the value of the feature in the right type.

## Methods

getName signature( this = "mtkFeature"): Returns the value of the slot "name".

getValue signature( this = "mtkFeature"): Returns the value of the slot "val".

getType signature(this = "mtkFeature"): Returns the value of the slot "type".

setName signature( this = "mtkFeature", name = "character"): Gives a new value to the slot "name".

setType signature( this = "mtkFeature", type = "character"): Gives a new value to the slot "type".

setValue signature(this = "mtkFeature", val = "ANY"): Gives a new value to the slot "val".

show signature( object = "mtkFeature"): Prints a report of the data managed by the underlying object.

print signature(x = "mtkFeature"): Prints the information managed by the underlying object.

## Author(s)

Juhui WANG, MIA-jouy, INRA

## Examples

```
# Create an object of the 'mtkFeature' class.

f <- mtkFeature(name="x", type="double", val=0.0)

# We usually use the make.mtkFeatureList function to define a list of mtkFeature
# instead of the constructor of the mtkFeature class

flist <- make.mtkFeatureList(list(min=-1,max=+1,shape="hello"))
```

mtkIshigamiEvaluator *The constructor of the class* mtkIshigamiEvaluator

## Description

The constructor

## Usage

```
mtkIshigamiEvaluator()
```

## Value

an object of the mtkIshigamiEvaluator class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Carry out a sensitivity analysis with the Ishigami model

## Input the factors
data(Ishigami.factors)

## Specify the experiments designer
designer <- mtkNativeDesigner ("BasicMonteCarlo",
information=list(size=20))

## Specify the model simulator
model <- mtkIshigamiEvaluator()

## Specify the sensiticity analyser
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

## Specify the workflow
ishiReg <- new("mtkExpWorkflow", expFactors=Ishigami.factors,
   processesVector=c(
              design=designer,
              evaluate=model,
              analyze=analyser)
   )
## Run and report the results
run(ishiReg)
```

```
summary(ishiReg)
```

---

mtkIshigamiEvaluator-class
*The* mtkIshigamiEvaluator *class*

---

## Description

The mtkIshigamiEvaluator class is a sub-class of the class mtkEvaluator used to manage the simulation of the model Ishigami.

## Class Hierarchy

**Parent classes :** mtkEvaluator

**Direct Known Subclasses :**

## Constructor

mtkIshigamiEvaluator signature()

## Slots

name**:** (character) always takes the string "evaluate".

protocol**:** (character) a string to name the protocol used to run the process: http, system, R, etc. Here, it takes the character "R".

site**:** (character) a string to indicate where the service is located. Here, it always takes the string "mtk".

service**:** (character) a string to name the service to invoke. Here, it always takes the string "Ishigami".

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service. The "Ishigami" model does not need parameters.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

## Methods

setName signature(this = "mtkIshigamiEvaluator", name = "character"): non useful, method inherited from the parent class.

setParameters signature(this = "mtkIshigamiEvaluator", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkIshigamiEvaluator"): Returns the parameters as a named list.

is.ready signature( = "mtkIshigamiEvaluator"): Tests if the process is ready to run.

setReady signature(this = "mtkIshigamiEvaluator", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkIshigamiEvaluator"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkIshigamiEvaluator", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkIshigamiEvaluator"): Returns the results produced by the process as a [`mtkEvaluatorResult`].

`getData` signature(this = "mtkIshigamiEvaluator"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkIshigamiEvaluator"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkIshigamiEvaluator", context= "mtkExpWorkflow"): runs the simulation.

`summary` signature(object = "mtkIshigamiEvaluator"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkIshigamiEvaluator"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkIshigamiEvaluator"): Plots the results produced by the process.

`report` signature(this = "mtkIshigamiEvaluator"): Reports the results produced by the process.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Carry out a sensitivity analysis with the Ishigami model

## Input the factors
data(Ishigami.factors)

## Specify the experiments designer
designer <- mtkNativeDesigner ("BasicMonteCarlo",
information=list(size=20))

## Specify the model simulator
model <- mtkIshigamiEvaluator()

## Specify the sensiticity analyser
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

## Specify the workflow
ishiReg <- new("mtkExpWorkflow", expFactors=Ishigami.factors,
   processesVector=c(
              design=designer,
              evaluate=model,
```

```
                analyze=analyser)
   )
## Run and report the results
run(ishiReg)
summary(ishiReg)
```

---

`mtkLevels` *The constructor of the class* `mtkLevels`

---

## Description

The constructor of the class `mtkLevels`.

## Usage

```
mtkLevels(type = "categorical", levels=vector(), weights=numeric(0))
```

## Arguments

| | |
|---|---|
| `type` | a string to specify the type of the discrete distribution: categorical, qualitative, etc. |
| `levels` | a vector of levels for a discrete domain. |
| `weights` | a vector of numeric values used to weight the levels. |

## Value

an object of the `mtkLevels` class

## Author(s)

Juhui WANG, MIA-jouy, INRA

## Examples

```
# creates an object of the class mtkLevel
l1 <- mtkLevels(type="qualitative",levels = c("x", "y"), weights=c(0.5, 0.5))
l2 <- mtkLevels(levels = c("a", "b", "c"))
l3 <- mtkLevels(levels = c("a", "b", "c"), weights=c(3, 5, 3))
```

mtkLevels-class *The* mtkLevels *class*

## Description

The mtkLevels class is a class used to manage the weighting levels associated with a factor's domain.

## Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :**

## Constructor

mtkLevesl signature(type = "categorical", levels=vector(), weights=numeric(0))

## Slots

type**:** (character) a string to give the type of the discrete distribution such as 'categorical', 'qualitative', etc.

levels**:** (vector) a vector to specify the levels.

weights**:** (numeric) a numeric vector used to weight the levels.

## Methods

getType signature(this = "mtkLevels"): Returns the type of the discrete distribution such as 'categorical', 'qualitative', etc .

setType signature(this = "mtkLevels", type="character"): Assigns a new type to the underlying object.

getLevels signature(this = "mtkLevels"): Returns the vector of the levels.

setLevels signature(this = "mtkLevels", levels = "vector"): Assigns a new vector to the levels.

getWeights signature(this = "mtkLevels"): Returns the vector of the weights.

setWeights signature(this = "mtkLevels", weights = "numeric"): Assigns new vector to the weight.

print signature(x = "mtkLevel"): Prints a summarized report about the underlying object of the class mtkLevels.

summary signature(object = "mtkLevel"): Gives a summary about the underlying object.

show signature(object = "mtkLevel"): Displays informations about the underlying object.

## Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create an object of the class 'mtkLevels'

l <- mtkLevels(type='categorical', levels=seq(1:3), weight=rep(0.33, 3))

# Set the levels'name to ('a', 'b', 'c')

setLevels(l, levels=c('a', 'b', 'c'))
```

---

## mtkMorrisAnalyser *The constructor of the class* mtkMorrisAnalyser

---

### Description

The constructor

### Usage

```
mtkMorrisAnalyser(mtkParameters = NULL, listParameters = NULL)
```

### Arguments

| | |
|---|---|
| mtkParameters | a vector of [mtkParameter] holding the parameters necessary to run the process. |
| listParameters | a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters. |

### Value

an object of the mtkMorrisAnalyser class

### References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

### See Also

help(morris, sensitivity) and help(Morris)

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Morris" method


# Generate the factors
data(Ishigami.factors)
```

```
#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkMorrisDesigner( listParameters
     = list(r=20, type="oat", levels=4, grid.jump=2))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

`mtkMorrisAnalyser-class`

*The* `mtkMorrisAnalyser` *class*

---

## Description

The `mtkMorrisAnalyser` class is a sub-class of the class `mtkAnalyser`. It implements the sensitivity analysis method `Morris` and provides all the slots and methods defined in the class `mtkAnalyser`.

## Class Hierarchy

**Parent classes :** `mtkAnalyser`

**Direct Known Subclasses :**

## Constructor

`mtkMorrisAnalyser` signature(mtkParameters = NULL, listParameters = NULL)

## Slots

`name`**:** (`character`) always takes the string "analyze".

`protocol`**:** (`character`) always takes the string "R".

`site`**:** (`character`) always takes the string "mtk".

`service`**:** (`character`) always takes the string "Morris".

`parameters`**:** (`vector`) a vector of [`mtkParameter`] containing the parameters to pass while calling the service.

`ready`**:** (`logical`) a logical to tell if the process is ready to run.

`state`: (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`: (`ANY`) a data holder to hold the results produced by the process

## Methods

`setName` signature(this = "mtkMorrisAnalyser", name = "character"): Not used, method inherited from the parent class.

`setParameters` signature(this = "mtkMorrisAnalyser", f = "vector"): Assigns new parameters to the process.

`getParameters` signature(this = "mtkMorrisAnalyser"): Returns the parameters as a named list.

`is.ready` signature( = "mtkMorrisAnalyser"): Tests if the process is ready to run.

`setReady` signature(this = "mtkMorrisAnalyser", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkMorrisAnalyser"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkMorrisAnalyser", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkMorrisAnalyser"): Returns the results produced by the process as a [`mtkMorrisAnalyserResult`].

`getData` signature(this = "mtkMorrisAnalyser"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkMorrisAnalyser"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkMorrisAnalyser", context= "mtkExpWorkflow"): Runs the process to generate the results.

`summary` signature(object = "mtkMorrisAnalyser"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkMorrisAnalyser"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkMorrisAnalyser"): Plots the results produced by the process.

`report` signature(this = "mtkMorrisAnalyser"): Reports the results produced by the process.

## References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

## See Also

`help(morris, sensitivity)` and `help(Morris)`

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Morris" method


# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkMorrisDesigner( listParameters
     = list(r=20, type="oat", levels=4, grid.jump=2))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

## mtkMorrisAnalyserResult

*The constructor of the class* `mtkMorrisAnalyserResult`

---

### Description

The constructor

### Usage

```
mtkMorrisAnalyserResult(main,information=NULL)
```

### Arguments

| | |
|---|---|
| `main` | a data.frame holding the results of the sensitivity analysis produced by the analyser. |
| `information` | a named list containing the information about the managed data. |

### Value

an object of the `mtkMorrisAnalyserResult` class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# See examples from the help of the method: help(Morris)
```

---

## mtkMorrisAnalyserResult-class

*The* mtkMorrisAnalyserResult *class*

---

### Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Morris.

### Class Hierarchy

**Parent classes :** mtkAnalyserResult

**Direct Known Subclasses :**

### Constructor

mtkMorrisAnalyserResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the results produced by the "Morris" analyser.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkMorrisAnalyserResult"): Provides a summary of the results produced by the analyser.

print signature(x = "mtkMorrisAnalyserResult"): Prints a report of the results produced by the analyser.

plot signature(x = "mtkMorrisAnalyserResult"): Plots the results produced by the analyser.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# See examples from the help of the method: help(Morris)
```

---

# mtkMorrisDesigner *The constructor of the class* mtkMorrisDesigner

---

## Description

The constructor

## Usage

```
mtkMorrisDesigner(mtkParameters = NULL, listParameters = NULL)
```

## Arguments

| | |
|---|---|
| mtkParameters | a vector of [mtkParameter] representing the parameters necessary to run the process. |
| listParameters | a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters. |

## Value

an object of the mtkMorrisDesigner class

## References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

## See Also

help(morris, sensitivity) and help(Morris)

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Morris" method


# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkMorrisDesigner( listParameters
     = list(r=20, type="oat", levels=4, grid.jump=2))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

## mtkMorrisDesigner-class

*The* `mtkMorrisDesigner` *class*

---

### Description

The `mtkMorrisDesigner` class is a sub-class of the class `mtkDesigner`. It implements the method `Morris` and provides all the slots and methods defined in the class `mtkDesigner`.

### Class Hierarchy

**Parent classes :** `mtkDesigner`

**Direct Known Subclasses :**

### Constructor

`mtkMorrisDesigner` signature(mtkParameters = NULL, listParameters = NULL)

## Slots

name: (character) always takes the string "design".

protocol: (character) always takes the string "R".

site: (character) always takes the string "mtk".

service: (character) always takes the string "Morris".

parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready: (logical) a logical to tell if the process is ready to run.

state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result: (ANY) a data holder to hold the results produced by the process

## Methods

setName signature(this = "mtkMorrisDesigner", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkMorrisDesigner", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkMorrisDesigner"): Returns the parameters as a named list.

is.ready signature( = "mtkMorrisDesigner"): Tests if the process is ready to run.

setReady signature(this = "mtkMorrisDesigner", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkMorrisDesigner"): Tests if the results produced by the process are available.

setReady signature(this = "mtkMorrisDesigner", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkMorrisDesigner"): Returns the results produced by the process as a [mtkMorrisDesignerResult].

getData signature(this = "mtkMorrisDesigner"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkMorrisDesigner"): Returns all data managed by the process as a named list.

run signature(this = "mtkMorrisDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

summary signature(object = "mtkMorrisDesigner"): Provides a summary of the results produced by the process.

print signature(x = "mtkMorrisDesigner"): Prints a report of the results produced by the process.

plot signature(x = "mtkMorrisDesigner"): Plots the results produced by the process.

report signature(this = "mtkMorrisDesigner"): Reports the results produced by the process.

## References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

### See Also

help(morris, sensitivity) and help(Morris)

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Morris" method


# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkMorrisDesigner( listParameters
     = list(r=20, type="oat", levels=4, grid.jump=2))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))

#  Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

## mtkMorrisDesignerResult

*The constructor of the class* mtkMorrisDesignerResult

---

### Description

The constructor

### Usage

```
mtkMorrisDesignerResult(main,information=NULL)
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the experimental design produced by the designer. |
| information | a named list containing the information about the managed data. |

### Value

an object of the mtkMorrisDesignerResult class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# See examples from the help of the method: help(Morris)
```

---

## mtkMorrisDesignerResult-class

*The* mtkMorrisDesignerResult *class*

---

### Description

A class to collect the experimental design produced by the designer implementing the method Morris.

### Class Hierarchy

**Parent classes :** mtkDesignerResult

**Direct Known Subclasses :**

### Constructor

mtkMorrisDesignerResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design produced by the designer.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkMorrisDesignerResult"): Provides a summary of the experimental design produced by the designer.

print signature(x = "mtkMorrisDesignerResult"): Prints a report of the experimental design produced by the designer.

plot signature(x = "mtkMorrisDesignerResult"): Plots the experimental design produced by the designer.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# See examples from the help of the method: help(Morris)
```

---

## mtkNativeAnalyser *The constructor of the class* mtkNativeAnalyser

---

### Description

The constructor.

### Usage

```
mtkNativeAnalyser(analyze=NULL, X=NULL, information=NULL)
```

### Arguments

| | |
|---|---|
| analyze | NULL, an R function or a string to specify the analyser to use. |
| X | NULL or a data.frame to load the results produced off-line. |
| information | a named list to provide with supplementary information about the analysis produced off-line or the parameters used by the analyser. |

### Value

an object of the mtkNativeAnalyser class

### Details

We can construct an object of the mtkNativeAnalyser class in three manners:

- the analyser is provided within the package "mtk"The argument "analyze" takes a string giving the name of the method used to carry out the sensitivity analysis, the argument "information" gives the list of parameters used by the analyser.
- the analyser is available as an R function implemented outside the package "mtk"The argument "analyze" takes an R function implementing the analyser, the argument "information" may be used to give supplementary information about the R function.
- the results of the sensitivity analysis are already available as a data.frame. We use "mtk" only for reporting.The argument "X" takes the data.frame holding the available results, and the argument "information" may be omitted or simply used to give supplementary information about the analysis.

More examples for using this class, see ?class(mtkNativeEvaluator).

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### See Also

`?class(mtkNativeEvaluator)`

### Examples

```
# Create a native analyser with the method "Morris" implemented in the package "mtk"

analyser <- mtkNativeAnalyser(
analyze="Morris",
information=list(nboot=20))
```

---

## mtkNativeAnalyser-class

*The* `mtkNativeAnalyser` *class*

---

### Description

The `mtkNativeAnalyser` class is a sub-class of the class `mtkAnalyser` used to manage the sensitivity analysis task implemented locally (i.e. tasks don't need to call services from the Web). It provides all the slots and methods defined in the class `mtkAnalyser`.

### Class Hierarchy

**Parent classes :** `mtkAnalyser`

**Direct Known Subclasses :**

### Constructor

`mtkNativeAnalyser` signature(analyze=NULL, X=NULL, information=NULL)

### Slots

`analyze`**:** (`ANY`) a string, an R function, or NULL to inform the method to use for the sensitivity analysis.

`name`**:** (`character`) always takes the string "analyze".

`protocol`**:** (`character`) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes "R".

`site`**:** (`character`) a string to indicate where the service is located.

`service`**:** (`character`) a string to name the service to invoke. Here, it may be a R function or a method implemented in the package "mtk".

`parameters`: (`vector`) a vector of [`mtkParameter`] containing the parameters to pass while calling the service.

`ready`: (`logical`) a logical to tell if the process is ready to run.

`state`: (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`: (`ANY`) a data holder to hold the results produced by the process

## Methods

`setName` signature(this = "mtkNativeAnalyser", name = "character"): Not used here, method inherited from the parent class.

`setParameters` signature(this = "mtkNativeAnalyser", f = "vector"): Assigns new parameters to the process.

`getParameters` signature(this = "mtkNativeAnalyser"): Returns the parameters as a named list.

`is.ready` signature( = "mtkNativeAnalyser"): Tests if the process is ready to run.

`setReady` signature(this = "mtkNativeAnalyser", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkNativeAnalyser"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkNativeAnalyser", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkNativeAnalyser"): Returns the results produced by the process as a [`mtkAnalyserResult`].

`getData` signature(this = "mtkNativeAnalyser"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkNativeAnalyser"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkNativeAnalyser", context= "mtkExpWorkflow"): Runs the Analyser.

`summary` signature(object = "mtkNativeAnalyser"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkNativeAnalyser"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkNativeAnalyser"): Plots the results produced by the process.

`report` signature(this = "mtkNativeAnalyser"): Reports the results produced by the process.

## Details

We can construct an object of the `mtkNativeAnalyser` class from the following situations:

1. The analyser is provided within the package "mtk";
2. The analyser is provided as an R function implemented outside the package "mtk"; If so, the R function must produce a result as a named list with two elements: X and information, where `X` is a date.frame containing the analysis result and `information` is a named list containing supplementary information about the analysis process.
3. The results of the model exploration are produced off-line and available as a data.frame. We just want to use the "mtk" package for reporting.

