

# Package: rrepast (via r-universe)

October 17, 2024

**Type** Package

**Title** Invoke 'Repast Symphony' Simulation Models

**Version** 0.8.0

**Date** 2020-02-18

**Encoding** UTF-8

**Author** Antonio Prestes Garcia [aut, cre], Alfonso Rodriguez-Paton  
[aut, ths]

**Maintainer** Antonio Prestes Garcia <antonio.pgarcia@alumnos.upm.es>

**URL** <https://github.com/antonio-pgarcia/rrepast>

**BugReports** <https://github.com/antonio-pgarcia/RRepast/issues>

**Description** An R and Repast integration tool for running individual-based (IbM) simulation models developed using 'Repast Symphony' Agent-Based framework directly from R code supporting multicore execution. This package integrates 'Repast Symphony' models within R environment, making easier the tasks of running and analyzing model output data for automated parameter calibration and for carrying out uncertainty and sensitivity analysis using the power of R environment.

**License** MIT + file LICENSE

**Imports** lhs, sensitivity, ggplot2, digest, xlsx, doSNOW, rJava, gridExtra, parallel, foreach

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-02-19 06:00:05 UTC

## Contents

AddFactor . . . . .	5
AddFactor0 . . . . .	5

AddResults	6
AoE.Base	6
AoE.ColumnCoV	7
AoE.CoV	7
AoE.FullFactorial	8
AoE.GetMorrisOutput	8
AoE.LatinHypercube	9
AoE.MAE	9
AoE.Morris	10
AoE.NRMSD	10
AoE.RandomSampling	11
AoE.RMSD	12
AoE.Sobol	12
AoE.Stability	13
ApplyFactorRange	13
BuildParameterSet	14
Calibration.GetMemberKeys	14
Calibration.GetMemberList	15
check.integration	15
check.scenario	16
ClearResults	16
col.sum	17
config.check	17
config.copylib	18
config.scenario	18
createOutputDir	19
df2matrix	19
dffilterby	20
dfround	20
dfsumcol	21
Easy.Calibration	21
Easy.getChart	22
Easy.getPlot	23
Easy.Morris	23
Easy.Run	24
Easy.RunExperiment	24
Easy.Setup	25
Easy.ShowModelParameters	25
Easy.Sobol	26
Easy.Stability	26
Engine	27
Engine.endAt	28
Engine.Finish	28
Engine.getId	28
Engine.GetModelOutput	29
Engine.getParameter	29
Engine.getParameterAsDouble	30
Engine.getParameterAsNumber	30

Engine.getParameterAsString . . . . .	31
Engine.getParameterNames . . . . .	31
Engine.getParameterType . . . . .	32
Engine.LoadModel . . . . .	32
Engine.resetModelOutput . . . . .	33
Engine.RunModel . . . . .	33
Engine.SetAggregateDataSet . . . . .	33
Engine.setParameter . . . . .	34
enginestats.calls . . . . .	35
enginestats.reset . . . . .	35
getExperimentDataset . . . . .	35
getExperimentOutput . . . . .	36
getExperimentParamSet . . . . .	36
GetFactorLevels . . . . .	37
GetFactorsSize . . . . .	38
getId . . . . .	38
getKeyRandom . . . . .	38
getLogDir . . . . .	39
GetOutput . . . . .	39
getOutputDir . . . . .	40
getpkgcores . . . . .	40
getpkgdefaultcores . . . . .	40
GetResults . . . . .	41
GetResultsParameters . . . . .	41
GetSimulationParameters . . . . .	41
GetSimulationParameterType . . . . .	42
GoToPreviousDir . . . . .	42
GoToWorkDir . . . . .	42
hybrid.distance . . . . .	43
hybrid.value . . . . .	43
jarfile . . . . .	44
jvm.enablejmx . . . . .	44
jvm.getRuntime . . . . .	45
jvm.get_parameters . . . . .	45
jvm.init . . . . .	45
jvm.memory . . . . .	46
jvm.resetOut . . . . .	46
jvm.runtimegc . . . . .	47
jvm.setOut . . . . .	47
jvm.set_parameters . . . . .	48
lcontains . . . . .	48
lget . . . . .	49
Load . . . . .	49
Logger.setLevelInfo . . . . .	50
Logger.setLevelWarning . . . . .	50
Model . . . . .	50
ParallelClose . . . . .	51
ParallelInit . . . . .	51

parallelize . . . . .	52
ParallelRunExperiment . . . . .	52
ParallelRun . . . . .	53
PB.close . . . . .	54
PB.disable . . . . .	54
PB.enable . . . . .	54
PB.get . . . . .	55
PB.init . . . . .	55
PB.isEnabled . . . . .	55
PB.pset . . . . .	56
PB.rnum . . . . .	56
PB.set . . . . .	56
PB.update . . . . .	57
pick.fittest . . . . .	57
Plot.Calibration . . . . .	58
Plot.Morris . . . . .	58
Plot.Sobol . . . . .	59
Plot.Stability . . . . .	59
Results.GetCharts . . . . .	60
Results.GetExperiment . . . . .	60
Results.GetObject . . . . .	61
Run . . . . .	61
RunExperiment . . . . .	62
SaveSimulationData . . . . .	63
SequenceItem . . . . .	63
setId . . . . .	64
setKeyRandom . . . . .	64
setOutputDir . . . . .	64
setpkgcores . . . . .	65
SetResults . . . . .	65
SetResultsParameters . . . . .	65
SetSimulationParameter . . . . .	66
SetSimulationParameters . . . . .	66
ShowClassPath . . . . .	67
ShowCores . . . . .	67
ShowModelPaths . . . . .	68
UpdateDefaultParameters . . . . .	68
WrapperRun . . . . .	69
WrapperRunExperiment . . . . .	69

---

AddFactor	<i>Adds a paramter to factor collection</i>
-----------	---

---

**Description**

Builds up the factor collection.

**Usage**

```
AddFactor(factors = c(), lambda = "qunif", name, min, max,
           int = FALSE)
```

**Arguments**

factors	The current factor collection
lambda	The function to apply FUN(p,min,max)
name	The name of factor
min	The minimum of parameter p
max	The maximum of parameter p
int	Boolean for truncating the factor value

**Value**

The collection of created factors

**Examples**

```
## Not run:
f<- AddFactor(name="Age",min=20,max=60)
f<- AddFactor(factors=f, name="Weight",min=50,max=120)
## End(Not run)
```

---

AddFactor0	<i>AddFactor0</i>
------------	-------------------

---

**Description**

Creates or appends the factor collection

**Usage**

```
AddFactor0(factors = c(), ...)
```

**Arguments**

factors            The current factor collection  
 ...                The variadic parameter list

**Value**

The factor collection

**Examples**

```
## Not run:
f<- AddFactor0(name="Age",min=20,max=60)
f<- AddFactor0(factors=f, name="Weight",min=50,max=120)
## End(Not run)
```

---

AddResults	<i>Concatenate results of multiple runs</i>
------------	---

---

**Description**

This function stores the output of the last model execution and it is intended to be used internally.

**Usage**

```
AddResults(d)
```

**Arguments**

d                    A data frame containing one replication data

---

AoE.Base	<i>AoE.Base</i>
----------	-----------------

---

**Description**

The Design Of Experiments Base function

**Usage**

```
AoE.Base(m, factors = c(), fun = NULL)
```

**Arguments**

m                    The base design matrix  
 factors            A subset of model parameters  
 fun                 The function which will be applied to m

**Value**

The design matrix

---

AoE.ColumnCoV

*AoE.ColumnCoV*

---

**Description**

This function Calculates the relative squared deviation (RSD or CoV) for an used provided column name key in the parameter dataset.

**Usage**

AoE.ColumnCoV(dataset, key)

**Arguments**

dataset	A model output dataset
key	Column name from output dataset

**Value**

A data frame with Coefficient of variations

---

AoE.CoV

*AoE.CoV*

---

**Description**

A simple function for calculate the Coefficient of Variation

**Usage**

AoE.CoV(d)

**Arguments**

d	The data collection
---	---------------------

**Value**

The coefficient of variation for data

AoE.FullFactorial      *AoE.FullFactorial design generator*

---

**Description**

Generate a Full Factorial sampling for evaluating the parameters of a model.