For detail uses, see examples from `help(mtkNativeEvaluator)`.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a native analyser with the method "Morris" implemented in the package "mtk"

analyser <- mtkNativeAnalyser(
analyze="Morris",
information=list(nboot=20))
```

---

# mtkNativeDesigner *The constructor of the class* mtkNativeDesigner

---

## Description

The constructor.

## Usage

```
mtkNativeDesigner(design=NULL, X=NULL, information=NULL)
```

## Arguments

| | |
|---|---|
| design | NULL, an R function or a string to specify the method used to generate the experiments design. |
| X | NULL or a data.frame to load the experimental design produced off-line. |
| information | a named list to provide with supplementary information about the experimental design produced off-line or the parameters used by the designer. |

## Value

an object of the mtkNativeDesigner class

## Details

We can construct an object of the mtkNativeDesigner class from the following situations:

- the designer is provided within the package "mtk"The argument "design" takes a string giving the method used to generate the experimental design, and the argument "information" gives the list of parameters used by the designer. e.g. designer <- mtkNativeDesigner( design="Morris", information = list(nboot=20)).
- the designer is provided with an R function implemented outside the package "mtk"The argument "design" takes the R function, the argument "information" may be used to give supplementary information about the R function.

- the experimental design is produced off-line and available as a data.frameThe argument "design" is not used, the argument "X" takes the data.frame holding the available experimental design, and the argument "information" may be omitted or simply used to give supplementary information about the method used to generate the experimental design. e.g. Designer <- mtkNativeDesigner( X = mcDesign, information = list(sampling = "Monte-Carlo")).

For details uses, see examples from `help(mtkNativeEvaluator)`.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## See Also

`help(mtkNativeEvaluator)`

## Examples

```
# Create a native designer with the method "Morris"
# implemented in the package "mtk"

designer <- mtkNativeDesigner(design="Morris", information=list(size=20))
```

---

# mtkNativeDesigner-class

*The* `mtkNativeDesigner` *class*

---

## Description

The `mtkNativeDesigner` class is a sub-class of the class `mtkDesigner` used to manage the sampling task implemented locally (i.e. tasks don't need to call services from the Web). By object inheriting, it provides all the slots and methods defined in the class `mtkDesigner`.

## Class Hierarchy

**Parent classes :** `mtkDesigner`

**Direct Known Subclasses :**

## Constructor

`mtkNativeDesigner` signature(design=NULL, X=NULL, information=NULL)

## Slots

`design`: (`ANY`) a string, an R function, or NULL to inform the designer to use.

`name`: (`character`) always takes the string "design".

`protocol`: (`character`) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes "R".

`site`: (`character`) a string to indicate where the service is located. Here, it gives no sense.

`service`: (`character`) a string to name the service to invoke.

`parameters`: (`vector`) a vector of [`mtkParameter`] containing the parameters to pass while calling the service.

`ready`: (`logical`) a logical to tell if the process is ready to run.

`state`: (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`: (`ANY`) a data holder to hold the results produced by the process

## Methods

`setName` signature(this = "mtkNativeDesigner", name = "character"): Method inherited from the parent class. It gives no sense here.

`setParameters` signature(this = "mtkNativeDesigner", f = "vector"): Assigns new parameters vector to the process.

`getParameters` signature(this = "mtkNativeDesigner"): Returns the parameters vector as a named list.

`is.ready` signature( = "mtkNativeDesigner"): Tests if the process is ready to run.

`setReady` signature(this = "mtkNativeDesigner", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkNativeDesigner"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkNativeDesigner", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkNativeDesigner"): Returns the results produced by the process as a [`mtkDesignerResult`].

`getData` signature(this = "mtkNativeDesigner"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkNativeDesigner"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkNativeDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

`summary` signature(object = "mtkNativeDesigner"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkNativeDesigner"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkNativeDesigner"): Produces a graphical report of the results produced by the process.

`report` signature(this = "mtkNativeDesigner"): Reports the results produced by the process.

### Details

We can construct an object of the mtkNativeDesigner class from the following situations:

1. The designer is provided within the package "mtk";
2. The designer is provided as an R function implemented outside the package "mtk"; If so, the R function must produce a result as a named list with two elements: X and information, where X is a date.frame containing the analysis result and information is a named list containing supplementary information about the analysis process.
3. The experiments design is produced off-line and available as a data.frame. We just want to use the "mtk" package for reporting.

For detail uses, see examples from help(mtkNativeEvaluator).

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Create a native designer with the method "Morris"
# implemented in the package "mtk"

designer <- mtkNativeDesigner(
design ="Morris",
information=list(size=20)
)
```

---

## mtkNativeEvaluator *The constructor of the class* mtkNativeEvaluator

---

### Description

The constructor.

### Usage

```
mtkNativeEvaluator(model=NULL, Y=NULL, information=NULL)
```

### Arguments

| | |
|---|---|
| model | NULL, an R function or a string to specify the model to simulate. |
| Y | NULL or a data.frame to load the results of model simulation produced off-line. |
| information | a named list to provide with supplementary information about the simulation produced off-line or the parameters used by the evaluator. |

## Value

an object of the mtkNativeEvaluator class

## Details

We can construct an object of the mtkNativeEvaluator class from the following situations:

- The model is provided within the package "mtk"The argument "model" takes a string giving the model to simulate, and the argument "information" gives the list of parameters used for the model simulation. e.g. model <- mtkNativeEvaluator( model="Ishigami").
- The model is provided with an R function implemented outside the package "mtk"The argument "model" takes the R function, the argument "information" may be used to give supplementary information about the R function.
- The simulation results are produced off-line and available as a data.frameThe argument "model" is not used, the argument "Y" takes the data.frame holding the model simulation, and the argument "information" may be omitted or simply used to give supplementary information about the simulation process. e.g. model <- mtkNativeDesigner( Y = simultedData, information = list(model = "Ishigami")).

For details uses, see examples from `?class(mtkNativeEvaluator)`.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## See Also

`?class(mtkNativeEvaluator)`

## Examples

```
## 1) Create a model simulation  with the model "Ishigami" implemented in the package "mt
evaluator <- mtkNativeEvaluator(model="Ishigami")

## 2) Create a model simulation  with a R function implemented outside the package "mtk"

# a) Create a R function to represent the model of population

ME <- function(K, Y0, a, t=5, ...){

res <- exp(-a*t)
res <- Y0+res*(K-Y0)
res <- K*Y0/res
out <- as.integer(res)

return(out)
}
```

```
# b) Do the sensitivity analysis for the function "ME"

K <- make.mtkFactor(name="K", nominal=400, distribName="unif",
distribPara=list(min=100, max=1000))
Y0 <- make.mtkFactor(name="Y0", nominal=20, distribName="unif",
distribPara=list(min=1, max=40))
a <- make.mtkFactor(name="a", nominal=0.1, distribName="unif",
distribPara=list(min=0.05, max=0.2))
factors <- mtkExpFactors(list(K,Y0,a))


plan <- mtkNativeDesigner ("BasicMonteCarlo",
information=c(size=500))

model <- mtkNativeEvaluator(model=ME, information=c(t=5))

index<- mtkNativeAnalyser("Regression", information=c(nboot=20) )

expt <- mtkExpWorkflow( expFactors=factors,
processesVector=c(
design= plan,
evaluate= model,
analyze= index)
)
run(expt)
summary(expt)

## 3) Import the results of model simulation produced off-line into
##    an object of mtkNativeEvaluator

data <- data.frame()
model <- mtkNativeEvaluator(Y=data,
information = list(model="Ishigami"))
```

---

`mtkNativeEvaluator-class`

*The* `mtkNativeEvaluator` *class*

---

## Description

The `mtkNativeEvaluator` class is a sub-class of the class `mtkEvaluator` used to manage the simulation task implemented locally (i.e. tasks don't need to call services from the Web). It provides all the slots and methods defined in the class `mtkEvaluator`.

## Class Hierarchy

**Parent classes :** `mtkEvaluator`

**Direct Known Subclasses :**

## Constructor

`mtkNativeEvaluator` signature(model=NULL, Y=NULL, information=NULL)

## Slots

`model`: (ANY) a string, an R fonction, or NULL to inform the model to simulate.

`name`: (`character`) always takes the string "evaluate".

`protocol`: (`character`) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes "R".

`site`: (`character`) a string to indicate where the service is located. Here, it always takes "mtk".

`service`: (`character`) a string to name the service to invoke.

`parameters`: (`vector`) a vector of [`mtkParameter`] containing the parameters to pass while calling the service.

`ready`: (`logical`) a logical to tell if the process is ready to run.

`state`: (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`: (ANY) a data holder to hold the results produced by the process

## Methods

`setName` signature(this = "mtkNativeEvaluator", name = "character"): Not used, method inherited from the parent class.

`setParameters` signature(this = "mtkNativeEvaluator", f = "vector"): Assigns new parameters to the process.

`getParameters` signature(this = "mtkNativeEvaluator"): Returns the parameters as a named list.

`is.ready` signature( = "mtkNativeEvaluator"): Tests if the process is ready to run.

`setReady` signature(this = "mtkNativeEvaluator", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkNativeEvaluator"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkNativeEvaluator", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkNativeEvaluator"): Returns the results produced by the process as a [`mtkEvaluatorResult`].

`getData` signature(this = "mtkNativeEvaluator"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkNativeEvaluator"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkNativeEvaluator", context= "mtkExpWorkflow"): runs the simulation.

`summary` signature(object = "mtkNativeEvaluator"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkNativeEvaluator"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkNativeEvaluator"): Plots the results produced by the process.

`report` signature(this = "mtkNativeEvaluator"): Reports the results produced by the process.

## Details

We can construct an object of the `mtkNativeEvaluator` class from the following situations: 1) 2) 3) the experimental design is produced off-line and available as a data.frame.

We can construct an object of the `mtkNativeEvaluator` class from the following situations:

1. The evaluator is provided within the package "mtk";
2. The evaluator is provided as an R function outside the package "mtk";
3. The simulation is carried out off-line. We just want to use the "mtk" package for reporting.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
## 1) Create a model simulation  with the model "Ishigami" implemented in the package "mt

evaluator <- mtkNativeEvaluator(model="Ishigami")

## 2) Create a model simulation  with a R function implemented outside the package "mtk"

# a) Create a R function to represent the model of population

ME <- function(K, Y0, a, t=5, ...){

res <- exp(-a*t)
res <- Y0+res*(K-Y0)
res <- K*Y0/res
out <- as.integer(res)

return(out)
}
# b) Do the sensitivity analysis for the function "ME"

K <- make.mtkFactor(name="K", nominal=400, distribName="unif",
distribPara=list(min=100, max=1000))
Y0 <- make.mtkFactor(name="Y0", nominal=20, distribName="unif",
distribPara=list(min=1, max=40))
a <- make.mtkFactor(name="a", nominal=0.1, distribName="unif",
distribPara=list(min=0.05, max=0.2))
factors <- mtkExpFactors(list(K,Y0,a))


plan <- mtkNativeDesigner ("BasicMonteCarlo",
information=c(size=500))

model <- mtkNativeEvaluator(model=ME, information=c(t=5))
```

```
index<- mtkNativeAnalyser("Regression", information=c(nboot=20) )

expt <- mtkExpWorkflow( expFactors=factors,
processesVector=c(
design= plan,
evaluate= model,
analyze= index)
)
run(expt)
summary(expt)

## 3) Import the results of model simulation produced off-line
##    into an object of mtkNativeEvaluator

data <- data.frame()
model <- mtkNativeEvaluator(Y=data,
information = list(model="Ishigami"))
```

---

`mtkParameter` *The constructor of the class* `mtkParameter`

---

## Description

The constructor of the class `mtkParameter`. See alos `make.mtkParameterList`

## Usage

```
mtkParameter(name='unknown', type='logical', val=NULL)
```

## Arguments

| | |
|---|---|
| `name` | (`character`) the name of the parameter. |
| `type` | (`character`) the type of the parameter such as 'numeric', 'double', 'logical', etc.. |
| `val` | (`ANY`) the value of the parameter. |

## Value

an object of the `mtkParameter` class

## Author(s)

Juhui WANG, MIA-jouy, INRA

## Examples

```
# Create an object of the 'mtkParameter' class.

p <- mtkParameter(name="x", type="double", val=0.0)

# We usually use the 'make.mtkParameterList()' function to define
# a list of 'mtkParameter' instead of the constructor
# of the 'mtkParameter' class
flist <- make.mtkParameterList(x=list(min=-1,max=+1))
```

mtkParameter-class *The* mtkParameter *class*

## Description

The mtkParameter class is a class used to manage the parameter concept.

## Class Hierarchy

**Parent classes :** mtkValue

**Direct Known Subclasses :**

## Constructor

mtkParameter signature(name='unknown', type='logical', val=NULL)

make.mtkParameterList signature(x=list())

## Slots

name**:** (character) the name of the parameter.

type**:** (character) the type of the parameter.

val**:** (ANY) the value of the parameter.

## Methods

getName signature( this = "mtkParameter"): Returns the value of the slot "name".

getValue signature( this = "mtkParameter"): Returns the value of the slot "val".

getType signature(this = "mtkParameter"): Returns the value of the slot "type".

setName signature( this = "mtkParameter", name="character"): Gives a new value to the slot "name".

setValue signature( this = "mtkParameter", val="ANY"): Gives a new value to the slot "val".

setType signature(this = "mtkParameter", type="character"): Gives a new value to the slot "type".

show signature( object = "mtkParameter"): Prints a report of the data managed by the underlying object.

print signature(x = "mtkParameter"): Prints the information managed by the underlying object.

## Author(s)

Juhui WANG, MIA-jouy, INRA

## Examples

```
# Create an object of the 'mtkParameter' class.

p <- mtkParameter(name="x", type="double", val=0.0)

# We usually use the 'make.mtkParameterList()' function to define a list of
# 'mtkParameter' instead of the constructor
# of the 'mtkParameter' class
plist <- make.mtkParameterList(list(min=-1,max=+1,shape="hello"))
```

mtkParsor *The constructor of the class* mtkParsor

## Description

The constructor

## Usage

```
mtkParsor(xmlPath="")
```

## Arguments

| | |
|---|---|
| xmlPath | a string to specify the XML file to parse. |

## Value

an object of the mtkParsor class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a parsor with the file "inst/extdata/WWDM.xml".

# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"

#  find where the examples are held.
#  This is only necessary for the example since the system does
#  not know where the file "WWDM.xml" is kept.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",xmlFile,sep = "")

## Create a parsor from the xml file
parsor <- mtkParsor(xmlFilePath)

# Create an empty workflow.
workflow <- mtkExpWorkflow()

# Parse the XML file and initialize the workflow
# with the data extracted from the XML file.
run(parsor, workflow)

# Run the workflow and report the results of the sensitivity analysis
```

```
run(workflow)
summary(workflow)
```

---

mtkParsor-class *The* mtkParsor *class*

---

## Description

The mtkParsor class is the main class used to parse the XML files used in the "mtk" package. It provides a generic way to communicate with the plate-form of model simulation.

## Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :**

## Constructor

mtkParsor signature(xmlPath="")

## Slots

xmlPath**:** (character) the XML file's path and name.

## Methods

setXMLFilePath signature(this = "mtkParsor", xmlPath = "character"): Sets the xml File.

run signature(this = "mtkParsor", context = "mtkExpWorkflow"): Parses the XML file and fills the workflow defined in the "context" argument with the data extracted from the XML file.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a parsor with the file "inst/extdata/WWDM.xml".

# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"

#  find where the examples are held.
#  This is only necessary for the example since the system does
#  not know where the file "WWDM.xml" is kept.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),
```

```
"/extdata/",xmlFile,sep = "")

## Create a parsor from the xml file
parsor <- mtkParsor(xmlFilePath)

# Create an empty workflow.
workflow <- mtkExpWorkflow()

# Parse the XML file and initialize the workflow
# with the data extracted from the XML file.
run(parsor, workflow)

# Run the workflow and report the results of the sensitivity analysis

run(workflow)
summary(workflow)
```

---

`mtkPLMMAnalyser` *The constructor of the class* `mtkPLMMAnalyser`

---

## Description

The constructor

## Usage

```
mtkPLMMAnalyser(mtkParameters = NULL, listParameters = NULL)
```

## Arguments

`mtkParameters`
: a vector of [`mtkParameter`] representing the parameters necessary to run the process.

`listParameters`
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

## Value

an object of the `mtkPLMMAnalyser` class

## Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# see examples with help(PLMM)
```

---

`mtkPLMMAnalyser-class`
*The* `mtkPLMMAnalyser` *class*

---

### Description

The `mtkPLMMAnalyser` class is a sub-class of the class `mtkAnalyser`. It implements the sensitivity analysis method PLMM and provides all the slots and methods defined in the class `mtkAnalyser`.

### Class Hierarchy

**Parent classes :** `mtkAnalyser`

**Direct Known Subclasses :**

### Constructor

`mtkPLMMAnalyser` signature(mtkParameters = NULL, listParameters = NULL)

### Slots

`name`**:** (`character`) always takes the string "analyze".

`protocol`**:** (`character`) always takes the string "R".

`site`**:** (`character`) always takes the string "mtk".

`service`**:** (`character`) always takes the string "PLMM".

`parameters`**:** (`vector`) a vector of [`mtkParameter`] containing the parameters to pass while calling the service.

`ready`**:** (`logical`) a logical to tell if the process is ready to run.

`state`**:** (`logical`) a logical to tell if the results produced by the process are available and ready to be consumed.

`result`**:** (`ANY`) a data holder to hold the results produced by the process

### Methods

`setName` signature(this = "mtkPLMMAnalyser", name = "character"): Not used, method inherited from the parent class.

`setParameters` signature(this = "mtkPLMMAnalyser", f = "vector"): Assigns new parameters to the process.

`getParameters` signature(this = "mtkPLMMAnalyser"): Returns the parameters as a named list.

`is.ready` signature( = "mtkPLMMAnalyser"): Tests if the process is ready to run.

`setReady` signature(this = "mtkPLMMAnalyser", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkPLMMAnalyser"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkPLMMAnalyser", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkPLMMAnalyser"): Returns the results produced by the process as a [`mtkPLMMAnalyserResult`].

`getData` signature(this = "mtkPLMMAnalyser"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkPLMMAnalyser"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkPLMMAnalyser", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

`summary` signature(object = "mtkPLMMAnalyser"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkPLMMAnalyser"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkPLMMAnalyser"): Plots the results produced by the process.

`report` signature(this = "mtkPLMMAnalyser"): Reports the results produced by the process.

## Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples with help(PLMM)
```

---

# `mtkPLMMAnalyserResult`

*The constructor of the class* `mtkPLMMAnalyserResult`

---

## Description

The constructor

## Usage

```
mtkPLMMAnalyserResult(main,information=NULL)
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the results of the sensitivity analysis produced by the PLMM analyser. |
| information | a named list containing the information about the managed data. |

### Value

an object of the mtkPLMMAnalyserResult class

### Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# see examples with help(PLMM)
```

---

## mtkPLMMAnalyserResult-class

*The* mtkPLMMAnalyserResult *class*

---

### Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method PLMM.

### Class Hierarchy

**Parent classes :** mtkAnalyserResult

**Direct Known Subclasses :**

### Constructor

mtkPLMMAnalyserResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkPLMMAnalyserResult"): Provides a summary of the experimental design produced by the analyser.

print signature(x = "mtkPLMMAnalyserResult"): Prints a report of the experimental design produced by the analyser.

plot signature(x = "mtkPLMMAnalyserResult"): Plots the experimental design produced by the analyser.

### Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# see examples with help(PLMM)
```

---

## mtkProcess *The constructor of the mtkProcess class*

---

### Description

The constructor

### Usage

```
mtkProcess(
name,
protocol = "R",
site = "mtk",
service = "",
parameters = NULL,
ready = FALSE,
state = FALSE,
result = NULL
)
```

## Arguments

| | |
|---|---|
| name | the processing step associated with this process. It may be "design", "evaluate", or "analyze". |
| protocol | a string from "http", "system", "R" respectively representing if the process is implemented remotety, locally or as R function. |
| site | the site where the process is implemented if remotely or the package where the process is implemented if as a R function. |
| service | the service name or a system call that implements the process. |
| parameters | a vector of [mtkParameter] representing the parameters necessary to run the process. |
| ready | a logical to indicate if the process is ready to run. |
| state | a logical to indicate if the process finished running and the results are available. |
| result | an object of a class derived from [mtkResult] to hold the results produced by the process. |

## Value

an object of the mtkProcess class

## Details

The mtkProcess class is a virtual class to manage the generic properties of processes involved in the "mtk" package.

For details uses, see examples from help(mtkNativeDesigner), help(mtkNativeEvaluator), help(mtkNativeAnalyser),.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples with help(mtkNativeDesigner)
```

mtkProcess-class *The* mtkProcess *class*

## Description

The mtkProcess is a class to represent the processes managed within the workflow. It provides a generic mechanism for conceptualizing the common behavior of the processes used in experimental design, model simulation and sensitivity analysis.

## Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :** mtkDesigner,mtkEvaluator, mtkAnalyser

## Constructor

mtkProcess signature(name, protocol = "R", site = "mtk", service = "", parameters = NULL, ready = FALSE, state = FALSE, result = NULL)

## Slots

name**:** (character) a string to name the step of the analysis: "design", "evaluate" or "analyze".

protocol**:** (character) a string to name the protocol used to run the process: "http", "system", "R", etc.

site**:** (character) a string to indicate where the service is located: "mtk", URI, etc.

service**:** (character) a string to name the service to invoke.

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to keep the results produced by the process

## Methods

setName signature(this = "mtkProcess", name = "character"): Gives a name to the process.

getName signature(this = "mtkProcess"): Returns the name of the process.

setParameters signature(this = "mtkProcess", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkProcess"): Returns the parameters as a named list.

is.ready signature(this = "mtkProcess"): Tests if the process is ready to run.

setReady signature(this = "mtkProcess", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkProcess"): Tests if the results produced by the process are available.

setReady signature(this = "mtkProcess", state = "logical"): Marks the process as already executed.

getResult signature(this = "mtkProcess"): Returns the results produced by the process as a mtkResult.

`getData` signature(this = "mtkProcess") : Returns the results produced by the process as a data frame.

`serializeOn` signature(this = "mtkProcess"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkProcess", context= "mtkExpWorkflow"): Runs the process.

`summary` signature(object = "mtkProcess", ...): Displays a summary of the results produced by the process.

`print` signature(x = "mtkProcess"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkProcess", y, ...): Plots the results produced by the process.

`report` signature(this = "mtkProcess"): Reports the results produced by the process.

## Details

The `mtkProcess` class is a virtual class to manage the generic properties of processes involved in the "mtk" package.

For details uses, see examples from `help(mtkNativeDesigner)`, `help(mtkNativeEvaluator)`, `help(mtkNativeAnalyser)`,.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples with help(mtkNativeDesigner)
```

---

# `mtkRandLHSDesigner` *The constructor of the class* `mtkRandLHSDesigner`

---

## Description

The constructor

## Usage

```
mtkRandLHSDesigner(mtkParameters = NULL, listParameters = NULL)
```

### Arguments

mtkParameters
: a vector of [mtkParameter] representing the parameters necessary to run the process.

listParameters
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the mtkRandLHSDesigner class

### See Also

package?lsh, help(LHS)

### Examples

```
# To do, example for LHS method
```

---

## mtkRandLHSDesigner-class

*The* mtkRandLHSDesigner *class*

---

### Description

The mtkRandLHSDesigner class is a sub-class of the class mtkDesigner. It implements the method RandLHS and provides all the slots and methods defined in the class mtkDesigner.