**Usage**

```
AoE.FullFactorial(n = 10, factors = c())
```

**Arguments**

n	The number of samples
factors	The model's parameters which will be evaluated

**Value**

The Full Factorial design matrix for provided parameters

**Examples**

```
## Not run:  
f<- AddFactor(name="cyclePoint",min=40,max=90)  
f<- AddFactor(factors=f, name="conjugationCost",min=1,max=80)  
d<- AoE.FullFactorial(2,f)  
## End(Not run)
```

---

AoE.GetMorrisOutput      *AoE.GetMorrisOutput*

---

**Description**

Returns a dataframe holding the Morris result set

**Usage**

```
AoE.GetMorrisOutput(obj)
```

**Arguments**

obj	A reference to a morris object instance
-----	---

**Value**

The results of Morris method



---

AoE.LatinHypercube      *AoE.LatinHypercube*

---

**Description**

Generate a LHS sample for model parameters

**Usage**

```
AoE.LatinHypercube(n = 10, factors = c(), convert = TRUE)
```

**Arguments**

n	The number of samples
factors	The model's parameters which will be evaluated
convert	Adjust experiment matrix to parameter scale

**Details**

Generate the LHS sampling for evaluating the parameters of a model.

**Value**

The LHS design matrix for provided parameters

**Examples**

```
## Not run:  
f<- AddFactor(name="cyclePoint",min=40,max=90)  
f<- AddFactor(factors=f, name="conjugationCost",min=1,max=80)  
d<- AoE.LatinHypercube(2,f)  
## End(Not run)
```

---

AoE.MAE      *AoE.MAE*

---

**Description**

Calculates the average-error magnitude (MAE)

**Usage**

```
AoE.MAE(xs, xe)
```

**Arguments**

<code>xs</code>	The simulated data set
<code>xe</code>	The experimental data set

**Value**

The MAE value for provided datasets

---

<code>AoE.Morris</code>	<i>AoE.Morris</i>
-------------------------	-------------------

---

**Description**

This is a wrapper for performing Morris's screening method on repast models. We rely on morris method from sensitivity package.

**Usage**

```
AoE.Morris(k = c(), p = 5, r = 4)
```

**Arguments**

<code>k</code>	The factors for morris screening.
<code>p</code>	The number of levels for the model's factors.
<code>r</code>	Repetitions. The number of random sampling points of Morris Method.

**References**

Gilles Pujol, Bertrand Iooss, Alexandre Janon with contributions from Sebastien Da Veiga, Jana Fruth, Laurent Gilquin, Joseph Guillaume, Loic Le Gratiet, Paul Lemaitre, Bernardo Ramos and Taieb Touati (2015). sensitivity: Sensitivity Analysis. R package version 1.11.1. <https://CRAN.R-project.org/package=sensitivity>

---

<code>AoE.NRMSD</code>	<i>AoE.NRMSD</i>
------------------------	------------------

---

**Description**

A simple Normalized Root-Mean-Square Deviation calculation using max and min values.  $NRMSD = RMSD(x) / (\max(x) - \min(x))$

**Usage**

```
AoE.NRMSD(xs, xe)
```

**Arguments**

xs	The simulated data set
xe	The experimental data set

**Value**

The NRRMSD value for provided datasets

---

AoE.RandomSampling     *AoE.RandomSampling experiment desing generator*

---

**Description**

Generate a Simple Random Sampling experiment design matrix.

**Usage**

```
AoE.RandomSampling(n = 10, factors = c())
```

**Arguments**

n	The number of samples
factors	The model's parameters which will be evaluated

**Value**

The random sampling design matrix

**Examples**

```
## Not run:  
f<- AddFactor(name="cyclePoint",min=40,max=90)  
f<- AddFactor(factors=f, name="conjugationCost",min=1,max=80)  
d<- AoE.RandomSampling(2,f)  
## End(Not run)
```

---

 AoE.RMSD

*AoE.RMSD*


---

**Description**

A simple Root-Mean-Square Deviation calculation.

**Usage**

```
AoE.RMSD(xs, xe)
```

**Arguments**

xs	The simulated data set
xe	The experimental data set

**Value**

The RMSD value for provided datasets

---

AoE.Sobol

*AoE.Sobol*


---

**Description**

This is a wrapper for performing Global Sensitivity Analysis using the Sobol Method provided by sensitivity package.