### Class Hierarchy

**Parent classes :** mtkDesigner

**Direct Known Subclasses :**

### Constructor

mtkRandLHSDesigner signature(mtkParameters = NULL, listParameters = NULL)

### Slots

name**:** (character) always takes the string "design".

protocol**:** (character) always takes the string "R".

site**:** (character) always takes the string "mtk".

service**:** (character) always takes the string "RandLHS".

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

### Methods

setName signature(this = "mtkRandLHSDesigner", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkRandLHSDesigner", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkRandLHSDesigner"): Returns the parameters as a named list.

is.ready signature( = "mtkRandLHSDesigner"): Tests if the process is ready to run.

setReady signature(this = "mtkRandLHSDesigner", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkRandLHSDesigner"): Tests if the results produced by the process are available.

setReady signature(this = "mtkRandLHSDesigner", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkRandLHSDesigner"): Returns the results produced by the process as a [mtkRandLHSDesignerResult].

getData signature(this = "mtkRandLHSDesigner"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkRandLHSDesigner"): Returns all data managed by the process as a named list.

run signature(this = "mtkRandLHSDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

summary signature(object = "mtkRandLHSDesigner"): Provides a summary of the results produced by the process.

print signature(x = "mtkRandLHSDesigner"): Prints a report of the results produced by the process.

plot signature(x = "mtkRandLHSDesigner"): Plots the results produced by the process.

report signature(this = "mtkRandLHSDesigner"): Reports the results produced by the process.

### See Also

package?lsh, help(LHS)

### Examples

```
# To do, example for LHS method
```

---

## mtkRandLHSDesignerResult

*The constructor of the class* mtkRandLHSDesignerResult

---

### Description

The constructor

### Usage

```
mtkRandLHSDesignerResult(main,information=NULL)
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the experimental design produced by the designer. |
| information | a named list containing the information about the managed data. |

### Value

an object of the mtkRandLHSDesignerResult class

### See Also

package?lsh, help(LHS)

### Examples

```
# To do, example for LHS method
```

---

## mtkRandLHSDesignerResult-class

*The* mtkRandLHSDesignerResult *class*

---

### Description

A class to collect the experimental design produced by the designer implementing the method RandLHS.

### Class Hierarchy

**Parent classes :** mtkDesignerResult

**Direct Known Subclasses :**

### Constructor

mtkRandLHSDesignerResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkRandLHSDesignerResult"): Provides a summary of the experimental design produced by the designer.

print signature(x = "mtkRandLHSDesignerResult"): Prints a report of the experimental design produced by the designer.

plot signature(x = "mtkRandLHSDesignerResult"): Plots the experimental design produced by the designer.

### See Also

package?lsh, help(LHS)

### Examples

```
# To do, example for LHS method
```

---

## mtkReadFactors-methods

*The* mtkReadFactor *method*

---

### Description

a list of factors

### Usage

```
mtkReadFactors(file, path)
```

### Arguments

| | |
|---|---|
| file | the name of the file to read. |
| path | the path to the file to read. |

### Value

an object of the class mtkDomain

### Author(s)

Hervé Richard, BioSP, INRA, Domaine Saint paul, 84914 Avignon Cedex 9

### Examples

```
# see examples for the \code{\linkS4class{mtkExpFactors}} class.
```

---

## mtkRegressionAnalyser

*The constructor of the class* mtkRegressionAnalyser

---

### Description

The constructor

### Usage

```
 mtkRegressionAnalyser(
mtkParameters = NULL,
listParameters = NULL
)
```

### Arguments

`mtkParameters`
: a vector of [`mtkParameter`] representing the parameters necessary to run the process.

`listParameters`
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the `mtkRegressionAnalyser` class

### See Also

`help(morris, sensitivity)` and `help(Regression)`

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Monte-Carlo" and "Regression" m


# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp.designer <- mtkBasicMonteCarloDesigner (listParameters=list(size=20))

#   2) the simulation process
exp.evaluator <- mtkIshigamiEvaluator()

#   3) the analysis process
exp.analyser <- mtkRegressionAnalyser(listParameters=list(nboot=20) )

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp.designer,
evaluate=exp.evaluator, analyze=exp.analyser))

#  Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

## mtkRegressionAnalyser-class

*The* `mtkRegressionAnalyser` *class*

---

## Description

The mtkRegressionAnalyser class is a sub-class of the class mtkAnalyser. It implements the sensitivity analysis method Regression and provides all the slots and methods defined in the class mtkAnalyser.

## Class Hierarchy

**Parent classes :** mtkAnalyser

**Direct Known Subclasses :**

## Constructor

mtkRegressionAnalyser signature(mtkParameters = NULL, listParameters = NULL)

## Slots

name**:** (character) always takes the string "analyze".

protocol**:** (character) always takes the string "R".

site**:** (character) always takes the string "mtk".

service**:** (character) always takes the string "Regression".

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

## Methods

setName signature(this = "mtkRegressionAnalyser", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkRegressionAnalyser", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkRegressionAnalyser"): Gets the parameters as a named list.

is.ready signature( = "mtkRegressionAnalyser"): Tests if the process is ready to run.

setReady signature(this = "mtkRegressionAnalyser", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkRegressionAnalyser"): Tests if the results produced by the process are available.

setReady signature(this = "mtkRegressionAnalyser", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkRegressionAnalyser"): Returns the results produced by the process as a [mtkRegressionAnalyserResult].

getData signature(this = "mtkRegressionAnalyser"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkRegressionAnalyser"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkRegressionAnalyser", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

`summary` signature(object = "mtkRegressionAnalyser"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkRegressionAnalyser"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkRegressionAnalyser"): Plots the results produced by the process.

`report` signature(this = "mtkRegressionAnalyser"): Reports the results produced by the process.

## See Also

`help(morris, sensitivity)` and `help(Regression)`

## Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Monte-Carlo" and "Regression" m


# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp.designer <- mtkBasicMonteCarloDesigner (listParameters=list(size=20))

#   2) the simulation process
exp.evaluator <- mtkIshigamiEvaluator()

#   3) the analysis process
exp.analyser <- mtkRegressionAnalyser(listParameters=list(nboot=20) )

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp.designer,
evaluate=exp.evaluator, analyze=exp.analyser))

#  Run the workflow and report the results.
run(exp1)
print(exp1)
```

---

# mtkRegressionAnalyserResult

*The constructor of the class* `mtkRegressionAnalyserResult`

---

## Description

The constructor

### Usage

```
mtkRegressionAnalyserResult(main,information=NULL)
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the results of the sensitivity analysis produced by the analyser. |
| information | a named list containing the information about the managed data. |

### Value

an object of the mtkRegressionAnalyserResult class

### See Also

help(morris, sensitivity) and help(Regression)

### Examples

```
## See examples from help(mtkAnalyserResult)
```

---

## mtkRegressionAnalyserResult-class

*The* mtkRegressionAnalyserResult *class*

---

### Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Regression.

### Class Hierarchy

**Parent classes :** mtkAnalyserResult

**Direct Known Subclasses :**

### Constructor

mtkRegressionAnalyserResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkRegressionAnalyserResult"): Provides a summary of the experimental design produced by the analyser.

print signature(x = "mtkRegressionAnalyserResult"): Prints a report of the experimental design produced by the analyser.

plot signature(x = "mtkRegressionAnalyserResult"): Plots the experimental design produced by the analyser.

### See Also

help(morris, sensitivity) and help(Regression)

### Examples

```
## See examples from help(mtkAnalyserResult)
```

---

## mtkResult *The constructor of the class* mtkResult

---

### Description

The constructor

### Usage

```
mtkResult(information=list())
```

### Arguments

information a named list containing the information about the managed data.

### Value

an object of the mtkResult class

### Details

The mtkResult class is a virtual class to manage the generic properties of results produced by the processes involved in the "mtk" package.

For details uses, see examples from help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorResult).

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## See Also

help(mtkAnalyserResult),help(mtkDesignerResult),help(mtkEvaluatorResult)

## Examples

```
## See examples from help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorR
```

---

# mtkResult-class *The* mtkResult *class*

---

## Description

A general and simple class to collect the results produced by diverse processes involved in the "mtk" package.

## Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :** mtkDesignerResult,mtkEvaluatorResult, etc.

## Constructor

mtkResult signature(information=list())

## Slots

information**:** (list) a named list containing information about the managed data.

## Methods

summary signature(object = "mtkResult"): Provides a summary report about the managed data.

serializeOn signature(this = "mtkResult"): Returns all managed data as a named list.

## Details

The mtkResult class is a virtual class to manage the generic properties of results produced by the processes involved in the "mtk" package.

For details uses, see examples from help(mtkAnalyserResult),help(mtkDesignerResult), help(mtkEvaluatorResult).

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### See Also

`help(mtkAnalyserResult),help(mtkDesignerResult),help(mtkEvaluatorResult)`

### Examples

```
## See examples from help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorR
```

---

## `mtkSobolAnalyser` *The constructor of the class* `mtkSobolAnalyser`

---

### Description

The constructor

### Usage

```
mtkSobolAnalyser(mtkParameters = NULL, listParameters = NULL)
```

### Arguments

`mtkParameters`
: a vector of [`mtkParameter`] representing the parameters necessary to run the process.

`listParameters`
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the `mtkSobolAnalyser` class

### References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

### See Also

help(sobol2002, sensitivity) and help(Sobol)

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method
```

---

## mtkSobolAnalyser-class

*The* mtkSobolAnalyser *class*

---

### Description

The mtkSobolAnalyser class is a sub-class of the class mtkAnalyser. It implements the sensitivity analysis method Sobol and provides all the slots and methods defined in the class mtkAnalyser.

### Class Hierarchy

**Parent classes :** mtkAnalyser

**Direct Known Subclasses :**

### Constructor

mtkSobolAnalyser signature(mtkParameters = NULL, listParameters = NULL)

### Slots

name**:** (character) always takes the string "analyze".

protocol**:** (character) always takes the string "R".

site**:** (character) always takes the string "mtk".

service**:** (character) always takes the string "Sobol".

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

### Methods

setName signature(this = "mtkSobolAnalyser", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkSobolAnalyser", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkSobolAnalyser"): Returns the parameters as a named list.

is.ready signature( = "mtkSobolAnalyser"): Tests if the process is ready to run.

`setReady` signature(this = "mtkSobolAnalyser", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkSobolAnalyser"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkSobolAnalyser", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkSobolAnalyser"): Returns the results produced by the process as a [`mtkSobolAnalyserResult`].

`getData` signature(this = "mtkSobolAnalyser"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkSobolAnalyser"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkSobolAnalyser", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

`summary` signature(object = "mtkSobolAnalyser"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkSobolAnalyser"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkSobolAnalyser"): Plots the results produced by the process.

`report` signature(this = "mtkSobolAnalyser"): Reports the results produced by the process.

## References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

## See Also

`help(sobol, sensitivity)` and `help(Sobol)`

## Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method
```

---

# mtkSobolAnalyserResult

*The constructor of the class* `mtkSobolAnalyserResult`

---

## Description

The constructor

## Usage

```
mtkSobolAnalyserResult(main,information=NULL)
```

### Arguments

| | |
|---|---|
| main | a data.frame holding the results of the sensitivity analysis produced by the analyser. |
| information | a named list containing the information about the managed data. |

### Value

an object of the mtkSobolAnalyserResult class

### See Also

help(mtkAnalyserResult) and help(Sobol)

### Examples

```
## See examples from help(mtkAnalyserResult).
```

---

## mtkSobolAnalyserResult-class

*The* mtkSobolAnalyserResult *class*

---

### Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Sobol.

### Class Hierarchy

**Parent classes :** mtkAnalyserResult

**Direct Known Subclasses :**

### Constructor

mtkSobolAnalyserResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkSobolAnalyserResult"): Provides a summary of the experimental design produced by the analyser.

print signature(x = "mtkSobolAnalyserResult"): Prints a report of the experimental design produced by the analyser.

plot signature(x = "mtkSobolAnalyserResult"): Plots the experimental design produced by the analyser.

### See Also

help(mtkAnalyserResult) and help(Sobol)

### Examples

```
## See examples from help(mtkAnalyserResult).
```

---

## mtkSobolDesigner *The constructor of the class* mtkSobolDesigner

---

### Description

The constructor

### Usage

```
mtkSobolDesigner(mtkParameters = NULL, listParameters = NULL)
```

### Arguments

| | |
|---|---|
| mtkParameters | a vector of [mtkParameter] representing the parameters necessary to run the process. |
| listParameters | a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters. |

### Value

an object of the mtkSobolDesigner class

### References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

### See Also

help(sobol2002, sensitivity) and help(Sobol)

### Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method
```

mtkSobolDesigner-class
*The* mtkSobolDesigner *class*

## Description

This class is a sub-class of the class mtkDesigner. It implements the sampling method 'Sobol' and provides all the slots and methods defined in the class mtkDesigner.

## Class Hierarchy

**Parent classes :** mtkDesigner

**Direct Known Subclasses :**

## Constructor

mtkSobolDesigner signature(mtkParameters = NULL, listParameters = NULL)

## Slots

name**:** (character) always takes the string "design".

protocol**:** (character) always takes the string "R".

site**:** (character) always takes the string "mtk".

service**:** (character) always takes the string "Sobol".

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

## Methods

setName signature(this = "mtkSobolDesigner", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkSobolDesigner", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkSobolDesigner"): Returns the parameters as a named list.

is.ready signature( = "mtkSobolDesigner"): Tests if the process is ready to run.

setReady signature(this = "mtkSobolDesigner", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkSobolDesigner"): Tests if the results produced by the process are available.

setReady signature(this = "mtkSobolDesigner", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkSobolDesigner"): Returns the results produced by the process as a [mtkSobolDesignerResult].

`getData` signature(this = "mtkSobolDesigner"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkSobolDesigner"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkSobolDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

`summary` signature(object = "mtkSobolDesigner"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkSobolDesigner"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkSobolDesigner"): Plots the results produced by the process.

`report` signature(this = "mtkSobolDesigner"): Reports the results produced by the process.

## References

1. Campolongo, F., J. Cariboni, and A. Saltelli (2007). An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software, 22, 1509–1518.
2. A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

## See Also

`help(sobol, sensitivity)` and `help(Sobol)`

## Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method
```

---

# mtkSobolDesignerResult

*The constructor of the class* `mtkSobolDesignerResult`

---

## Description

The constructor

## Usage

```
mtkSobolDesignerResult(main,information=NULL)
```

## Arguments

| | |
|---|---|
| `main` | a data.frame holding the experimental design produced by the designer. |
| `information` | a named list containing the information about the managed data. |

## Value

an object of the `mtkSobolDesignerResult` class

### See Also

help(mtkDesignerResult) and help(Sobol)

### Examples

```
## See examples from help(mtkDesignerResult).
```

---

## mtkSobolDesignerResult-class

*The* mtkSobolDesignerResult *class*

---

### Description

A class to collect the experimental design produced by the Designer implementing the method Sobol.

### Class Hierarchy

**Parent classes :** mtkDesignerResult

**Direct Known Subclasses :**

### Constructor

mtkSobolDesignerResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the experimental design.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkSobolDesignerResult"): Provides a summary of the experimental design produced by the designer.

print signature(x = "mtkSobolDesignerResult"): Prints a report of the experimental design produced by the designer.

plot signature(x = "mtkSobolDesignerResult"): Plots the experimental design produced by the designer.

### See Also

help(mtkDesignerResult) and help(Sobol)

### Examples

```
## See examples from help(mtkDesignerResult).
```

`mtkSystemEvaluator` *The constructor of the class* `mtkSystemEvaluator`

## Description

The constructor

## Usage

```
mtkSystemEvaluator(
service = "",
mtkParameters = NULL,
listParameters = NULL
)
```

## Arguments

| | |
|---|---|
| `service` | a string specifying the way to invoke the application implementing the model. |
| `mtkParameters` | a vector of [`mtkParameter`] representing the parameters necessary to run the process. |
| `listParameters` | a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters. |

## Value

an object of the `mtkSystemEvaluator` class

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples
```

mtkSystemEvaluator-class
*The* mtkSystemEvaluator *class*

## Description

The mtkSystemEvaluator class is a sub-class of the class mtkEvaluator used to manage the simulation of the model implemented as a system application.

## Class Hierarchy

**Parent classes :** mtkEvaluator

**Direct Known Subclasses :**

## Constructor

mtkSystemEvaluator signature( service="",mtkParameters=NULL,listParameters = NULL)

## Slots

name**:** (character) always takes the string "evaluate".

protocol**:** (character) always takes the string "system".

site**:** (character) not used here.

service**:** (character) a string to invoke the system command implementing the model.

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while invoking the system command.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

## Methods

setName signature(this = "mtkSystemEvaluator", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkSystemEvaluator", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkSystemEvaluator"): Returns the parameters as a named list.

is.ready signature( = "mtkSystemEvaluator"): Tests if the process is ready to run.

setReady signature(this = "mtkSystemEvaluator", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkSystemEvaluator"): Tests if the results produced by the process are available.

setReady signature(this = "mtkSystemEvaluator", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkSystemEvaluator"): Returns the results produced by the process as a [mtkEvaluatorResult].

getData signature(this = "mtkSystemEvaluator"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkSystemEvaluator"): Returns all data managed by the process as a named list.

run signature(this = "mtkSystemEvaluator", context= "mtkExpWorkflow"): runs the simulation.

summary signature(object = "mtkSystemEvaluator"): Provides a summary of the results produced by the process.

print signature(x = "mtkSystemEvaluator"): Prints a report of the results produced by the process.

plot signature(x = "mtkSystemEvaluator"): Plots the results produced by the process.

report signature(this = "mtkSystemEvaluator"): Reports the results produced by the process.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples
```

---

# mtkSystemEvaluatorResult

*The constructor of the class* mtkSystemEvaluatorResult

---

## Description

The constructor

## Usage

```
mtkSystemEvaluatorResult(main,information=NULL)
```

## Arguments

| | |
|---|---|
| main | a data.frame holding the results produced by the evaluator. |
| information | a named list containing the information about the managed data. |

## Value

an object of the mtkSystemEvaluatorResult class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### Examples

```
# See examples
```

---

## mtkSystemEvaluatorResult-class

*The* mtkSystemEvaluatorResult *class*

---

### Description

A class to collect the results produced by the evaluator implemented as a system application.

### Class Hierarchy

**Parent classes :** mtkEvaluatorResult

**Direct Known Subclasses :**

### Constructor

mtkSystemEvaluatorResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the results produced by the model simulation.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkSystemEvaluatorResult"): Provides a summary of the results produced by the evaluator.

print signature(x = "mtkSystemEvaluatorResult"): Prints a report of the results produced by the evaluator.

plot signature(x = "mtkSystemEvaluatorResult"): Plots the results produced by the evaluator.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### Examples

```
# See examples
```

## mtkValue *The constructor of the class* mtkValue

### Description

The constructor

### Usage

```
mtkValue(name='unknown', type='', val=NULL)
```

### Arguments

| | |
|---|---|
| name | the name of the variable. |
| type | the type of the variable, i.e. double, integer, character, logical, null, etc. |
| val | the value of the variable. It may be a single or a vector of values. |

### Value

an object of the mtkValue class

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create an object of 'mtkValue'

triple <- mtkValue('a', 'double', c(2.5,3.0))
```

## mtkValue-class *The* mtkValue *class*

### Description

The mtkValue class is a virtual class used to manage a triple (name, type, value).

### Class Hierarchy

**Parent classes :**

**Direct Known Subclasses :** mtkParameter, codemtkFeature

### Constructor

mtkValue signature(name='unknown', type='', val=NULL)

### Slots

`name`: (`character`) the name of the variable.

`type`: (`character`) the type of the variable.

`val`: (`ANY`) the value of the variable in the right type. It may be a single value or a vector of values

### Methods

`getName` signature( this = "mtkValue"): Returns the value of the slot "name".

`getValue` signature( this = "mtkValue"): Returns the value of the slot "val".

`getType` signature(this = "mtkValue"): Returns the value of the slot "type".

`setName` signature(this = "mtkValue", name = "character"): Gives a new value to the slot "name".