**Usage**

```
AoE.Sobol(n = 100, factors = c(), o = 2, nb = 100,
  fun.doe = AoE.LatinHypercube, fun.sobol = sobolmartinez)
```

**Arguments**

n	The number of samples
factors	The model's parameters which will be evaluated
o	Maximum order in the ANOVA decomposition
nb	Number of bootstrap replicates
fun.doe	The sampling function to be used for sobol method
fun.sobol	The sobol implementation

**Details**

This function is not intended to be used directly from user programs.

**References**

Gilles Pujol, Bertrand Iooss, Alexandre Janon with contributions from Sebastien Da Veiga, Jana Fruth, Laurent Gilquin, Joseph Guillaume, Loic Le Gratiet, Paul Lemaitre, Bernardo Ramos and Taieb Touati (2015). sensitivity: Sensitivity Analysis. R package version 1.11.1. <https://CRAN.R-project.org/package=sensitivity>

---

AoE.Stability	<i>AoE.Stability</i>
---------------	----------------------

---

**Description**

This function verifies the stability of CoV for all columns given by parameter keys or all dataset columns if keys is empty.

**Usage**

```
AoE.Stability(dataset, keys = c())
```

**Arguments**

dataset	A model output dataset
keys	A list of column names

**Value**

A data frame with Coefficient of variations

---

ApplyFactorRange	<i>Corrects the LHS design matrix</i>
------------------	---------------------------------------

---

**Description**

Correct the LHS sampling matrix for a specific range applying the lambda function. The default value of 'lambda' is 'qunif'.

**Usage**

```
ApplyFactorRange(design, factors)
```

**Arguments**

design	The LHS design matrix
factors	The collection of factors

**Value**

The corrected design matrix

---

BuildParameterSet      *Builds the simulation parameter set*

---

### Description

Merges the design matrix with parameters which will be keep fixed along simulation runs.

### Usage

```
BuildParameterSet(design, parameters)
```

### Arguments

design                    The experimental desing matrix for at least one factor  
parameters              All parameters of the repast model.

### Value

A data frame holding all parameters required for running the model

### Examples

```
## Not run:
modeldir<- "c:/usr/models/BactoSim(HaldaneEngine-1.0)"
e<- Model(modeldir=modeldir,dataset="ds::Output")
Load(e)

f<- AddFactor(name="cyclePoint",min=40,max=90)

p<- GetSimulationParameters(e)

d<- AoE.LatinHypercube(factors=f)

p1<- BuildParameterSet(d,p)
## End(Not run)
```

---

Calibration.GetMemberKeys  
*Calibration.GetMemberKeys*

---

### Description

Gets the list of keys (the factor names)

**Usage**

Calibration.GetMemberKeys(obj)

**Arguments**

obj                    An instance of the object returned by Easy methods

**Value**

The collection of keys

---

Calibration.GetMemberList

*Calibration.GetMemberList*

---

**Description**

Gets the member list value

**Usage**

Calibration.GetMemberList(obj, key, name)

**Arguments**

obj                    An instance of the object returned by Easy methods

key                    The key value

name                   The column name

**Value**

The member list

---

check.integration

*check.integration*

---

**Description**

Check if the integration jar library is correctly installed in the model lib directory

**Usage**

check.integration(modelpath)

**Arguments**

modelpath      The path where model is installed

**Value**

TRUE if the integration code is correctly deployed

---

check.scenario      *check.scenario*

---

**Description**

Check if the scenario.xml is configured with the rrepast itegration code

**Usage**

```
check.scenario(modelpath)
```

**Arguments**

modelpath      The path where model is installed

**Value**

TRUE if scenario is properly configured

---

ClearResults      *Clear the results data.frame*

---

**Description**

This function is called automatically every time Run method is called.

**Usage**

```
ClearResults()
```



---

col.sum	<i>col.sum</i>
---------	----------------

---

**Description**

Sum all columns but one (pset) of a data frame

**Usage**

```
col.sum(d, skip = c())
```

**Arguments**

d	The data frame
skip	The columns which should not be included in the sum

**Value**

The original data frame with a new column (sum) holding the sum

---

config.check	<i>config.check</i>
--------------	---------------------

---

**Description**

Verify if the installed model is correctly configured.