`setValue` signature(this = "mtkValue", type = "ANY"): Gives a new value to the slot "val".

`setType` signature(this = "mtkValue", type = "character"): Gives a new value to the slot "type".

`show` signature( object = "mtkValue"): Prints a report of the data managed by the underlying object.

`print` signature(x = "mtkValue"): Prints the information managed by the underlying object.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a  new object of 'mtkValue'
d <- mtkValue("a", "double", c(0,1))
getType(d) # gives "double"
getName(d) # gives "a"
getValue(d) # gives (0, 1)

setType(d, 'character')
getValue(d) # gives ("0", "1")

setValue(d, "3.14")
getValue(d) # gives "3.14"
```

---

## `mtkWWDMEvaluator` *The constructor of the class* `mtkWWDMEvaluator`

---

### Description

The constructor

### Usage

```
mtkWWDMEvaluator(mtkParameters = NULL, listParameters = NULL)
```

### Arguments

mtkParameters
: a vector of [mtkParameter] representing the parameters necessary to run the process.

listParameters
: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the mtkWWDMEvaluator class

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

1. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.
2. R. Faivre, D. Makowski, J. Wang, H. Richard, R. Monod (2013). Exploration numérique d'un modèle agronomique avec le package mtk. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### See Also

help(WWDM)

### Examples

```
# Carry out a sensitivity analysis with the WWDM model

## Input the factors
data(WWDM.factors)

## Specify the experiments designer
designer <- mtkMorrisDesigner (
listParameters = list(type="oat", levels=5, grid.jump=3, r=10)
)

## Specify the model simulator
model <- mtkWWDMEvaluator(
listParameters = list(year=3, tout=FALSE)
)

## Specify the sensiticity analyser
analyser <- mtkMorrisAnalyser()

## Specify the workflow
exp <- new("mtkExpWorkflow", expFactors=WWDM.factors,
```

```
    processesVector=c(
                design=designer,
                evaluate=model,
                analyze=analyser)
    )
## Run and report the results
run(exp)
summary(exp)
```

---

mtkWWDMEvaluator-class
*The* mtkWWDMEvaluator *class*

---

## Description

The mtkWWDMEvaluator class is a sub-class of the class mtkEvaluator used to manage the simulation of the model WWDM.

## Class Hierarchy

**Parent classes :** mtkEvaluator

**Direct Known Subclasses :**

## Constructor

mtkWWDMEvaluator signature(mtkParameters = NULL, listParameters = NULL)

## Slots

name**:** (character) always takes the string "evaluate".

protocol**:** (character) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes the character "R".

site**:** (character) a string to indicate where the service is located. Here, it always takes the string "mtk".

service**:** (character) a string to name the service to invoke. Here, it always takes the string "WWDM".

parameters**:** (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service. The WWDM model does not need parameters.

ready**:** (logical) a logical to tell if the process is ready to run.

state**:** (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result**:** (ANY) a data holder to hold the results produced by the process

## Methods

`setName` signature(this = "mtkWWDMEvaluator", name = "character"): Not used, method inherited from the parent class.

`setParameters` signature(this = "mtkWWDMEvaluator", f = "vector"): Assigns new parameters to the process.

`getParameters` signature(this = "mtkWWDMEvaluator"): Returns the parameters as a named list.

`is.ready` signature( = "mtkWWDMEvaluator"): Tests if the process is ready to run.

`setReady` signature(this = "mtkWWDMEvaluator", switch = "logical"): Makes the process ready to run.

`is.ready` signature( = "mtkWWDMEvaluator"): Tests if the results produced by the process are available.

`setReady` signature(this = "mtkWWDMEvaluator", switch = "logical"): Marks the process as already executed.

`getResult` signature(this = "mtkWWDMEvaluator"): Returns the results produced by the process as a [`mtkWWDMEvaluatorResult`].

`getData` signature(this = "mtkWWDMEvaluator"): Returns the results produced by the process as a data.frame.

`serializeOn` signature(this = "mtkWWDMEvaluator"): Returns all data managed by the process as a named list.

`run` signature(this = "mtkWWDMEvaluator", context= "mtkExpWorkflow"): runs the simulation.

`summary` signature(object = "mtkWWDMEvaluator"): Provides a summary of the results produced by the process.

`print` signature(x = "mtkWWDMEvaluator"): Prints a report of the results produced by the process.

`plot` signature(x = "mtkWWDMEvaluator"): Plots the results produced by the process.

`report` signature(this = "mtkWWDMEvaluator"): Reports the results produced by the process.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

1. J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.
2. R. Faivre, D. Makowski, J. Wang, H. Richard, R. Monod (2013). Exploration numérique d'un modèle agronomique avec le package `mtk`. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## See Also

```
help(WWDM)
```

## Examples

```
# Carry out a sensitivity analysis with the WWDM model

## Input the factors
data(WWDM.factors)

## Specify the experiments designer
designer <- mtkMorrisDesigner (
listParameters = list(type="oat", levels=5, grid.jump=3, r=10)
)

## Specify the model simulator
model <- mtkWWDMEvaluator(
listParameters = list(year=3)
)

## Specify the sensiticity analyser
analyser <- mtkMorrisAnalyser()

## Specify the workflow
exp <- new("mtkExpWorkflow", expFactors=WWDM.factors,
   processesVector=c(
               design=designer,
               evaluate=model,
               analyze=analyser)
   )
## Run and report the results
run(exp)
summary(exp)
```

---

# mtkWWDMEvaluatorResult

*The constructor of the class* `mtkWWDMEvaluatorResult`

---

## Description

The constructor

## Usage

```
mtkWWDMEvaluatorResult(main,information=NULL)
```

## Arguments

| | |
|---|---|
| `main` | a data.frame holding the results produced by the evaluator. |
| `information` | a named list containing the information about the managed data. |

## Value

an object of the `mtkWWDMEvaluatorResult` class

### See Also

help(mtkEvaluatorResult) and help(WWDM)

### Examples

```
## See examples from help(mtkEvaluatorResult).
```

---

## mtkWWDMEvaluatorResult-class

*The* mtkWWDMEvaluatorResult *class*

---

### Description

A class to collect the results produced by the evaluator implementing the model WWDM.

### Class Hierarchy

**Parent classes :** mtkEvaluatorResult

**Direct Known Subclasses :**

### Constructor

mtkWWDMEvaluatorResult signature(main,information=NULL)

### Slots

main**:** (data.frame) a data.frame holding the results produced by the model simulation.

information**:** (NULL) a named list containing optional information about the managed data.

### Methods

summary signature(object = "mtkWWDMEvaluatorResult"): Provides a summary of the results produced by the evaluator.

print signature(x = "mtkWWDMEvaluatorResult"): Prints a report of the results produced by the evaluator.

plot signature(x = "mtkWWDMEvaluatorResult"): Plots the results produced by the evaluator.

### See Also

help(mtkEvaluatorResult) and help(WWDM)

### Examples

```
## See examples from help(mtkEvaluatorResult).
```

PLMM *The* PLMM *method for sensitivity analysis*

## Description

A mtk compliant implementation of the PLMM method for sensitivity analysis using polynomial linear metamodelling.

## Usage

- mtkPLMMAnalyser(listParameters = NULL)
- mtkNativeAnalyser(analyze="PLMM", information=NULL)

## Parameters

degree.pol: the maximum degree of polynomials (the sum of the degrees of cross products of polynomials is lower or equal to degree.pol). See details.

rawX: orthogonal polynomials (default value FALSE) or raw polynomials (TRUE). See poly, polym.

numY: the column number of the dependent variable (default is the first column of the dataframe of outputs).

listeX: the column numbers of the dependent variables (default is all the dependent variables).

## Parameters for auxiliary functions

all: all the specific summaries and plots are displayed if TRUE (default is FALSE). Else, see the which option.

which: when all=FALSE, the name of the specific summary or plot. Options are "best" (default), "full", "best.adjustedR2", "full.adjustedR2". See details.

lang: language of the summary and plot ("en" (default) for english, "fr" for french).

digits: number of digits in the summary (default = options()$digits).

colors: colors used in plot (default = c("red", "orange","blue")).

legend.loc: location of the legend in plot (default no legend( NULL), options are "topleft", "topright", ... See help(legend)).

## Details

1. The PLMM metamodelling approach consists in estimating 3 models and comparing the percentage of variance (coefficient of determination) explained by these 3 models. The 3 models are polym(A,B,C), poly(A), polym(B,C) where polym computes orthogonal polynomials. polym(A,B,C) gives the total variance explained by the full metamodel, poly(A) gives the variance that can be explained by factor A only (in the sense of polynomials of A) and polym(B,C) gives the variance not explained by factor A. Total sensitiviy index of factor A is computed as max( R2(poly(A)), 1 - R2(polym(A,B,C)) - R2(polym(B,C))) where R2(M) is the coefficient of determination of model M, and first order sensitivity index as min( R2(poly(A)), 1 - R2(polym(A,B,C)) - R2(polym(B,C))). The PLMM function computes a best model in the sense of stepwise model selection starting with the constant model with direction fixed to both (see stepAIC for more details). Total sensitivity and first order indices are computed in the same. Additional results are givent when using adjusted R2 for both best and full models. Names of the results (needed in which option) are: best, full, best.adjustedR2, full.adjustedR2.

2. Computational aspects: PLMM does not use the polym function (as polym needs time to orthogonalize when the number of factors and the degree of the polynomials are high). The cross products are computed as cross products of one dimensional orthogonal polynomials poly(A) * poly(B) * poly(C). So we have to take care with the selected components of the best model (obtained with a stepwise model selection). Care should be taken for interpreting them because the dependent variables are orthogonalized. This not the case when the rawX option is set to TRUE. To prevent from computational side effects, the input factors are first scaled.
3. The mtk implementation of the PLMM method includes the following classes:
   - mtkPLMMAnalyser: for PLMM analysis processes.
   - mtkPLMMAnalyserResult: to store and manage the analysis results.
4. The mtk implementation of the PLMM method includes the following generic functions:
   - summary: to display summary of analysis results. See parameters for auxiliary functions.
   - plot: to plot analysis results. See parameters for auxiliary functions.
5. Many ways to create a PLMM analyser are available in mtk, but we recommend the following class constructors: mtkPLMMAnalyser or mtkNativeAnalyser.

## References

1. Faivre R., 2013. Exploration par construction de métamodèles. In Faivre R., Iooss B., Mahévas S., Makowski D., Monod H., editors. Analyse de sensibilité et exploration de modèles. Applications aux modèles environnementaux. Collection « Savoir Faire », Quae, Versailles, 37p.

## See Also

```
help(polym, stepAIC)
```

## Examples

```
## Sensitivity analysis of the "Ishigami" model with the "PLMM" method

# Generates the factors
data(Ishigami.factors)

#  Builds the processes and workflow:

#   1) the experimental design process with the method "BasicMonteCarlo".
    exp1.designer  <- mtkNativeDesigner("BasicMonteCarlo", information=list(size=100))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
exp1.analyser <- mtkNativeAnalyser("PLMM", information = list(degree.pol=3,numY=1))

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
 processesVector = c(design=exp1.designer,
               evaluate=exp1.evaluator,
```

```
                analyze=exp1.analyser)
                )

#  Runs the workflow and reports the results.
run(exp1)
summary(exp1)
summary(getProcess(exp1,name="analyze"), lang="fr")
summary(getProcess(exp1,name="analyze"), lang="fr",
 which="full", all=FALSE, digit=4)
extractData(exp1,name="analyze")$best$call
plot(getProcess(exp1,name="analysis"), lang="fr", legend.loc="topleft")
plot(getProcess(exp1,name="analysis"), which="full",
 all=FALSE, legend.loc="topright")

##  Example II:  comparing metamodels of the WWDM model

# Generates the factors
data(WWDM.factors)

# 1) to create a sampler with the Monte-Carlo method

sampler <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=100) )

# 2) to create a simulator with the WWDM model
model <- mtkNativeEvaluator("WWDM" , information = list(year=3))

# 3) to create a partial workflow (design and evaluation)

experience1 <- mtkExpWorkflow(expFactors=WWDM.factors,
processesVector=c(design=sampler, evaluate=model) )
run(experience1)

# 4) to create an "analysor" with the Regression method

analyser1 <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

# to add to the workflow the analyser "Regression"


addProcess(experience1, p = analyser1, name = "analyze")
run(experience1)

# 4bis) to create new analysers PLMM  and to add them to the workflow

experience2 <- experience1

analyser2 <- mtkNativeAnalyser("PLMM")

setProcess(experience2, p = analyser2, name = "analyze")
run(experience2) ;

## to comment out the following lines  to compare others analysers
##  with 'analyser1' and 'analyser2'
# experience4 <- experience3 <- experience2
# analyser3 <- mtkNativeAnalyser("PLMM", information = list(degree.pol = 3))
# analyser4 <- mtkNativeAnalyser("PLMM",
# information = list(degree.pol = 3, rawX = TRUE))
```

```
# setProcess(experience3, p = analyser3, name = "analyze")
# setProcess(experience4, p = analyser4, name = "analyze")
# run(experience3) ; run(experience4)

summary(getProcess(experience1,name="analyze"))
       summary(getProcess(experience2,name="analyze"))
# summary(getProcess(experience3,name="analyze"))
# summary(getProcess(experience4,name="analyze"), digi=3)
```

---

`plot,mtkProcess-method`

*The* `plot` *method*

---

## Description

Plots graphically the results produced by the process.

## Usage

```
plot(x, y, ...)
```

## Arguments

| | |
|---|---|
| `x` | the underlying object of class `mtkProcess` |
| `y` | see `par` for details about the graphical parameter arguments |
| `...` | see `par` for details about the graphical parameter arguments |

## Value

invisble()

## Details

1. The behavior of the `plot` depends on the sub-class where the method is implemented.
2. See the documentation of the particular sub-class for details of what is produced. Use `methods("plot")` to get all the methods for the `plot` generic.
3. See `par` for details about the graphical parameter arguments.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## See Also

```
help(plot
```

## Examples

```
# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
     distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
#   1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
      information=list(r=20,type="oat",levels=4,grid.jump=2))

#   2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   #   3) the analysis process with the default method.
#      Here, it is the Morris method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the  workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
     processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))
# Run the workflow and plot the results.
run(exp1)
plot(exp1)

# Extract a process and report its results

p <- getProcess(exp1, "analyze")
plot(p)
```

---

`print,mtkProcess-method`
*The* `print` *method*

---

## Description

Prints a report of the results produced by the process.

## Usage

```
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | the underlying object of class `mtkProcess`. |
| `...` | see the documentation of the function: `base::print()`. |

### Value

invisble()

### Details

1. The behavior of the `print` depends on the sub-class where the method is implemented.
2. See the documentation of the particular sub-class for details of what is produced.
3. Use `methods("print")` to get all the methods for the `print` generic.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### See Also

`help(print)`

### Examples

```
# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
     distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
#   1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
      information=list(r=20,type="oat",levels=4,grid.jump=2))

#   2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   #   3) the analysis process with the default method.
#      Here, it is the Morris method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the  workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
     processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))
# Run the workflow and plot the results.
run(exp1)
print(exp1)

# Extract a process and report its results

p <- getProcess(exp1, "analyze")
print(p)
```

## Quantiles *The* Quantiles *function*

### Description

Calculates the quantiles of a univariate distribution.

### Usage

```
Quantiles(pvalues, distribName, distribParameters, shrink=0.95)
```

### Arguments

| | |
|---|---|
| pvalues | a vector of probability values. |
| distribName | a string giving the name of a probability distribution. |
| distribParameters | a list of parameters of the distribution. |
| shrink | a scalar eqn<=1 to determine how to shrink the pvalues(used when the quantiles are infinite for pvalues equal to 0 or 1). |

### Value

the q-values

### Author(s)

Hervé Monod, MIA-Jouy, Inra, Domaine de Vilvert, 78352 Jouy en Josas, France

### Examples

```
Quantiles(seq(0,1,length=11),"unif",list(min=8,max=10))
Quantiles(seq(0,1,length=11),"unif",list(min=8,max=10),shrink=0.5)
Quantiles(seq(0,1,length=11),"norm",list(mean=0, sd=1),shrink=0.5)
```

## RandLHS *The* RandLHS *Method*

### Description

A mtk compliant implementation of the method for drawing Random Latin Hypercube Design.

### Usage

- mtkRandLHSDesigner(listParameters = NULL)
- mtkNativeDesigner(design="RandLHS", information=NULL)

### Parameters used to manage the method

`size`: The number of partitions (simulations or design points).

`preserveDraw`: logical (default FALSE). Ensures that two subsequent draws with the same n, but one with k and one with m variables (k<m), will have the same first k columns if the seed is the same.

### Details

1. The `mtk` implementation uses the `randomLHS` function of the package `lhs`. For further details on the arguments and the behavior, see `help(randomLHS, lhs)`.
2. The implementation of the `RandLHS` method includes the class `mtkRandLHSDesigner` to manage the sampling task and the class `mtkRandLHSDesignerResult` to manage the results produced by the sampling process.

### References

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. Technometrics. 29, 143–151.

### See Also

`help(randomLHS, lhs)`

### Examples

```
# uses the RandLHS method
## Random Latin Hypercude draws for the "Ishigami" model

#  Example I: by using the class constructors: mtkRandLHSDesigner()

# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkRandLHSDesigner( listParameters = list(size=10) )

#   2) the workflow

   exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer) )

#  Run the workflow and reports the results.
    run(exp1)
    print(exp1)
    plot(exp1)
```

reevaluate-methods *The* reevaluate *method*

## Description

Re-evaluates the processes of the workflow to know if they should be re-run. This must be done after changing a process in the workflow. The argument "name" gives the process from which the workflow should be reevaluated. i.e. if name="design", we tell the workflow that all the processes after the process "design" should be reevaluated. If name="evaluate", we tell the workflow that only the processes after the process "evaluate" should be re-evaluated, etc.

## Usage

```
reevaluate(this, name)
```

## Arguments

| | |
|---|---|
| this | the underlying object of class mtkExpWorkflow. |
| name | a string from "design", "evaluate", or "analyze" to specify the process from which we re-evaluate the workflow. |

## Value

invisble()

## Details

This function is only useful for the kernel programming.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# see examples.
```

`Regression` *The* `Regression` *Method*

### Description

A `mtk` compliant implementation of the `src` method for computing the sensitivity index based on standardized (rank) regression coefficients.

### Usage

- mtkRegressionAnalyser(listParameters = NULL)
- mtkNativeAnalyser(analyze="Regression", information=NULL)

### Parameters used to manage the method

`rank`**:** logical. If TRUE, the analysis is done on the ranks (default is `FALSE`). See the help on function `src` in the package `sensitivity`.

`nboot`**:** the number of bootstrap replicates (default 100). See the help on function `src` in the package `sensitivity`.

`conf`**:** the confidence level for bootstrap confidence intervals (default 0.95). See the help on function `src` in the package `sensitivity`.

### Details

1. The `mtk` implementation uses the `src` function of the package `sensitivity`. For further details on the arguments and the behavior, see `help(src, sensitivity)`.
2. The implementation of the "Regression" method includes the class `mtkRegressionAnalyser` to manage the analysis task and the class `mtkRegressionAnalyserResult` to manage the results produced by the analysis process.

### References

A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis, Edition Wiley

### See Also

`help(src, sensitivity)`

### Examples

```
# Uses the method "Regression" to analyze the model "Ishigami":

#  Generate the factors
    data(Ishigami.factors)

# Builds experiment design with the Monte-Carlo method
  designer <- mtkBasicMonteCarloDesigner( listParameters=list(size=20) )

# Builds a simulator for the model "Ishigami" with the defined factors
  model <- mtkNativeEvaluator("Ishigami" )

# Builds an analyser with the method "Regression" implemented in the package "mtk"
```

```
  analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

# Builds a workflow to manage the processes scheduling.
  ishiReg <- mtkExpWorkflow( expFactors=Ishigami.factors,
   processesVector=c(design=designer, evaluate=model, analyze=analyser) )

# Runs the workflow et reports the results
  run(ishiReg)
  summary(ishiReg)
    plot(ishiReg)
```

---

`report-methods` *The* `report` *method*

---

## Description

Returns a detail report of the results produced by the process.

## Usage

```
report(this)
```

## Arguments

| | |
|---|---|
| `this` | the underlying object of class `mtkProcess` |

## Value

The form of the value returned by `report` depends on the sub-class where the method is implemented.

See the documentation of the particular sub-class for details of what is produced.

By default, it prints the report on the display device and return `invisible()`.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
```

```
x2 <- make.mtkFactor(name="x2", distribName="unif",
     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
     distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
#   1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
      information=list(r=20,type="oat",levels=4,grid.jump=2))

#   2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   #   3) the analysis process with the default method.
#      Here, it is the Morris method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the  workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
     processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))
# Run the workflow and plot the results.
run(exp1)
report(exp1)

# Extract a process and report its results

p <- getProcess(exp1, "analyze")
report(p)
```

---

`run-methods` *The* `run` *method*

---

## Description

Runs a task defined in a process or workflow. Examples classes in which this function is implemented are the following: [mtkParsor], [mtkExpWorkflow], [mtkProcess] and their sub-classes . Examples of "run" are:

- `run(this, context)` "this" is an object of class [mtkNativeDesigner], and "context" is an object of class [mtkExpWorkflow].
- `run(this, context)` "this" is an object of class [mtkParsor], and "context" is an object of class [mtkExpWorkflow].