**Usage**

```
config.check(modelpath)
```

**Arguments**

modelpath	The path where model is installed
-----------	-----------------------------------

**Value**

TRUE when all requisites are met

---

config.copylib	<i>config.copylib</i>
----------------	-----------------------

---

**Description**

Install or uninstall the integration jar file. This function manages the installation process of required jars to the model lib dir.

**Usage**

```
config.copylib(modelpath, uninstall = FALSE)
```

**Arguments**

modelpath	The path where model is installed
uninstall	If TRUE uninstall integration jar

**Value**

TRUE if install operation succeed

---

config.scenario	<i>config.scenario</i>
-----------------	------------------------

---

**Description**

Add the integration library to the model's configuration

**Usage**

```
config.scenario(modelpath, uninstall = FALSE)
```

**Arguments**

modelpath	The path where model is installed
uninstall	If TRUE restore original scenario.xml file

**Value**

A logical TRUE if the model's scenario file has been modified

---

createOutputDir	<i>Create output directory</i>
-----------------	--------------------------------

---

**Description**

A simple function to make a directory to save the model's data.

**Usage**

```
createOutputDir()
```

**Details**

Create the, if required, the directory to save the output data generate by the model. It is intended for internal use.

---

df2matrix	<i>df2matrix</i>
-----------	------------------

---

**Description**

This function converts data frames to matrix data type.

**Usage**

```
df2matrix(d, n = c())
```

**Arguments**

d	The data frame
n	The column names to be converted. Null for all data frame columns

**Value**

The data frame converted to a matrix

*dfilterby**dfilterby*

---

**Description**

Selects a subset of a data frame, filtering by column values.

**Usage**

```
dfilterby(d, key, values = c())
```

**Arguments**

d	The data frame holding data to be filtered
key	The column name for selection values
values	The collection of values used to filter the data set

**Value**

The filtered data set

---

*dfround**dfround*

---

**Description**

Round all numeric columns of a data frame

**Usage**

```
dfround(d, p)
```

**Arguments**

d	The data frame
p	The number of decimal digits to be kept

**Value**

A data frame with rounded columns

---

dfsumcol	<i>dfsumcol</i>
----------	-----------------

---

**Description**

Sum data frame columns but tho

**Usage**

```
dfsumcol(d, lst = c(), invert = FALSE)
```

**Arguments**

d	The data frame
lst	Skip columns included. Sum columns NOT included
invert	Sum only the columns included in lst

**Value**

The original data frame with a new column (sum) holding the sum

---

Easy.Calibration	<i>Easy.Calibration</i>
------------------	-------------------------

---

**Description**

Search for the best set of parameters trying to minimize the calibration function provided by the user. The function has to operational models, the first based on the experimental setup where all parameters are defined a priori and the second using optimization techniques. Currently the only supported optimization technique is the particle swarm optimization.

**Usage**

```
Easy.Calibration(m.dir, m.ds, m.time = 300, parameters, exp.n = 100,  
exp.r = 1, smax = 4, design = "lhs", FUN, default = NULL)
```

**Arguments**

m.dir	The installation directory of some repast model
m.ds	The name of any model aggregate dataset
m.time	The total simulated time
parameters	The input factors
exp.n	The experiment sample size
exp.r	The number of experiment replications

smax	The number of solutions to be generated
design	The sampling scheme ["lhs" "mcs" "ffs"]
FUN	The objective or cost function. A function defined over the model output.
default	The alternative values for parameters which should be kept fixed

**Value**

A list with holding experiment, object and charts

**Examples**

```
## Not run:
my.cost<- function(params, results) {
  criteria<- c()
  Rate<- AoE.RMSD(results$X.Simulated,results$X.Experimental)
  G<- AoE.RMSD(results$G.T.,52)
  total<- Rate + G
  criteria<- cbind(total,Rate,G)
  return(criteria)
}

Easy.Setup("/models/BactoSim")
v<- Easy.Calibration("/models/BactoSim","ds::Output",360,
  f,exp.n = 1000, exp.r=1, smax=4,
  design="mcs", my.cost)

## End(Not run)
```

---

Easy.getChart

*Easy.getChart*

---

**Description**

Returns the chart instance

**Usage**

```
Easy.getChart(obj, key)
```

**Arguments**

obj	A reference to the output of Easy.Stability
key	The param name

**Value**

The plot instance

---

Easy.getPlot	<i>Easy.getPlot</i>
--------------	---------------------

---

**Description**

Returns the chart instance

**Usage**

```
Easy.getPlot(obj, c, key)
```

**Arguments**

obj	A reference to the output of an "Easy" API method
c	The output name
key	The param name

**Value**

The plot instance

---

Easy.Morris	<i>Easy API for Morris's screening method</i>
-------------	---

---

**Description**

This function wraps all calls to perform Morris method.

**Usage**

```
Easy.Morris(m.dir, m.ds, m.time = 300, parameters, mo.p, mo.r, exp.r,  
FUN, default = NULL)
```

**Arguments**

m.dir	The installation directory of some repast model
m.ds	The name of any model aggregate dataset
m.time	The total simulated time
parameters	The factors for morris screening.
mo.p	The number of levels for the model's factors.
mo.r	Repetitions. The number of random sampling points of Morris Method.
exp.r	The number of experiment replications
FUN	The objective or cost function. A function defined over the model output.
default	The alternative values for parameters which should be kept fixed

**Value**

A list with holding experiment, object and charts

---

Easy.Run	<i>Easy API for running a model</i>
----------	-------------------------------------

---

**Description**

This function provides a simple wrapper for performing a single or replicated model execution with a single set of parameters.

**Usage**

```
Easy.Run(m.dir, m.ds, m.time = 300, r = 1, default = NULL)
```

**Arguments**

m.dir	The installation directory of some repast model
m.ds	The name of any model aggregate dataset
m.time	The total simulated time
r	The number of replications
default	The alternative values for the default model parameters

---

Easy.RunExperiment	<i>Easy API for Running Experiments</i>
--------------------	---

---

**Description**

This function provides a simple wrapper for performing experimental setups using a design matrix

**Usage**

```
Easy.RunExperiment(m.dir, m.ds, m.time = 300, r = 1, design, FUN,
  default = NULL)
```

**Arguments**

m.dir	The installation directory of some repast model
m.ds	The name of any model aggregate dataset
m.time	The total simulated time
r	The number of replications
design	The design matrix holding parameter sampling
FUN	The objective or cost function. A function defined over the model output.
default	The alternative values for parameters which should be kept fixed



**Value**

The experiment results

---

Easy.Setup

*Easy.Setup*

---

**Description**

This function configures the deployment directory where logs and output dataset will be generated. By default the deployment directory will be created under the model installation directory. The output generated by the Repast model will be redirected to the SystemOut.log file.

**Usage**

```
Easy.Setup(model, multicore = FALSE, deployment = c())
```

**Arguments**

model	The base directory where Repast model is installed.
multicore	Boolean flag indicating to use multiplecore.
deployment	The directory to save the output and logs.

**Details**

If the deployment directory is empty the installation directory given by the parameter `model` is used instead as the base directory. The deployment directory is `/rrepast-deployment/`.

---

Easy.ShowModelParameters

*Easy.ShowModelParameters*

---

**Description**

Returns the list current model parameters

**Usage**

```
Easy.ShowModelParameters(v)
```

**Arguments**

v	The installation directory of some repast model
---	---

**Value**

The model parameters

---

 Easy.Sobol

*Easy API for Sobol's SA method*


---

**Description**

This functions wraps all required calls to perform Sobol method for global sensitivity analysis.