## Usage

```
run(this,context)
```

## Arguments

| | |
|---|---|
| `this` | an object corresponding to the task to launch. It may be an object of the following classes: [mtkParsor], [mtkExpWorkflow], [mtkProcess] or their sub-classes. |

| | |
|---|---|
| context | missing or an object specifying the context which manages the task. It may be an object of the following classes: [mtkExpWorkflow] or its sub-classes. |

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
     distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
#   1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
      information=list(r=20,type="oat",levels=4,grid.jump=2))

#   2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   #   3) the analysis process with the default method.
#      Here, it is the Morris method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the  workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
     processesVector = c(design=exp1.designer,
evaluate=exp1.evaluator, analyze=exp1.analyser))
# Run the workflow and plot the results.
run(exp1)
print(exp1)
```

---

# serializeOn-methods

*The* serializeOn *method*

---

## Description

Returns all data and informations managed by an object as a named list.

### Usage

```
serializeOn(this)
```

### Arguments

| | |
|---|---|
| this | the underlying object |

### Value

a named list

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### Examples

```
# Function not used yet in the current release.
```

---

## setDistributionParameters-methods

*The* setDistributionParameters *method*

---

### Description

Sets the parameters of the distribution associated with a factor's domain.

### Usage

```
setDistributionParameters(this, aDistParamList)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class mtkDomain. |
| aDistParamList | a list of objects of class mtkParameter or a named list from which we can build a list of objects of class mtkParameter . |

### Value

invisible()

### Author(s)

Juhui WANG, MIA-jouy, INRA

## Examples

```
# 1)  Build an object of the "mtkDomain" class
d <- mtkDomain(distributionName="unif", domainNominalValue=0)

## Define the parameters
p <- make.mtkParameterList(list(min=-pi, max=pi))

## Assign the parameters to the mtkDomain's object

setDistributionParameters(d, p)
# 2) Build an object of the "mtkDomain" class
d <- mtkDomain(distributionName="unif", domainNominalValue=0)

## Assign the parameters to the mtkDomain's object

setDistributionParameters(d, list(min=-pi, max=pi))

# 3) Build an object of the "mtkDomain" class with a discrete distribution
d <- mtkDomain(distributionName="discrete", domainNominalValue=0)

## Assign the parameters to the mtkDomain's object

setDistributionParameters(d, list(type='categorical', levels=seq(1:3), weights=rep(0.33,3
```

---

# setDomain-methods *The* setDomain *method*

---

## Description

Associates a new domain with the factor.

## Usage

```
setDomain(this, domain)
```

## Arguments

| | |
|---|---|
| this | an object of the class mtkFactor . |
| domain | an object of the class mtkDomain . |

## Value

invisible()

## Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1")

# Define a domain
d <- mtkDomain(distributionName="unif",
 domainNominalValue=0, distributionParameters = list(max=3, min=0))

# Use the setDomain to change the domain of the factor
setDomain(x1,d)
```

---

## setFactors-methods *The* setFactors *method*

---

### Description

Assigns a list of objects of the class mtkFactor to the underlying obejct.

### Usage

```
setFactors(this, aFactList)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class mtkExpFactors. |
| aFactList | a list of objects of the class mtkFactor. |

### Value

invisible()

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors()

# Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
       distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
       distribPara=list(min=-pi, max=pi))
# Assign the factors to the mtkExpFactors' object

setFactors(ishi.factors, list(x1,x2,x3))
```

## setFeatures-methods *The* setFeatures *method*

### Description

Sets the features to an object of the mtkFactor class.

### Usage

```
setFeatures(this, aFList)
```

### Arguments

| | |
|---|---|
| this | an object of the class mtkFactor |
| aFList | a list of mtkFeature objects. |

### Value

invisible

### Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

### Examples

```
# Build an object of the "mtkFactor" class
x1 <- make.mtkFactor(name="x1", type="double", nominal=0, distribName="unif",
 distribPara=list(min=-pi, max=pi))
# Define the list of features
f <- make.mtkFeatureList(list(f=4.5,c=+6,shape="parabolic"))

# Assign the features to the factor

setFeatures(x1,f)
```

## setLevels-methods *The* setLevels *method*

### Description

Sets new levels to a discrete distribution.

### Usage

```
setLevels(this, levels)
```

### Arguments

| | |
|---|---|
| this | an object of the class mtkDomain or mtkLevels. |
| levels | an object of the class mtkLevels or a list from which we can create an object of the class mtkLevels. |

### Value

invisible

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a  new mtkLevels for a discrete distribution

l <- mtkLevels(type='categorical', levels = c(1,2,3,4,5), weights=rep(0.2, 5))
# Change the levels'name to ('a','b','c','d','e')
setLevels(l, c('a','b','c','d','e'))

# Create a  new domain with a discrete distribution
d <- mtkDomain(distributionName="discrete", domainNominalValue=3,
distributionParameters = list(type='categorical',
  levels = c(1,2,3,4,5), weights=rep(0.2, 5)))

# Create a  new mtkLevels for a discrete distribution and assign it to the domain

l <- mtkLevels(type='categorical', levels = c('a','b','c','d','e'), weights=rep(0.2, 5))
setLevels(d, l)

# Change the domain's levels to type='categorical', levels = c(5,4,3,2,1), weights=rep(0.

setLevels(d, levels=list(type='categorical', levels = c(5,4,3,2,1), weights=rep(0.2, 5)))
```

---

## setName-methods *The* setName *method*

---

### Description

Gives a new name to the underlying object

### Usage

```
setName(this, name)
```

### Arguments

| | |
|---|---|
| this | the underlying object |
| name | a string indicating the new name. |

### Value

invisble()

### Details

Used by many classes. The behavior depends on the underlying class.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1", type="double", distribName="unif",
 distribPara=list(min=-pi, max=pi))

# Change the numeric value of the factor to "numeric" type.

setName(x1, name="mit")

# Create a  new object of mtkValue
d <- mtkValue("a", "double", 0)

# Change the name of the object to "x" type.
setName(d, "x")
```

---

## setParameters-methods

*The* `setParameters` *method*

---

### Description

Assigns a vector of parameters to the process.

### Usage

```
setParameters(this,f)
```

### Arguments

| | |
|---|---|
| `this` | the underlying object of class `mtkProcess` |
| `f` | a vector of `mtkParameter`. |

### Value

invisble()

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Create a process for experiments design

designer <- mtkNativeDesigner(design ="Morris")

# Create a list of mtkParameter for the parameters: min, max, shape.
p <- make.mtkParameterList(list(size=20))

# Assign the parameters to the process

setParameters(designer, p)
```

---

# setProcess-methods *The* setProcess *method*

---

## Description

Places or replaces a process into the workflow.

## Usage

```
setProcess(this, p, name)
```

## Arguments

| | |
|---|---|
| this | the underlying object of the class mtkExpWorkflow. |
| p | an object of the class mtkProcess. |
| name | a string from "design", "evaluate", or "analyze" to specify the process to place or replace. |

## Value

invisble()

## Details

This method is especially useful when we need to compare different methods or models.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Analyze the "Ishigami" model with the "Regression" method

x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))


designer <- mtkNativeDesigner("BasicMonteCarlo",
             information=list(size=20))
model <- mtkNativeEvaluator("Ishigami" )
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

ishiReg <- mtkExpWorkflow( expFactors=ishi.factors,
    processesVector=c( design=designer,
      evaluate=model,
      analyze=analyser)
   )
run(ishiReg)
summary(ishiReg)

# Re-analyzes the model "Ishigami" with the method "Morris"

# 1) Build a designer with the method "Morris" and put it into the workflow
morris.designer <- mtkNativeDesigner(
design="Morris",
information=list(r=20,type="oat",levels=4,grid.jump=2)
)
setProcess(ishiReg, morris.designer, "design")

# 2) Build an analysis process with the default method and put it
#     into the workflow
default.analyser <- mtkDefaultAnalyser()
setProcess(ishiReg, default.analyser, "analyze")
# 3) Run the new workflow

run(ishiReg)
summary(ishiReg)
```

---

`setReady-methods` *The* `setReady` *method*

---

### Description

Makes the process ready to run.

### Usage

```
setReady(this, switch)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class mtkProcess |
| switch | a logical (TRUE or FALSE). |

### Value

invisble()

### Details

This function is only useful for the programmers who need to program the mtk's internal functions.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# This function is only useful for the programmers
#  who need to program the mtk's internal functions.
```

---

## setState-methods *The* setState *method*

---

### Description

Marks the state of the process as TRUE when the results produced by the process are available.

### Usage

```
setState(this, state)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the mtkProcess class |
| state | a logical (TRUE or FALSE). |

### Value

invisble()

### Details

This function is only useful for the programmers who need to program the mtk's internal functions.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package `mtk`, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# This function is only useful for the programmers
#  who need to program the mtk's internal functions.
```

---

## `setType-methods` *The* `setType` *method*

---

### Description

Gives a new type to the underlying object.

### Usage

```
setType(this, type)
```

### Arguments

| | |
|---|---|
| `this` | the underlying object |
| `type` | a string indicating the new type for the data. It may be "numeric", "integer", "double", etc. |

### Value

invisble()

### Details

Used by many classes. The behavior depends on the underlying class.

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1", type="double", distribName="unif",
 distribPara=list(min=-pi, max=pi))

# Change the numeric value of the factor to "numeric" type.

setType(x1, type="numeric")

# Create a  new object of mtkValue
d <- mtkValue("a", "double", 0)

# Change the numeric value of the object to "numeric" type.
setType(d, "numeric")
```

---

## setValue-methods *The* setValue *method*

---

### Description

Gives a new value to the underlying object

### Usage

```
setValue(this, val)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the corresponding class. |
| val | a new value. |

### Value

invisble()

### Details

Used by many classes. The behavior depends on the underlying class.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Create a  new object of mtkValue
d <- mtkValue("a", "double", 0)
getValue(d) # gives 0.0

setValue(d, 3.14)
getValue(d) # gives 3.14
```

---

## setWeights-methods *The* setWeights *method*

---

### Description

Gives new weights to the discrete distribution associated with the factor's domain.

### Usage

```
setWeights(this, weights)
```

### Arguments

| | |
|---|---|
| this | the underlying object of the class to proceed (mtkLevels). |
| weights | a vector of numeric value. |

### Value

invisible

### Author(s)

Juhui WANG, MIA-jouy, INRA

### Examples

```
# Create a mtkLevels object

l <- mtkLevels(type='categorical', levels=c(1,2,3,4))

setWeights(l, weights=rep(0.25,4))
```

setXMLFilePath-methods
*The* setXMLFilePath *function*

## Description

Specifies the XML file to parse.

## Usage

```
setXMLFilePath(this, xmlPath)
```

## Arguments

| | |
|---|---|
| this | the underlying object of class mtkParsor |
| xmlPath | a string indicating the XML file to parse. |

## Value

invisble()

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

## Examples

```
# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"

## Find where the example XML file is held in the 'mtk' package.
## (This line is nit useful for real life example!)
xmlFile <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",xmlFile,sep = "")

# Create a XML parsor.
parsor <- mtkParsor(xmlFile)

# Create an empty workflow.
workflow <- mtkExpWorkflow()

# Parse the XML file and initialize the workflow
# with the data extracted from the XML file.
run(parsor, workflow)

# Run the workflow
```

```
run(workflow)

# If you want to parse another XML file with the same parsor,
# just changes the XML file to "inst/extdata/ishigami_fast.xml".

xmlFile <- "ishigami_fast.xml"

# Find where the example XML file is held in the 'mtk' package.
# (This line is nit useful for real life example!)
xmlFile <- paste(path.package("mtk", quiet = TRUE),
"/extdata/",xmlFile,sep = "")

# Change the XML file to the new one
setXMLFilePath(parsor, xmlFile)

# Parse the new XML file and initialize the workflow
# with the data extracted from the XML file.
run(parsor, workflow)

# Run the workflow
run(workflow)
```

---

`Sobol` *The* `Sobol` *Method*

---

## Description

A `mtk` compliant implementation of the `Sobol`' method for design of experiments and sensitivity analysis.

## Usage

- mtkSobolDesigner(listParameters = NULL)
- mtkNativeDesigner(design="Sobol", information=NULL)
- mtkSobolAnalyser(listParameters = NULL)
- mtkNativeAnalyser(analyze="Sobol", information=NULL)

## Parameters

`N`**:** the size of the basic samples; the final sample size will be N*(k+2) where k is the number of the factors to analyze.

`nboot`**:** the number of bootstrap replicates (default 0). See the help on function `sobol2002` in the package `sensitivity`.

`conf`**:** the confidence level for bootstrap confidence intervals (default 0.95). See the help on function `sobol2002` in the package `sensitivity`.

`sampling`**:** character string specifying the type of sampling method: "MC" (default) for Monte Carlo sampling, "LHS" for Latin Hypercube sampling.

`shrink`**:** a scalar or a vector of scalars between 0 and 1 (default 1), specifying shrinkage to be used on the probabilities before calculating the quantiles.

## Details

1. The `mtk` implementation uses the `sobol2002` function of the `sensitivity` package. For further details on the arguments and the behavior, see `help(sobol2002, sensitivity)`.
2. The `mtk` implementation of the `Sobol'` method includes the following classes:
   - `mtkSobolDesigner`: for the `Sobol` design processes.
   - `mtkSobolAnalyser`: for `Sobol` analysis processes.
   - `mtkSobolDesignerResult`: to store and manage the design.
   - `mtkSobolAnalyserResult`: to store and manage the analysis results.
3. Many ways to create a `Sobol` designer are available in `mtk`, but we recommend the following class constructors: `mtkSobolDesigner` or `mtkNativeDesigner`.
4. Many ways to create a `Sobol` analyser are available in `mtk`, but we recommend the following class constructors: `mtkSobolAnalyser` or `mtkNativeAnalyser`.
5. The `Sobol'` method is usually used both to build the experiment design and to carry out the sensitivity analysis. In such case, we can use the `mtkDefaultAnalyser` instead of naming explicitly the method for sensitivity analysis (see example III in the examples section)

## References

A. Saltelli, K. Chan and E. M. Scott (2000). Sensitivity Analysis. Wiley, New York

## See Also

`help(sobol2002, sensitivity)`, `Quantiles`

## Examples

```
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method

# Example I: by using the class constructors: mtkSobolDesigner() and mtkSobolAnalyser()

# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkSobolDesigner( listParameters = list(N=100))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process
   exp1.analyser <- mtkSobolAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
     evaluate=exp1.evaluator,
     analyze=exp1.analyser))

#  Run the workflow and reports the results.
```

```
run(exp1)
print(exp1)
plot(exp1)

##  Example II:  by using the class constructors: mtkNativeDesigner() and mtkSobolAnalyse

# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkNativeDesigner(design = "Sobol", information = list(N=10))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method
exp1.analyser <- mtkSobolAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
     evaluate=exp1.evaluator,
     analyze=exp1.analyser))

#  Run the workflow and reports the results.
run(exp1)
print(exp1)
    plot(exp1)


##  Example III:  by using the class constructors: mtkSobolDesigner() and mtkDefaultAnaly

# Generate the factors
data(Ishigami.factors)

#  Build the processes and workflow:

#   1) the design process
exp1.designer <- mtkSobolDesigner( listParameters = list(N=10))

#   2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

#   3) the analysis process with the default method
exp1.analyser <- mtkDefaultAnalyser()

#   4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=exp1.designer,
     evaluate=exp1.evaluator,
     analyze=exp1.analyser))

#  Run the workflow and reports the results.
```

```
run(exp1)
print(exp1)
plot(exp1)
```

---

summary,mtkProcess-method
*The* summary *method*

---

## Description

Returns a summary report of the results produced by the process.

## Usage

```
summary(object, ...)
```

## Arguments

| | |
|---|---|
| object | the underlying object of class mtkProcess. |
| ... | see the help for the function: base::summary(). |

## Value

The form of the value returned by summary depends on the sub-class where the method is implemented.

By default, it prints the report on the display device.

## Details

1. The behavior of the print depends on the sub-class where the method is implemented.
2. See the documentation of the particular sub-class for details of what is produced.
3. Use methods("summary") to get all the methods for the summary generic.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## References

J. Wang, H. Richard, R. Faivre, H. Monod (2013). Le package mtk, une bibliothèque R pour l'exploration numérique des modèles. *In:* Analyse de sensibilité et exploration de modèles : Application aux sciences de la nature et de l'environnement (R. Faivre, B. Iooss, S. Mahévas, D. Makowski, H. Monod, Eds). Editions Quae, Versailles.

### Examples

```
# Carry out a sensitivity analysis with the Ishigami model

## Input the factors
data(Ishigami.factors)

## Specify the experiments designer
designer <- mtkNativeDesigner ("BasicMonteCarlo",
information=list(size=20))

## Specify the model simulator
model <- mtkIshigamiEvaluator()

## Specify the sensiticity analyser
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

## Specify the workflow
ishiReg <- new("mtkExpWorkflow", expFactors=Ishigami.factors,
   processesVector=c(
               design=designer,
               evaluate=model,
               analyze=analyser)
   )
## Run and report a summary of the results produced by the workflow
run(ishiReg)
summary(ishiReg)
```

---

`WWDM` *The* `WWDM` *model*

---

### Description

The `WWDM` (Winter Wheat Dry Matter Model) is a very simple dynamic crop model with a daily time step. It has been developed at INRA (France) by David Makowski, Marie-Hélène Jeuffroy and Martine Guérif.

The behavior of the model is influenced by seven factors:

`Eb:` Radiation use efficiency

`Eimax:` Maximal ratio of intercepted to incident radiation

`K:` Coefficient of extinction

`Lmax:` Maximal value of the Leaf Area Index (LAI)

`A:` Coefficient of LAI increase

`B:` Coefficient of LAI decrease

`TI:` Temperature threshold

### Details

1. The implementation of the WWDM model includes the object WWDM.factors on the input factors, the class mtkWWDMEvaluator to run the simulations, and the data frame wwdm.climates containing the climate data.
2. In mtk, there are a few ways to build an evaluator of the WWDM model, but we usually recommend the following class constructors: mtkWWDMEvaluator , mtkNativeEvaluator.

### Usage

- mtkWWDMEvaluator(listParameters=NULL)
- mtkNativeEvaluator(model="WWDM",information=NULL)
- mtkEvaluator(protocol = "R", site = "mtk", service = "WWDM", parametersList=NULL)

### Parameters used to manage the simulation

year Either NULL or a number between 1 and 14 to specify the number of years to simulate. A database with 14 yearly sequences of meteorological data are included in the environment (data frame wwdm.climates).

### References

1. Makowski, D., Jeuffroy, M.-H., Guérif, M., 2004. Bayseian methods for updating crop model predictions, applications for predicting biomass and grain protein content. In: Bayseian Statistics and Quality Modelling in the Agro-Food Production Chain (van Boeakel et al. eds), pp. 57-68. Kluwer, Dordrecht.
2. Monod, H., Naud, C., Makowski, D., 2006. Uncertainty and sensitivity analysis for crop models. In: Working with Dynamic Crop Models (Wallach D., Makowski D. and Jones J. eds), pp. 55-100. Elsevier, Amsterdam.

### See Also

```
help(WWDM.factors)
```

### Examples

```
## Evaluation of the "WWDM" model

# Example I: by using the class constructors: mtkWWDMEvaluator()

# Generate the factors
data(WWDM.factors)

# Build the workflow:
#   1) specify the design process
designer <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=50)  )

#   2) specify the evaluation process;
model <- mtkWWDMEvaluator(listParameters = list(year=3) )

#   3) specify the workflow with the processes defined previously

exp <- mtkExpWorkflow( expFactors=WWDM.factors,
    processesVector=c( design=designer, evaluate=model) )
```

```
# Run the workflow and report the results.
  run(exp)
  summary(exp)

# Personnalize the data reporting

  designData <-  extractData(exp,name="design")

  simulationData <- extractData(exp,name="evaluate")

  plot(designData$Eb, simulationData$Biomass, xlab="Eb",ylab="Biomass")

## Example II: by using the class constructor: mtkNativeEvaluator()

# Generate the input factors
data(WWDM.factors)

# Build the workflow:
#   1) specify the design process
designer <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=20)  )

#   2) specify the evaluation process;
model <- mtkNativeEvaluator(model="WWDM", information=list(year=3)  )

#   3) specify the workflow with the processes defined previously

exp <- mtkExpWorkflow(expFactors=WWDM.factors,
    processesVector=c( design=designer, evaluate=model) )

# Run the workflow and report the results.
  run(exp)
  summary(exp)
  plot(exp)
```

---

`wwdm.climates` *Dataset used with the* `WWDM` *model*

---

## Description

This dataset gives climatic data needed by the `WWDM` crop model model.

`ANNEE numeric`, year of weather data: from 1 to 14.

`RG` Global Radiation

`Tmin` Minimal temperature

`Tmax` Maximal temperature

## References

1. Makowski, D., Jeuffroy, M.-H., Gu\'erif, M., 2004. Bayseian methods for updating crop model predictions, applications for predicting biomass and grain protein content. In: Bayseian Statistics and Quality Modelling in the Agro-Food Production Chain (van Boeakel et al. eds), pp. 57-68. Kluwer, Dordrecht.

2. Monod, H., Naud, C., Makowski, D., 2006. Uncertainty and sensitivity analysis for crop models. In: Working with Dynamic Crop Models (Wallach D., Makowski D. and Jones J. eds), pp. 55-100. Elsevier, Amsterdam.

### See Also

```
help(WWDM)
```

### Examples

```
data(wwdm.climates)
summary(wwdm.climates)
wwdm.climates[1:20,]
par(mfrow=c(3,1)) ;
 for(i in 1:3) ts.plot(wwdm.climates[ wwdm.climates[,1]==1,1+i],
 ylab=names(wwdm.climates[1+i])
 )
```

---

## WWDM.factors *The input factors of the WWDM model*

---

### Description

This dataset gives the input factors and their uncertainty domains involved in the WWDM model.

`Eb` Radiation use efficiency

`Eimax` Maximal ratio of intercepted to incident radiation

`K` Coefficient of extinction

`Lmax` Maximal value of the Leaf Area Index (LAI)

`A` Coefficient of LAI increase

`B` Coefficient of LAI decrease

`TI` Temperature threshold

### Usage

```
data(WWDM.factors)
```

### Format

an object of the calss `mtkExpFactors`.

### References

1. Makowski, D., Jeuffroy, M.-H., Gu\'erif, M., 2004. Bayseian methods for updating crop model predictions, applications for predicting biomass and grain protein content. In: Bayseian Statistics and Quality Modelling in the Agro-Food Production Chain (van Boeakel et al. eds), pp. 57-68. Kluwer, Dordrecht.
2. Monod, H., Naud, C., Makowski, D., 2006. Uncertainty and sensitivity analysis for crop models. In: Working with Dynamic Crop Models (Wallach D., Makowski D. and Jones J. eds), pp. 55-100. Elsevier, Amsterdam.

## See Also

```
help(WWDM)
```

## Examples

```
# The code used to generate the WWDM.factors is as follows:
Eb <- make.mtkFactor(name="Eb", distribName="unif",
nominal=1.85, distribPara=list(min=0.9, max=2.8), unit="g/MJ")
Eimax <- make.mtkFactor(name="Eimax", distribName="unif",
nominal=0.94, distribPara=list(min=0.9, max=0.99))
K <- make.mtkFactor(name="K", distribName="unif",
nominal=0.7, distribPara=list(min=0.6, max=0.8))
Lmax <- make.mtkFactor(name="Lmax", distribName="unif",
nominal=7.5, distribPara=list(min=3, max=12), unit="m\u00b2/m\u00b2")
A <- make.mtkFactor(name="A", distribName="unif",
nominal=0.0065, distribPara=list(min=0.0035, max=0.01))
B <- make.mtkFactor(name="B", distribName="unif",
nominal=0.00205, distribPara=list(min=0.0011, max=0.0025))
TI <- make.mtkFactor(name="TI", distribName="unif",
nominal=900, distribPara=list(min=700, max=1100),unit="\u00b0C")

WWDM.factors <- mtkExpFactors(list(Eb,Eimax,K,Lmax,A,B,TI))

# To import the WWDM.factors, just use the following line
data(WWDM.factors)
```

# Index


∗Topic **datasets**
Ishigami.factors, 42
∗Topic **dataset**
wwdm.climates, 201
WWDM.factors, 202
[,mtkExpFactors-method
*(mtkExpFactors-class)*, 82
[[,mtkExpFactors-method
*(mtkExpFactors-class)*, 82
$,mtkExpFactors-method
*(mtkExpFactors-class)*, 82

addProcess, *80*, *86*
addProcess *(addProcess-methods)*, 7
addProcess,mtkExperiment,mtkProcess,character-method
*(mtkExperiment-class)*, 80
addProcess,mtkExpWorkflow,mtkProcess,character-method
*(mtkExpWorkflow-class)*, 85
addProcess-methods, 7
ANY, 8, *56*, *60*, *65*, *68*, *75*, *92*, *97*, *100, 101*, *103*, *109*, *114*, *118, 119*, *122*, *126*, *128, 129*, *133*, *138*, *140*, *145*, *151*, *155*, *159*, *163*, *165*

BasicMonteCarlo, 9

character, *56*, *60*, *65*, *67, 68*, *72*, *75*, *78, 79*, *84*, *92*, *97*, *100, 101*, *103*, *106*, *108*, *114*, *118*, *122*, *126*, *128, 129*, *131*, *133*, *138*, *140*, *145*, *151*, *155*, *159*, *163*, *165*

data.frame, *58*, *63*, *71*, *78*, *94*, *99*, *111*, *116*, *135*, *142*, *147*, *153*, *157*, *161*, *168*
deleteProcess, *80*, *86*
deleteProcess
*(deleteProcess-methods)*, 10
deleteProcess,mtkExperiment,character-method
*(mtkExperiment-class)*, 80
deleteProcess,mtkExpWorkflow,character-method
*(mtkExpWorkflow-class)*, 85
deleteProcess-methods, 10

extractData, *80*, *86*
extractData
*(extractData-methods)*, 12
extractData,mtkExperiment,character-method
*(mtkExperiment-class)*, 80
extractData,mtkExpWorkflow,character-method
*(mtkExpWorkflow-class)*, 85
extractData-methods, 12

Fast, 13

getData, *56*, *76*, *92*, *97*, *104*, *109*, *114*, *119*, *122*, *126*, *134*, *139*, *141*, *145*, *152*, *156*, *160*, *166*
getData *(getData-methods)*, 15
getData,mtkAnalyser-method
*(mtkAnalyser-class)*, 55
getData,mtkBasicMonteCarloDesigner-method
*(mtkBasicMonteCarloDesigner-class)*, 60
getData,mtkDefaultAnalyser-method
*(mtkDefaultAnalyser-class)*, 65
getData,mtkDesigner-method
*(mtkDesigner-class)*, 68
getData,mtkEvaluator-method
*(mtkEvaluator-class)*, 75
getData,mtkFastAnalyser-method
*(mtkFastAnalyser-class)*, 91
getData,mtkFastDesigner-method
*(mtkFastDesigner-class)*, 96
getData,mtkIshigamiEvaluator-method
*(mtkIshigamiEvaluator-class)*, 103
getData,mtkMorrisAnalyser-method
*(mtkMorrisAnalyser-class)*, 108
getData,mtkMorrisDesigner-method
*(mtkMorrisDesigner-class)*, 113
getData,mtkNativeAnalyser-method
*(mtkNativeAnalyser-class)*, 118
getData,mtkNativeDesigner-method
*(mtkNativeDesigner-class)*,

121
getData,mtkNativeEvaluator-method (*mtkNativeEvaluator-class*), 125
getData,mtkPLMMAnalyser-method (*mtkPLMMAnalyser-class*), 133
getData,mtkProcess-method (*mtkProcess-class*), 138
getData,mtkRandLHSDesigner-method (*mtkRandLHSDesigner-class*), 140
getData,mtkRegressionAnalyser-method (*mtkRegressionAnalyser-class*), 144
getData,mtkSobolAnalyser-method (*mtkSobolAnalyser-class*), 151
getData,mtkSobolDesigner-method (*mtkSobolDesigner-class*), 155
getData,mtkSystemEvaluator-method (*mtkSystemEvaluator-class*), 159
getData,mtkWWDMEvaluator-method (*mtkWWDMEvaluator-class*), 165
getData-methods, 15
getDiscreteDistributionLevels, *89*
getDiscreteDistributionLevels (*getDiscreteDistributionLevels-methods*), 17
getDiscreteDistributionLevels,mtkFactor-method (*mtkFactor-class*), 89
getDiscreteDistributionLevels-methods, 17
getDiscreteDistributionType, *73*, *89*
getDiscreteDistributionType (*getDiscreteDistributionType-methods*), 17
getDiscreteDistributionType,mtkDomain-method (*mtkDomain-class*), 72
getDiscreteDistributionType,mtkFactor-method (*mtkFactor-class*), 89
getDiscreteDistributionType-methods, 17
getDiscreteDistributionWeights, *89*
getDiscreteDistributionWeights (*getDiscreteDistributionWeights-methods*), 18
getDiscreteDistributionWeights,mtkFactor-method (*mtkFactor-class*), 89
getDiscreteDistributionWeights-methods, 18
getDistributionName, *73*, *89*
getDistributionName (*getDistributionName-methods*), 19
getDistributionName,mtkDomain-method (*mtkDomain-class*), 72
getDistributionName,mtkFactor-method (*mtkFactor-class*), 89
getDistributionName-methods, 19
getDistributionNames (*getDistributionNames-methods*), 20
getDistributionNames,mtkExpFactors-method (*mtkExpFactors-class*), 82
getDistributionNames-methods, 20
getDistributionNominalValue, *89*
getDistributionNominalValue (*getDistributionNominalValue-methods*), 21
getDistributionNominalValue,mtkFactor-method (*mtkFactor-class*), 89
getDistributionNominalValue-methods, 21
getDistributionNominalValues (*getDistributionNominalValues-methods*), 21
getDistributionNominalValues,mtkExpFactors-method (*mtkExpFactors-class*), 82
getDistributionNominalValues-methods, 21
getDistributionNominalValueType, *89*
getDistributionNominalValueType (*getDistributionNominalValueType-methods*), 22
getDistributionNominalValueType,mtkFactor-method (*mtkFactor-class*), 89
getDistributionNominalValueType-methods, 22
getDistributionNominalValueTypes (*getDistributionNominalValueTypes-methods*), 23
getDistributionNominalValueTypes,mtkExpFactors-method (*mtkExpFactors-class*), 82
getDistributionNominalValueTypes-methods, 23
getDistributionParameters, *73*, *90*
getDistributionParameters

(getDistributionParameters-methods), 24
getDistributionParameters,mtkDomain-method (mtkDomain-class), 72
getDistributionParameters,mtkExpFactors-method (mtkExpFactors-class), 82
getDistributionParameters,mtkFactor-method (mtkFactor-class), 89
getDistributionParameters-methods, 24
getDomain, 89
getDomain (getDomain-methods), 24
getDomain,mtkFactor-method (mtkFactor-class), 89
getDomain-methods, 24
getFactorFeatures (getFactorFeatures-methods), 25
getFactorFeatures,mtkExpFactors-method (mtkExpFactors-class), 82
getFactorFeatures-methods, 25
getFactorNames (getFactorNames-methods), 26
getFactorNames,mtkExpFactors-method (mtkExpFactors-class), 82
getFactorNames-methods, 26
getFactors (getFactors-methods), 27
getFactors,mtkExpFactors-method (mtkExpFactors-class), 82
getFactors-methods, 27
getFeatures, 90
getFeatures (getFeatures-methods), 28
getFeatures,mtkFactor-method (mtkFactor-class), 89
getFeatures-methods, 28
getLevels, 73, 106
getLevels (getLevels-methods), 28
getLevels,mtkDomain-method (mtkDomain-class), 72
getLevels,mtkLevels-method (mtkLevels-class), 106
getLevels-methods, 28
getMTKFeatures, 90
getMTKFeatures (getMTKFeatures-methods), 29
getMTKFeatures,mtkFactor-method (mtkFactor-class), 89
getMTKFeatures-methods, 29
getName, 89, 101, 129, 138, 163
getName (getName-methods), 30
getName,mtkFactor-method (mtkFactor-class), 89
getName,mtkFeature-method (mtkFeature-class), 101
getName,mtkParameter-method (mtkParameter-class), 129
getName,mtkProcess-method (mtkProcess-class), 138
getName,mtkValue-method (mtkValue-class), 162
getName-methods, 30
getNames (getNames-methods), 30
getNames,mtkExpFactors-method (mtkExpFactors-class), 82
getNames-methods, 30
getNominalValue, 73
getNominalValue (getNominalValue-methods), 31
getNominalValue,mtkDomain-method (mtkDomain-class), 72
getNominalValue-methods, 31
getNominalValueType, 73
getNominalValueType (getNominalValueType-methods), 32
getNominalValueType,mtkDomain-method (mtkDomain-class), 72
getNominalValueType-methods, 32
getParameters, 56, 75, 92, 97, 103, 109, 114, 119, 122, 126, 133, 138, 141, 145, 151, 155, 159, 166
getParameters (getParameters-methods), 33
getParameters,mtkAnalyser-method (mtkAnalyser-class), 55
getParameters,mtkBasicMonteCarloDesigner-method (mtkBasicMonteCarloDesigner-class), 60
getParameters,mtkDefaultAnalyser-method (mtkDefaultAnalyser-class), 65
getParameters,mtkDesigner-method (mtkDesigner-class), 68
getParameters,mtkEvaluator-method (mtkEvaluator-class), 75
getParameters,mtkFastAnalyser-method (mtkFastAnalyser-class), 91
getParameters,mtkFastDesigner-method (mtkFastDesigner-class), 96

getParameters,mtkIshigamiEvaluator-method (*mtkIshigamiEvaluator-class*), 103
getParameters,mtkMorrisAnalyser-method (*mtkMorrisAnalyser-class*), 108
getParameters,mtkMorrisDesigner-method (*mtkMorrisDesigner-class*), 113
getParameters,mtkNativeAnalyser-method (*mtkNativeAnalyser-class*), 118
getParameters,mtkNativeDesigner-method (*mtkNativeDesigner-class*), 121
getParameters,mtkNativeEvaluator-method (*mtkNativeEvaluator-class*), 125
getParameters,mtkPLMMAnalyser-method (*mtkPLMMAnalyser-class*), 133
getParameters,mtkProcess-method (*mtkProcess-class*), 138
getParameters,mtkRandLHSDesigner-method (*mtkRandLHSDesigner-class*), 140
getParameters,mtkRegressionAnalyser-method (*mtkRegressionAnalyser-class*), 144
getParameters,mtkSobolAnalyser-method (*mtkSobolAnalyser-class*), 151
getParameters,mtkSobolDesigner-method (*mtkSobolDesigner-class*), 155
getParameters,mtkSystemEvaluator-method (*mtkSystemEvaluator-class*), 159
getParameters,mtkWWDMEvaluator-method (*mtkWWDMEvaluator-class*), 165
getParameters-methods, 33
getProcess, *80*, *86*
getProcess (*getProcess-methods*), 34
getProcess,mtkExperiment,character-method (*mtkExperiment-class*), 80
getProcess,mtkExpWorkflow,character-method (*mtkExpWorkflow-class*), 85
getProcess-methods, 34
getResult, *56*, *76*, *92*, *97*, *104*, *109*, *114*, *119*, *122*, *126*, *134*, *138*, *141*, *145*, *152*, *155*, *160*, *166*
getResult (*getResult-methods*), 35
getResult,mtkAnalyser-method (*mtkAnalyser-class*), 55
getResult,mtkBasicMonteCarloDesigner-method (*mtkBasicMonteCarloDesigner-class*), 60
getResult,mtkDefaultAnalyser-method (*mtkDefaultAnalyser-class*), 65
getResult,mtkDesigner-method (*mtkDesigner-class*), 68
getResult,mtkEvaluator-method (*mtkEvaluator-class*), 75
getResult,mtkFastAnalyser-method (*mtkFastAnalyser-class*), 91
getResult,mtkFastDesigner-method (*mtkFastDesigner-class*), 96
getResult,mtkIshigamiEvaluator-method (*mtkIshigamiEvaluator-class*), 103
getResult,mtkMorrisAnalyser-method (*mtkMorrisAnalyser-class*), 108
getResult,mtkMorrisDesigner-method (*mtkMorrisDesigner-class*), 113
getResult,mtkNativeAnalyser-method (*mtkNativeAnalyser-class*), 118
getResult,mtkNativeDesigner-method (*mtkNativeDesigner-class*), 121
getResult,mtkNativeEvaluator-method (*mtkNativeEvaluator-class*), 125
getResult,mtkPLMMAnalyser-method (*mtkPLMMAnalyser-class*), 133
getResult,mtkProcess-method (*mtkProcess-class*), 138
getResult,mtkRandLHSDesigner-method (*mtkRandLHSDesigner-class*), 140
getResult,mtkRegressionAnalyser-method (*mtkRegressionAnalyser-class*), 144
getResult,mtkSobolAnalyser-method (*mtkSobolAnalyser-class*), 151
getResult,mtkSobolDesigner-method (*mtkSobolDesigner-class*),

155
getResult,mtkSystemEvaluator-method (*mtkSystemEvaluator-class*), 159
getResult,mtkWWDMEvaluator-method (*mtkWWDMEvaluator-class*), 165
getResult-methods, 35
getType, *89*, *101*, *106*, *129*, *163*
getType (*getType-methods*), 36
getType,mtkFactor-method (*mtkFactor-class*), 89
getType,mtkFeature-method (*mtkFeature-class*), 101
getType,mtkLevels-method (*mtkLevels-class*), 106
getType,mtkParameter-method (*mtkParameter-class*), 129
getType,mtkValue-method (*mtkValue-class*), 162
getType-methods, 36
getValue, *101*, *129*, *163*
getValue (*getValue-methods*), 37
getValue,mtkFeature-method (*mtkFeature-class*), 101
getValue,mtkParameter-method (*mtkParameter-class*), 129
getValue,mtkValue-method (*mtkValue-class*), 162
getValue-methods, 37
getWeights, *73*, *106*
getWeights (*getWeights-methods*), 38
getWeights,mtkDomain-method (*mtkDomain-class*), 72
getWeights,mtkLevels-method (*mtkLevels-class*), 106
getWeights-methods, 38

initialize, *89*
initialize,mtkDomain-method (*mtkDomain-class*), 72
initialize,mtkExpFactors-method (*mtkExpFactors-class*), 82
initialize,mtkFactor-method (*mtkFactor-class*), 89
is.finished (*is.finished-methods*), 38
is.finished,mtkAnalyser-method (*mtkAnalyser-class*), 55
is.finished,mtkBasicMonteCarloDesigner-method (*mtkBasicMonteCarloDesigner-class*), 60
is.finished,mtkDefaultAnalyser-method (*mtkDefaultAnalyser-class*), 65
is.finished,mtkDesigner-method (*mtkDesigner-class*), 68
is.finished,mtkEvaluator-method (*mtkEvaluator-class*), 75
is.finished,mtkFastAnalyser-method (*mtkFastAnalyser-class*), 91
is.finished,mtkFastDesigner-method (*mtkFastDesigner-class*), 96
is.finished,mtkIshigamiEvaluator-method (*mtkIshigamiEvaluator-class*), 103
is.finished,mtkMorrisAnalyser-method (*mtkMorrisAnalyser-class*), 108
is.finished,mtkMorrisDesigner-method (*mtkMorrisDesigner-class*), 113
is.finished,mtkNativeAnalyser-method (*mtkNativeAnalyser-class*), 118
is.finished,mtkNativeDesigner-method (*mtkNativeDesigner-class*), 121
is.finished,mtkNativeEvaluator-method (*mtkNativeEvaluator-class*), 125
is.finished,mtkPLMMAnalyser-method (*mtkPLMMAnalyser-class*), 133
is.finished,mtkProcess-method (*mtkProcess-class*), 138
is.finished,mtkRandLHSDesigner-method (*mtkRandLHSDesigner-class*), 140
is.finished,mtkRegressionAnalyser-method (*mtkRegressionAnalyser-class*), 144
is.finished,mtkSobolAnalyser-method (*mtkSobolAnalyser-class*), 151
is.finished,mtkSobolDesigner-method (*mtkSobolDesigner-class*), 155
is.finished,mtkSystemEvaluator-method (*mtkSystemEvaluator-class*), 159
is.finished,mtkWWDMEvaluator-method (*mtkWWDMEvaluator-class*), 165

is.finished-methods, 38
is.ready, *56*, *76*, *92*, *97*, *103, 104*, *109*, *114*, *119*, *122*, *126*, *133, 134*, *138*, *141*, *145*, *151, 152*, *155*, *159*, *166*
is.ready (*is.ready-methods*), 39
is.ready,mtkAnalyser-method (*mtkAnalyser-class*), 55
is.ready,mtkBasicMonteCarloDesigner-method (*mtkBasicMonteCarloDesigner-class*), 60
is.ready,mtkDefaultAnalyser-method (*mtkDefaultAnalyser-class*), 65
is.ready,mtkDesigner-method (*mtkDesigner-class*), 68
is.ready,mtkEvaluator-method (*mtkEvaluator-class*), 75
is.ready,mtkFastAnalyser-method (*mtkFastAnalyser-class*), 91
is.ready,mtkFastDesigner-method (*mtkFastDesigner-class*), 96
is.ready,mtkIshigamiEvaluator-method (*mtkIshigamiEvaluator-class*), 103
is.ready,mtkMorrisAnalyser-method (*mtkMorrisAnalyser-class*), 108
is.ready,mtkMorrisDesigner-method (*mtkMorrisDesigner-class*), 113
is.ready,mtkNativeAnalyser-method (*mtkNativeAnalyser-class*), 118
is.ready,mtkNativeDesigner-method (*mtkNativeDesigner-class*), 121
is.ready,mtkNativeEvaluator-method (*mtkNativeEvaluator-class*), 125
is.ready,mtkPLMMAnalyser-method (*mtkPLMMAnalyser-class*), 133
is.ready,mtkProcess-method (*mtkProcess-class*), 138
is.ready,mtkRandLHSDesigner-method (*mtkRandLHSDesigner-class*), 140
is.ready,mtkRegressionAnalyser-method (*mtkRegressionAnalyser-class*), 144
is.ready,mtkSobolAnalyser-method (*mtkSobolAnalyser-class*), 151
is.ready,mtkSobolDesigner-method (*mtkSobolDesigner-class*), 155
is.ready,mtkSystemEvaluator-method (*mtkSystemEvaluator-class*), 159
is.ready,mtkWWDMEvaluator-method (*mtkWWDMEvaluator-class*), 165
is.ready-methods, 39
Ishigami, 40, *42*
Ishigami.factors, *41*, 42

list, *58*, *63*, *71, 72*, *78, 79*, *83*, *149*
logical, *56*, *60*, *65*, *68*, *75*, *92*, *97*, *103*, *108, 109*, *114*, *119*, *122*, *126*, *133*, *138*, *140*, *145*, *151*, *155*, *159*, *165*

make.mtkFactor, 43, *88*
make.mtkFeatureList, 44, *100, 101*
make.mtkParameterList, 45, *128, 129*
Morris, 45
mtk-package, 5
mtk.analyserAddons, 48
mtk.designerAddons, 50
mtk.evaluatorAddons, 52
mtkAnalyser, 54, *55*, *65*, *91, 92*, *108*, *118*, *133*, *138*, *145*, *151*
mtkAnalyser-class, 55
mtkAnalyserResult, *55, 56*, 57, *57*, *66*, *92*, *94*, *111*, *119*, *135*, *147*, *153*
mtkAnalyserResult-class, 58
mtkBasicMonteCarloDesigner, *9*, 59, *59*
mtkBasicMonteCarloDesigner-class, 60
mtkBasicMonteCarloDesignerResult, *9*, *61*, 62, *62, 63*
mtkBasicMonteCarloDesignerResult-class, 63
mtkDefaultAnalyser, *14*, *46*, 64, *64, 65*, *196*
mtkDefaultAnalyser-class, 65
mtkDesigner, *35*, *60*, 67, *67*, *96*, *113*, *121*, *138*, *140*, *155*
mtkDesigner-class, 68
mtkDesignerResult, *35*, *63*, *67, 68*, 70, *70, 71*, *97*, *99*, *116*, *122*, *142*, *149*, *157*
mtkDesignerResult-class, 70
mtkDomain, *19*, *24*, *28*, *31, 32*, *38*, 71, *71–73*, *88, 89*, *182, 183*, *186*

mtkDomain-class, 72
mtkEvaluator, 74, *74*, *103*, *125*, *138*, *159*, *165*
mtkEvaluator-class, 75
mtkEvaluatorResult, *74*, *76*, 77, *77, 78*, *104*, *126*, *149*, *160, 161*, *168*
mtkEvaluatorResult-class, 77
mtkExperiment, 78, *79, 80*
mtkExperiment-class, 80
mtkExpFactors, *20, 21*, *23–25*, *27*, *30*, *43*, *78*, *80*, 82, *82–84*, *86*, *184*, *202*
mtkExpFactors-class, 82
mtkExpWorkflow, *7*, *11, 12*, *34*, *80*, 84, *84*, *86*, *177*, *180, 181*, *188*
mtkExpWorkflow-class, 85
mtkFactor, *17–19*, *21, 22*, *24, 25*, *27–29*, *44*, *82, 83*, 88, *88–90*, *183–185*
mtkFactor-class, 89
mtkFastAnalyser, *13, 14*, 90, *91, 92*
mtkFastAnalyser-class, 91
mtkFastAnalyserResult, *13*, 93, *94*
mtkFastAnalyserResult-class, 94
mtkFastDesigner, *13*, 95, *95, 96*
mtkFastDesigner-class, 96
mtkFastDesignerResult, *13*, 98, *99*
mtkFastDesignerResult-class, 99
mtkFeature, *29*, *44*, *90*, 100, *100, 101*, *162*, *185*
mtkFeature-class, 101
mtkIshigamiEvaluator, *41*, 102, *102, 103*
mtkIshigamiEvaluator-class, 103
mtkLevels, *28*, *38*, *72, 73*, 105, *105, 106*, *186*, *193*
mtkLevels-class, 106
mtkLevesl, *106*
mtkMorrisAnalyser, *46*, *55*, 107, *107, 108*
mtkMorrisAnalyser-class, 108
mtkMorrisAnalyserResult, *46*, *58*, *109*, 110, *110, 111*
mtkMorrisAnalyserResult-class, 111
mtkMorrisDesigner, *46*, *68*, 112, *112, 113*
mtkMorrisDesigner-class, 113
mtkMorrisDesignerResult, *46*, *114*, 115, *116*
mtkMorrisDesignerResult-class, 116
mtkNativeAnalyser, *14*, *46*, *55*, 117, *117–119*, *170*, *196*
mtkNativeAnalyser-class, 118
mtkNativeDesigner, *9*, *13*, *46*, *68*, 120, *120, 121*, *123*, *180*, *196*
mtkNativeDesigner-class, 121
mtkNativeEvaluator, *41*, *75*, 123, *124, 125*, *127*, *200*
mtkNativeEvaluator-class, 125
mtkParameter, *45*, *55, 56*, *59, 60*, *65*, *67, 68*, *72*, *74, 75*, *90*, *92*, *95*, *97*, *103, 107, 108*, *112*, *114*, *119*, *122*, *126*, 128, *128, 129*, *132, 133*, *137, 138*, *140*, *144, 145*, *150, 151*, *154, 155*, *158, 159*, *162*, *164, 165*, *182*, *187*
mtkParameter-class, 129
mtkParsor, 130, *130, 131*, *180*, *194*
mtkParsor-class, 131
mtkPLMMAnalyser, 132, *132, 133*, *170*
mtkPLMMAnalyser-class, 133
mtkPLMMAnalyserResult, *58*, 134, *134, 135*, *170*
mtkPLMMAnalyserResult-class, 135
mtkProcess, *7*, *16*, *33–35*, *38, 39*, *55*, *68*, *75*, *80*, *84*, *86*, 136, *136, 137*, *139*, *172, 173*, *179, 180*, *187, 188*, *190*, *198*
mtkProcess-class, 138
mtkRandLHSDesigner, 139, *140*, *176*
mtkRandLHSDesigner-class, 140
mtkRandLHSDesignerResult, 141, *141, 142*, *176*
mtkRandLHSDesignerResult-class, 142
mtkReadFactors *(mtkReadFactors-methods)*, 143
mtkReadFactors-methods, 143
mtkRegressionAnalyser, 143, *144, 145*, *178*
mtkRegressionAnalyser-class, 144
mtkRegressionAnalyserResult, *145*, 146, *147*, *178*
mtkRegressionAnalyserResult-class, 147
mtkResult, *35*, *58*, *70*, *77*, *137*, 148, *148, 149*
mtkResult-class, 149
mtkSobolAnalyser, 150, *150, 151*, *196*
mtkSobolAnalyser-class, 151
mtkSobolAnalyserResult, 152, *152, 153*, *196*
mtkSobolAnalyserResult-class, 153
mtkSobolDesigner, 154, *154, 155*, *196*

mtkSobolDesigner-class, 155
mtkSobolDesignerResult, *155*, 156, *156, 157*, *196*
mtkSobolDesignerResult-class, 157
mtkSystemEvaluator, 158, *158, 159*
mtkSystemEvaluator-class, 159
mtkSystemEvaluatorResult, 160, *160, 161*
mtkSystemEvaluatorResult-class, 161
mtkValue, *37*, *72*, *101*, *129*, 162, *162*
mtkValue-class, 162
mtkWWDMEvaluator, *75*, 163, *164, 165*, *200*
mtkWWDMEvaluator-class, 165
mtkWWDMEvaluatorResult, *77*, *166*, 167, *167, 168*
mtkWWDMEvaluatorResult-class, 168

NULL, *94*, *99*, *111*, *116*, *135*, *142*, *147*, *153*, *157*, *161*, *168*
numeric, *13*, *106*

PLMM, 169
plmm *(PLMM)*, 169
plot, *56*, *58*, *63*, *71*, *76*, *78*, *81*, *86*, *92*, *95*, *97*, *99*, *104*, *109*, *111*, *114*, *116*, *119*, *122*, *126*, *134*, *136*, *139*, *141, 142*, *146*, *148*, *152, 153*, *156, 157*, *160, 161*, *166*, *168*
plot *(plot,mtkProcess-method)*, 172
plot,mtkAnalyser-method *(mtkAnalyser-class)*, 55
plot,mtkAnalyserResult-method *(mtkAnalyserResult-class)*, 58
plot,mtkBasicMonteCarloDesigner-method *(mtkBasicMonteCarloDesigner-class)*, 60
plot,mtkBasicMonteCarloDesignerResult-method *(mtkBasicMonteCarloDesignerResult-class)*, 63
plot,mtkDefaultAnalyser-method *(mtkDefaultAnalyser-class)*, 65
plot,mtkDesigner-method *(mtkDesigner-class)*, 68
plot,mtkDesignerResult-method *(mtkDesignerResult-class)*, 70
plot,mtkEvaluator-method *(mtkEvaluator-class)*, 75
plot,mtkEvaluatorResult-method *(mtkEvaluatorResult-class)*, 77
plot,mtkExperiment-method *(mtkExperiment-class)*, 80
plot,mtkExpWorkflow-method *(mtkExpWorkflow-class)*, 85
plot,mtkFastAnalyser-method *(mtkFastAnalyser-class)*, 91
plot,mtkFastAnalyserResult-method *(mtkFastAnalyserResult-class)*, 94
plot,mtkFastDesigner-method *(mtkFastDesigner-class)*, 96
plot,mtkFastDesignerResult-method *(mtkFastDesignerResult-class)*, 99
plot,mtkIshigamiEvaluator-method *(mtkIshigamiEvaluator-class)*, 103
plot,mtkMorrisAnalyser-method *(mtkMorrisAnalyser-class)*, 108
plot,mtkMorrisAnalyserResult-method *(mtkMorrisAnalyserResult-class)*, 111
plot,mtkMorrisDesigner-method *(mtkMorrisDesigner-class)*, 113
plot,mtkMorrisDesignerResult-method *(mtkMorrisDesignerResult-class)*, 116
plot,mtkNativeAnalyser-method *(mtkNativeAnalyser-class)*, 118
plot,mtkNativeDesigner-method *(mtkNativeDesigner-class)*, 121
plot,mtkNativeEvaluator-method *(mtkNativeEvaluator-class)*, 125
plot,mtkPLMMAnalyser-method *(mtkPLMMAnalyser-class)*, 133
plot,mtkPLMMAnalyserResult-method *(mtkPLMMAnalyserResult-class)*, 135
plot,mtkProcess-method, 172
plot,mtkRandLHSDesigner-method *(mtkRandLHSDesigner-class)*, 140
plot,mtkRandLHSDesignerResult-method

(mtkRandLHSDesignerResult-class), 142
plot,mtkRegressionAnalyser-method (mtkRegressionAnalyser-class), 144
plot,mtkRegressionAnalyserResult-method (mtkRegressionAnalyserResult-class), 147
plot,mtkSobolAnalyser-method (mtkSobolAnalyser-class), 151
plot,mtkSobolAnalyserResult-method (mtkSobolAnalyserResult-class), 153
plot,mtkSobolDesigner-method (mtkSobolDesigner-class), 155
plot,mtkSobolDesignerResult-method (mtkSobolDesignerResult-class), 157
plot,mtkSystemEvaluator-method (mtkSystemEvaluator-class), 159
plot,mtkSystemEvaluatorResult-method (mtkSystemEvaluatorResult-class), 161
plot,mtkWWDMEvaluator-method (mtkWWDMEvaluator-class), 165
plot,mtkWWDMEvaluatorResult-method (mtkWWDMEvaluatorResult-class), 168
print, *56*, *58*, *63*, *71*, *73*, *76*, *78*, *81*, *83*, *86*, *90*, *92*, *95*, *97*, *99*, *101*, *104*, *106*, *109*, *111*, *114*, *116*, *119*, *122*, *126*, *129*, *134*, *136*, *139*, *141*, *142*, *146*, *148*, *152*, *153*, *156*, *157*, *160*, *161*, *163*, *166*, *168*
print (print,mtkProcess-method), 173
print,mtkAnalyser-method (mtkAnalyser-class), 55
print,mtkAnalyserResult-method (mtkAnalyserResult-class), 58
print,mtkBasicMonteCarloDesigner-method (mtkBasicMonteCarloDesigner-class), 60
print,mtkBasicMonteCarloDesignerResult-method (mtkBasicMonteCarloDesignerResult-class), 63
print,mtkDefaultAnalyser-method (mtkDefaultAnalyser-class), 65
print,mtkDesigner-method (mtkDesigner-class), 68
print,mtkDesignerResult-method (mtkDesignerResult-class), 70
print,mtkDomain-method (mtkDomain-class), 72
print,mtkEvaluator-method (mtkEvaluator-class), 75
print,mtkEvaluatorResult-method (mtkEvaluatorResult-class), 77
print,mtkExperiment-method (mtkExperiment-class), 80
print,mtkExpFactors-method (mtkExpFactors-class), 82
print,mtkExpWorkflow-method (mtkExpWorkflow-class), 85
print,mtkFactor-method (mtkFactor-class), 89
print,mtkFastAnalyser-method (mtkFastAnalyser-class), 91
print,mtkFastAnalyserResult-method (mtkFastAnalyserResult-class), 94
print,mtkFastDesigner-method (mtkFastDesigner-class), 96
print,mtkFastDesignerResult-method (mtkFastDesignerResult-class), 99
print,mtkFeature-method (mtkFeature-class), 101
print,mtkIshigamiEvaluator-method (mtkIshigamiEvaluator-class), 103
print,mtkLevels-method (mtkLevels-class), 106
print,mtkMorrisAnalyser-method (mtkMorrisAnalyser-class), 108
print,mtkMorrisAnalyserResult-method (mtkMorrisAnalyserResult-class), 111
print,mtkMorrisDesigner-method (mtkMorrisDesigner-class), 113
print,mtkMorrisDesignerResult-method (mtkMorrisDesignerResult-class), 116
print,mtkNativeAnalyser-method

(mtkNativeAnalyser-class), 118
print,mtkNativeDesigner-method (mtkNativeDesigner-class), 121
print,mtkNativeEvaluator-method (mtkNativeEvaluator-class), 125
print,mtkParameter-method (mtkParameter-class), 129
print,mtkPLMMAnalyser-method (mtkPLMMAnalyser-class), 133
print,mtkPLMMAnalyserResult-method (mtkPLMMAnalyserResult-class), 135
print,mtkProcess-method, 173
print,mtkRandLHSDesigner-method (mtkRandLHSDesigner-class), 140
print,mtkRandLHSDesignerResult-method (mtkRandLHSDesignerResult-class), 142
print,mtkRegressionAnalyser-method (mtkRegressionAnalyser-class), 144
print,mtkRegressionAnalyserResult-method (mtkRegressionAnalyserResult-class), 147
print,mtkSobolAnalyser-method (mtkSobolAnalyser-class), 151
print,mtkSobolAnalyserResult-method (mtkSobolAnalyserResult-class), 153
print,mtkSobolDesigner-method (mtkSobolDesigner-class), 155
print,mtkSobolDesignerResult-method (mtkSobolDesignerResult-class), 157
print,mtkSystemEvaluator-method (mtkSystemEvaluator-class), 159
print,mtkSystemEvaluatorResult-method (mtkSystemEvaluatorResult-class), 161
print,mtkValue-method (mtkValue-class), 162
print,mtkWWDMEvaluator-method (mtkWWDMEvaluator-class), 165
print,mtkWWDMEvaluatorResult-method (mtkWWDMEvaluatorResult-class), 168

Quantiles, 175

RandLHS, 175
reevaluate, *81*, *86*
reevaluate (reevaluate-methods), 177
reevaluate,mtkExperiment,character-method (mtkExperiment-class), 80
reevaluate,mtkExpWorkflow,character-method (mtkExpWorkflow-class), 85
reevaluate-methods, 177
Regression, 178
report, *56*, *76*, *81*, *86*, *92*, *97*, *104*, *109*, *114*, *119*, *122*, *126*, *134*, *139*, *141*, *146*, *152*, *156*, *160*, *166*
report (report-methods), 179
report,mtkAnalyser-method (mtkAnalyser-class), 55
report,mtkBasicMonteCarloDesigner-method (mtkBasicMonteCarloDesigner-class), 60
report,mtkDefaultAnalyser-method (mtkDefaultAnalyser-class), 65
report,mtkDesigner-method (mtkDesigner-class), 68
report,mtkEvaluator-method (mtkEvaluator-class), 75
report,mtkExperiment-method (mtkExperiment-class), 80
report,mtkExpWorkflow-method (mtkExpWorkflow-class), 85
report,mtkFastAnalyser-method (mtkFastAnalyser-class), 91
report,mtkFastDesigner-method (mtkFastDesigner-class), 96
report,mtkIshigamiEvaluator-method (mtkIshigamiEvaluator-class), 103
report,mtkMorrisAnalyser-method (mtkMorrisAnalyser-class), 108
report,mtkMorrisDesigner-method (mtkMorrisDesigner-class), 113
report,mtkNativeAnalyser-method (mtkNativeAnalyser-class), 118

report,mtkNativeDesigner-method
(*mtkNativeDesigner-class*),
121
report,mtkNativeEvaluator-method
(*mtkNativeEvaluator-class*),
125
report,mtkPLMMAnalyser-method
(*mtkPLMMAnalyser-class*),
133
report,mtkProcess-method
(*mtkProcess-class*), 138
report,mtkRandLHSDesigner-method
(*mtkRandLHSDesigner-class*),
140
report,mtkRegressionAnalyser-method
(*mtkRegressionAnalyser-class*),
144
report,mtkSobolAnalyser-method
(*mtkSobolAnalyser-class*),
151
report,mtkSobolDesigner-method
(*mtkSobolDesigner-class*),
155
report,mtkSystemEvaluator-method
(*mtkSystemEvaluator-class*),
159
report,mtkWWDMEvaluator-method
(*mtkWWDMEvaluator-class*),
165
report-methods, 179
run, *56*, *76*, *81*, *86*, *92*, *97*, *104*, *109*, *114*,
*119*, *122*, *126*, *131*, *134*, *139*, *141*,
*146*, *152*, *156*, *160*, *166*
run (*run-methods*), 180
run,mtkAnalyser,mtkExpWorkflow-method
(*mtkAnalyser-class*), 55
run,mtkBasicMonteCarloDesigner,mtkExpWorkflow-method
(*mtkBasicMonteCarloDesigner-class*),
60
run,mtkDefaultAnalyser,mtkExpWorkflow-method
(*mtkDefaultAnalyser-class*),
65
run,mtkDesigner,mtkExpWorkflow-method
(*mtkDesigner-class*), 68
run,mtkEvaluator,mtkExpWorkflow-method
(*mtkEvaluator-class*), 75
run,mtkExperiment,missing-method
(*mtkExperiment-class*), 80
run,mtkExpWorkflow,missing-method
(*mtkExpWorkflow-class*), 85
run,mtkFastAnalyser,mtkExpWorkflow-method
(*mtkFastAnalyser-class*), 91
run,mtkFastDesigner,mtkExpWorkflow-method
(*mtkFastDesigner-class*), 96
run,mtkIshigamiEvaluator,mtkExpWorkflow-method
(*mtkIshigamiEvaluator-class*),
103
run,mtkMorrisAnalyser,mtkExpWorkflow-method
(*mtkMorrisAnalyser-class*),
108
run,mtkMorrisDesigner,mtkExpWorkflow-method
(*mtkMorrisDesigner-class*),
113
run,mtkNativeAnalyser,mtkExpWorkflow-method
(*mtkNativeAnalyser-class*),
118
run,mtkNativeDesigner,mtkExpWorkflow-method
(*mtkNativeDesigner-class*),
121
run,mtkNativeEvaluator,mtkExpWorkflow-method
(*mtkNativeEvaluator-class*),
125
run,mtkParsor,mtkExpWorkflow-method
(*mtkParsor-class*), 131
run,mtkPLMMAnalyser,mtkExpWorkflow-method
(*mtkPLMMAnalyser-class*),
133
run,mtkProcess,mtkExpWorkflow-method
(*mtkProcess-class*), 138
run,mtkRandLHSDesigner,mtkExpWorkflow-method
(*mtkRandLHSDesigner-class*),
140
run,mtkRegressionAnalyser,mtkExpWorkflow-method
(*mtkRegressionAnalyser-class*),
144
run,mtkSobolAnalyser,mtkExpWorkflow-method
(*mtkSobolAnalyser-class*),
151
run,mtkSobolDesigner,mtkExpWorkflow-method
(*mtkSobolDesigner-class*),
155
run,mtkSystemEvaluator,mtkExpWorkflow-method
(*mtkSystemEvaluator-class*),
159
run,mtkWWDMEvaluator,mtkExpWorkflow-method
(*mtkWWDMEvaluator-class*),
165
run-methods, 180

serializeOn, *56*, *76*, *81*, *86*, *92*, *97*, *104*,
*109*, *114*, *119*, *122*, *126*, *134*, *139*,
*141*, *145*, *149*, *152*, *156*, *160*, *166*
serializeOn
(*serializeOn-methods*), 181

serializeOn,mtkAnalyser-method (*mtkAnalyser-class*), 55
serializeOn,mtkBasicMonteCarloDesigner-method (*mtkBasicMonteCarloDesigner-class*), 60
serializeOn,mtkDefaultAnalyser-method (*mtkDefaultAnalyser-class*), 65
serializeOn,mtkDesigner-method (*mtkDesigner-class*), 68
serializeOn,mtkEvaluator-method (*mtkEvaluator-class*), 75
serializeOn,mtkExperiment-method (*mtkExperiment-class*), 80
serializeOn,mtkExpWorkflow-method (*mtkExpWorkflow-class*), 85
serializeOn,mtkFastAnalyser-method (*mtkFastAnalyser-class*), 91
serializeOn,mtkFastDesigner-method (*mtkFastDesigner-class*), 96
serializeOn,mtkIshigamiEvaluator-method (*mtkIshigamiEvaluator-class*), 103
serializeOn,mtkMorrisAnalyser-method (*mtkMorrisAnalyser-class*), 108
serializeOn,mtkMorrisDesigner-method (*mtkMorrisDesigner-class*), 113
serializeOn,mtkNativeAnalyser-method (*mtkNativeAnalyser-class*), 118
serializeOn,mtkNativeDesigner-method (*mtkNativeDesigner-class*), 121
serializeOn,mtkNativeEvaluator-method (*mtkNativeEvaluator-class*), 125
serializeOn,mtkPLMMAnalyser-method (*mtkPLMMAnalyser-class*), 133
serializeOn,mtkProcess-method (*mtkProcess-class*), 138
serializeOn,mtkRandLHSDesigner-method (*mtkRandLHSDesigner-class*), 140
serializeOn,mtkRegressionAnalyser-method (*mtkRegressionAnalyser-class*), 144
serializeOn,mtkResult-method (*mtkResult-class*), 149
serializeOn,mtkSobolAnalyser-method (*mtkSobolAnalyser-class*), 151
serializeOn,mtkSobolDesigner-method (*mtkSobolDesigner-class*), 155
serializeOn,mtkSystemEvaluator-method (*mtkSystemEvaluator-class*), 159
serializeOn,mtkWWDMEvaluator-method (*mtkWWDMEvaluator-class*), 165
serializeOn-methods, 181
setDistributionParameters, *73*
setDistributionParameters (*setDistributionParameters-methods*), 182
setDistributionParameters,mtkDomain,list-metho (*mtkDomain-class*), 72
setDistributionParameters-methods, 182
setDomain, *90*
setDomain (*setDomain-methods*), 183
setDomain,mtkFactor,mtkDomain-method (*mtkFactor-class*), 89
setDomain-methods, 183
setFactors (*setFactors-methods*), 184
setFactors,mtkExpFactors,list-method (*mtkExpFactors-class*), 82
setFactors-methods, 184
setFeatures, *90*
setFeatures (*setFeatures-methods*), 185
setFeatures,mtkFactor,list-method (*mtkFactor-class*), 89
setFeatures-methods, 185
setLevels, *73*, *106*
setLevels (*setLevels-methods*), 185
setLevels,mtkDomain,list-method (*mtkDomain-class*), 72
setLevels,mtkDomain,mtkLevels-method (*mtkDomain-class*), 72
setLevels,mtkLevels,vector-method (*mtkLevels-class*), 106
setLevels-methods, 185
setName, *56*, *75*, *90*, *92*, *97*, *101*, *103*, *109*, *114*, *119*, *122*, *126*, *129*, *133*, *138*, *141*, *145*, *151*, *155*, *159*, *163*, *166*
setName (*setName-methods*), 186
setName,mtkAnalyser,character-method (*mtkAnalyser-class*), 55
setName,mtkBasicMonteCarloDesigner,character-m

(mtkBasicMonteCarloDesigner-class), 60
setName,mtkDefaultAnalyser,character-method (mtkDefaultAnalyser-class), 65
setName,mtkDesigner,character-method (mtkDesigner-class), 68
setName,mtkEvaluator,character-method (mtkEvaluator-class), 75
setName,mtkFactor,character-method (mtkFactor-class), 89
setName,mtkFastAnalyser,character-method (mtkFastAnalyser-class), 91
setName,mtkFastDesigner,character-method (mtkFastDesigner-class), 96
setName,mtkFeature,character-method (mtkFeature-class), 101
setName,mtkIshigamiEvaluator,character-method (mtkIshigamiEvaluator-class), 103
setName,mtkMorrisAnalyser,character-method (mtkMorrisAnalyser-class), 108
setName,mtkMorrisDesigner,character-method (mtkMorrisDesigner-class), 113
setName,mtkNativeAnalyser,character-method (mtkNativeAnalyser-class), 118
setName,mtkNativeDesigner,character-method (mtkNativeDesigner-class), 121
setName,mtkNativeEvaluator,character-method (mtkNativeEvaluator-class), 125
setName,mtkParameter,character-method (mtkParameter-class), 129
setName,mtkPLMMAnalyser,character-method (mtkPLMMAnalyser-class), 133
setName,mtkProcess,character-method (mtkProcess-class), 138
setName,mtkRandLHSDesigner,character-method (mtkRandLHSDesigner-class), 140
setName,mtkRegressionAnalyser,character-method (mtkRegressionAnalyser-class), 144
setName,mtkSobolAnalyser,character-method (mtkSobolAnalyser-class), 151
setName,mtkSobolDesigner,character-method (mtkSobolDesigner-class), 155
setName,mtkSystemEvaluator,character-method (mtkSystemEvaluator-class), 159
setName,mtkValue,character-method (mtkValue-class), 162
setName,mtkWWDMEvaluator,character-method (mtkWWDMEvaluator-class), 165
setName-methods, 186
setParameters, *56*, *75*, *92*, *97*, *103*, *109*, *114*, *119*, *122*, *126*, *133*, *138*, *141*, *145*, *151*, *155*, *159*, *166*
setParameters (setParameters-methods), 187
setParameters,mtkAnalyser,vector-method (mtkAnalyser-class), 55
setParameters,mtkBasicMonteCarloDesigner,vecto (mtkBasicMonteCarloDesigner-class), 60
setParameters,mtkDefaultAnalyser,vector-metho (mtkDefaultAnalyser-class), 65
setParameters,mtkDesigner,vector-method (mtkDesigner-class), 68
setParameters,mtkEvaluator,vector-method (mtkEvaluator-class), 75
setParameters,mtkFastAnalyser,vector-method (mtkFastAnalyser-class), 91
setParameters,mtkFastDesigner,vector-method (mtkFastDesigner-class), 96
setParameters,mtkIshigamiEvaluator,vector-meth (mtkIshigamiEvaluator-class), 103
setParameters,mtkMorrisAnalyser,vector-method (mtkMorrisAnalyser-class), 108
setParameters,mtkMorrisDesigner,vector-method (mtkMorrisDesigner-class), 113
setParameters,mtkNativeAnalyser,vector-method (mtkNativeAnalyser-class), 118
setParameters,mtkNativeDesigner,vector-method (mtkNativeDesigner-class), 121
setParameters,mtkNativeEvaluator,vector-metho (mtkNativeEvaluator-class), 125
setParameters,mtkPLMMAnalyser,vector-method

(mtkPLMMAnalyser-class), 133
setParameters,mtkProcess,vector-method (mtkProcess-class), 138
setParameters,mtkRandLHSDesigner,vector-method (mtkRandLHSDesigner-class), 140
setParameters,mtkRegressionAnalyser,vector-method (mtkRegressionAnalyser-class), 144
setParameters,mtkSobolAnalyser,vector-method (mtkSobolAnalyser-class), 151
setParameters,mtkSobolDesigner,vector-method (mtkSobolDesigner-class), 155
setParameters,mtkSystemEvaluator,vector-method (mtkSystemEvaluator-class), 159
setParameters,mtkWWDMEvaluator,vector-method (mtkWWDMEvaluator-class), 165
setParameters-methods, 187
setProcess, *80*, *86*
setProcess (setProcess-methods), 188
setProcess,mtkExperiment,mtkProcess,character-method (mtkExperiment-class), 80
setProcess,mtkExpWorkflow,mtkProcess,character-method (mtkExpWorkflow-class), 85
setProcess-methods, 188
setReady, *56*, *76*, *92*, *97*, *103*, *104*, *109*, *114*, *119*, *122*, *126*, *133*, *134*, *138*, *141*, *145*, *152*, *155*, *159*, *166*
setReady (setReady-methods), 189
setReady,mtkAnalyser,logical-method (mtkAnalyser-class), 55
setReady,mtkBasicMonteCarloDesigner,logical-method (mtkBasicMonteCarloDesigner-class), 60
setReady,mtkDefaultAnalyser,logical-method (mtkDefaultAnalyser-class), 65
setReady,mtkDesigner,logical-method (mtkDesigner-class), 68
setReady,mtkEvaluator,logical-method (mtkEvaluator-class), 75
setReady,mtkFastAnalyser,logical-method (mtkFastAnalyser-class), 91
setReady,mtkFastDesigner,logical-method (mtkFastDesigner-class), 96
setReady,mtkIshigamiEvaluator,logical-method (mtkIshigamiEvaluator-class), 103
setReady,mtkMorrisAnalyser,logical-method (mtkMorrisAnalyser-class), 108
setReady,mtkMorrisDesigner,logical-method (mtkMorrisDesigner-class), 113
setReady,mtkNativeAnalyser,logical-method (mtkNativeAnalyser-class), 118
setReady,mtkNativeDesigner,logical-method (mtkNativeDesigner-class), 121
setReady,mtkNativeEvaluator,logical-method (mtkNativeEvaluator-class), 125
setReady,mtkPLMMAnalyser,logical-method (mtkPLMMAnalyser-class), 133
setReady,mtkProcess,logical-method (mtkProcess-class), 138
setReady,mtkRandLHSDesigner,logical-method (mtkRandLHSDesigner-class), 140
setReady,mtkRegressionAnalyser,logical-method (mtkRegressionAnalyser-class), 144
setReady,mtkSobolAnalyser,logical-method (mtkSobolAnalyser-class), 151
setReady,mtkSobolDesigner,logical-method (mtkSobolDesigner-class), 155
setReady,mtkSystemEvaluator,logical-method (mtkSystemEvaluator-class), 159
setReady,mtkWWDMEvaluator,logical-method (mtkWWDMEvaluator-class), 165
setReady-methods, 189
setState (setState-methods), 190
setState,mtkAnalyser,logical-method (mtkAnalyser-class), 55
setState,mtkBasicMonteCarloDesigner,logical-method (mtkBasicMonteCarloDesigner-class), 60
setState,mtkDefaultAnalyser,logical-method (mtkDefaultAnalyser-class), 65
setState,mtkDesigner,logical-method (mtkDesigner-class), 68

setState,mtkEvaluator,logical-method (*mtkEvaluator-class*), 75
setState,mtkFastAnalyser,logical-method (*mtkFastAnalyser-class*), 91
setState,mtkFastDesigner,logical-method (*mtkFastDesigner-class*), 96
setState,mtkIshigamiEvaluator,logical-method (*mtkIshigamiEvaluator-class*), 103
setState,mtkMorrisAnalyser,logical-method (*mtkMorrisAnalyser-class*), 108
setState,mtkMorrisDesigner,logical-method (*mtkMorrisDesigner-class*), 113
setState,mtkNativeAnalyser,logical-method (*mtkNativeAnalyser-class*), 118
setState,mtkNativeDesigner,logical-method (*mtkNativeDesigner-class*), 121
setState,mtkNativeEvaluator,logical-method (*mtkNativeEvaluator-class*), 125
setState,mtkPLMMAnalyser,logical-method (*mtkPLMMAnalyser-class*), 133
setState,mtkProcess,logical-method (*mtkProcess-class*), 138
setState,mtkRandLHSDesigner,logical-method (*mtkRandLHSDesigner-class*), 140
setState,mtkRegressionAnalyser,logical-method (*mtkRegressionAnalyser-class*), 144
setState,mtkSobolAnalyser,logical-method (*mtkSobolAnalyser-class*), 151
setState,mtkSobolDesigner,logical-method (*mtkSobolDesigner-class*), 155
setState,mtkSystemEvaluator,logical-method (*mtkSystemEvaluator-class*), 159
setState,mtkWWDMEvaluator,logical-method (*mtkWWDMEvaluator-class*), 165
setState-methods, 190
setType, *90*, *101*, *106*, *129*, *163*
setType (*setType-methods*), 191
setType,mtkFactor,character-method (*mtkFactor-class*), 89
setType,mtkFeature,character-method (*mtkFeature-class*), 101
setType,mtkLevels,character-method (*mtkLevels-class*), 106
setType,mtkParameter,character-method (*mtkParameter-class*), 129
setType,mtkValue,character-method (*mtkValue-class*), 162
setType-methods, 191
setValue, *101*, *129*, *163*
setValue (*setValue-methods*), 192
setValue,mtkFeature,ANY-method (*mtkFeature-class*), 101
setValue,mtkParameter,ANY-method (*mtkParameter-class*), 129
setValue,mtkValue,ANY-method (*mtkValue-class*), 162
setValue,mtkValue-method (*mtkValue-class*), 162
setValue-methods, 192
setWeights, *106*
setWeights (*setWeights-methods*), 193
setWeights,mtkLevels,numeric-method (*mtkLevels-class*), 106
setWeights-methods, 193
setXMLFilePath, *131*
setXMLFilePath (*setXMLFilePath-methods*), 194
setXMLFilePath,mtkParsor,character-method (*mtkParsor-class*), 131
setXMLFilePath-methods, 194
show, *73*, *83*, *90*, *101*, *106*, *129*, *163*
show,mtkDomain-method (*mtkDomain-class*), 72
show,mtkExpFactors-method (*mtkExpFactors-class*), 82
show,mtkFactor-method (*mtkFactor-class*), 89
show,mtkFeature-method (*mtkFeature-class*), 101
show,mtkLevels-method (*mtkLevels-class*), 106
show,mtkParameter-method (*mtkParameter-class*), 129
show,mtkValue-method (*mtkValue-class*), 162
Sobol, 195
summary, *56*, *58*, *63*, *71*, *76*, *78*, *81*, *86*, *92*, *95*, *97*, *99*, *104*, *106*, *109*, *111*, *114*, *116*, *119*, *122*, *126*, *134*, *136*, *139*,

*141*, *142*, *146*, *148*, *149*, *152*, *153*, *156*, *157*, *160*, *161*, *166*, *168*
summary (*summary,mtkProcess-method*), 198
summary,mtkAnalyser-method (*mtkAnalyser-class*), 55
summary,mtkAnalyserResult-method (*mtkAnalyserResult-class*), 58
summary,mtkBasicMonteCarloDesigner-method (*mtkBasicMonteCarloDesigner-class*), 60
summary,mtkBasicMonteCarloDesignerResult-method (*mtkBasicMonteCarloDesignerResult-class*), 63
summary,mtkDefaultAnalyser-method (*mtkDefaultAnalyser-class*), 65
summary,mtkDesigner-method (*mtkDesigner-class*), 68
summary,mtkDesignerResult-method (*mtkDesignerResult-class*), 70
summary,mtkEvaluator-method (*mtkEvaluator-class*), 75
summary,mtkEvaluatorResult-method (*mtkEvaluatorResult-class*), 77
summary,mtkExperiment-method (*mtkExperiment-class*), 80
summary,mtkExpWorkflow-method (*mtkExpWorkflow-class*), 85
summary,mtkFastAnalyser-method (*mtkFastAnalyser-class*), 91
summary,mtkFastAnalyserResult-method (*mtkFastAnalyserResult-class*), 94
summary,mtkFastDesigner-method (*mtkFastDesigner-class*), 96
summary,mtkFastDesignerResult-method (*mtkFastDesignerResult-class*), 99
summary,mtkIshigamiEvaluator-method (*mtkIshigamiEvaluator-class*), 103
summary,mtkLevels-method (*mtkLevels-class*), 106
summary,mtkMorrisAnalyser-method (*mtkMorrisAnalyser-class*), 108
summary,mtkMorrisAnalyserResult-method (*mtkMorrisAnalyserResult-class*), 111
summary,mtkMorrisDesigner-method (*mtkMorrisDesigner-class*), 113
summary,mtkMorrisDesignerResult-method (*mtkMorrisDesignerResult-class*), 116
summary,mtkNativeAnalyser-method (*mtkNativeAnalyser-class*), 118
summary,mtkNativeDesigner-method (*mtkNativeDesigner-class*), 121
summary,mtkNativeEvaluator-method (*mtkNativeEvaluator-class*), 125
summary,mtkPLMMAnalyser-method (*mtkPLMMAnalyser-class*), 133
summary,mtkPLMMAnalyserResult-method (*mtkPLMMAnalyserResult-class*), 135
summary,mtkProcess-method, 198
summary,mtkRandLHSDesigner-method (*mtkRandLHSDesigner-class*), 140
summary,mtkRandLHSDesignerResult-method (*mtkRandLHSDesignerResult-class*), 142
summary,mtkRegressionAnalyser-method (*mtkRegressionAnalyser-class*), 144
summary,mtkRegressionAnalyserResult-method (*mtkRegressionAnalyserResult-class*), 147
summary,mtkResult-method (*mtkResult-class*), 149
summary,mtkSobolAnalyser-method (*mtkSobolAnalyser-class*), 151
summary,mtkSobolAnalyserResult-method (*mtkSobolAnalyserResult-class*), 153
summary,mtkSobolDesigner-method (*mtkSobolDesigner-class*), 155
summary,mtkSobolDesignerResult-method (*mtkSobolDesignerResult-class*), 157
summary,mtkSystemEvaluator-method (*mtkSystemEvaluator-class*),

159
summary,mtkSystemEvaluatorResult-method
*(mtkSystemEvaluatorResult-class)*,
159
summary,mtkWWDMEvaluator-method
*(mtkWWDMEvaluator-class)*,
165
summary,mtkWWDMEvaluatorResult-method
*(mtkWWDMEvaluatorResult-class)*,
168

vector, *56*, *60*, *65*, *68*, *75*, *80*, *84*, *86*, *92*, *97*,
*103*, *106*, *108*, *114*, *119*, *122*, *126*,
*133*, *138*, *140*, *145*, *151*, *155*, *159*,
*165*

WWDM, 199, *201, 202*
WWDM.climates *(wwdm.climates)*, 201
wwdm.climates, 201
WWDM.factors, *200*, 202