**Usage**

```
Easy.Sobol(m.dir, m.ds, m.time = 300, parameters, exp.n = 500,
  bs.size = 200, exp.r = 1, FUN, default = NULL,
  fsobol = sobol2002, fsampl = AoE.LatinHypercube)
```

**Arguments**

m.dir	The installation directory of some repast model
m.ds	The name of any model aggregate dataset
m.time	The total simulated time
parameters	The input factors
exp.n	The experiment sample size
bs.size	The bootstrap sample size for sobol method
exp.r	The number of experiment replications
FUN	The objective or cost function. A function defined over the model output.
default	The alternative values for parameters which should be kept fixed
fsobol	The alternative function for calculating sobol indices
fsampl	The function for sampling data

**Value**

A list with holding experimnt, object and charts

---

 Easy.Stability

*Easy API for output stability*


---

**Description**

This functions run model several times in order to determine how many experiment replications are required for model's output being stable (i.e. the convergence of standard deviation)

**Usage**

```
Easy.Stability(m.dir, m.ds, m.time = 300, parameters, samples = 1,
  tries = 100, vars = c(), FUN, default = NULL)
```

**Arguments**

m.dir	The installation directory of some repast model
m.ds	The name of any model aggregate dataset
m.time	The total simulated time
parameters	The factors or model's parameter list
samples	The number of factor samples.
tries	The number of experiment replications
vars	The model's output variables for compute CoV
FUN	The objective or cost function. A function defined over the model output.
default	The alternative values for parameters which should be kept fixed

**Value**

A list with holding experiment, object and charts

---

Engine

*Engine*

---

**Description**

Creates an instance of Engine

**Usage**

Engine()

**Details**

This function creates an instance of Repast model wrapper class. Before invoking the function Engine, make sure that environment was correctly initialized.

**Value**

An object instance of Engine class

---

<code>Engine.endAt</code>	<i>Engine.endAt</i>
---------------------------	---------------------

---

**Description**

Configure the maximum simulated time for the current model run

**Usage**

`Engine.endAt(e, v)`

**Arguments**

<code>e</code>	An engine object instance
<code>v</code>	The number of Repast time ticks

---

<code>Engine.Finish</code>	<i>Engine.Finish</i>
----------------------------	----------------------

---

**Description**

Performs a cleanup on an engine instance. Finalize and destroy repast controller data.

**Usage**

`Engine.Finish(e)`

**Arguments**

<code>e</code>	An engine object instance
----------------	---------------------------

---

<code>Engine.getId</code>	<i>Returns the model id</i>
---------------------------	-----------------------------

---

**Description**

This function provides a wrapper to the method `getId()` from repast context. The id is basically a String with the currently instantiated model name.

**Usage**

`Engine.getId(e)`

**Arguments**

<code>e</code>	An engine object instance
----------------	---------------------------

---

Engine.GetModelOutput *Engine.GetModelOutput*

---

**Description**

Gets the model output data as a CSV String array. Calls the engine method GetModelOutput to drain model output data.

**Usage**

```
Engine.GetModelOutput(e)
```

**Arguments**

e                    An engine object instance

**Value**

An array of strings containing the model's output

**Examples**

```
## Not run:  
d<- "c:/usr/models/your-model-directory"  
m<- Model(d)  
csv<- Engine.GetModelOutput(m)  
## End(Not run)
```

---

Engine.getParameter *Engine.getParameter*

---

**Description**

The function gets the value of model parameter k as java.lang.Object

**Usage**

```
Engine.getParameter(e, k)
```

**Arguments**

e                    An engine object instance  
k                    The parameter name

**Value**

The parameter value

---

`Engine.getParameterAsDouble`  
*Engine.getParameterAsDouble*

---

**Description**

Get the value of model parameter k as `java.lang.Double`

**Usage**

`Engine.getParameterAsDouble(e, k)`

**Arguments**

e	An engine object instance
k	The parameter name

**Value**

The parameter value as double

---

`Engine.getParameterAsNumber`  
*Engine.getParameterAsNumber*

---

**Description**

Get the value of model parameter k as `java.lang.Number`

**Usage**

`Engine.getParameterAsNumber(e, k)`

**Arguments**

e	An engine object instance
k	The parameter name

**Value**

The parameter value as number

---

`Engine.getParameterAsString`  
*Engine.getParameterAsString*

---

**Description**

Get the value of model parameter k as `java.lang.String`

**Usage**

`Engine.getParameterAsString(e, k)`

**Arguments**

e	An engine object instance
k	The parameter name

**Value**

The parameter value as string

---

`Engine.getParameterNames`  
*Engine.getParameterNames*

---

**Description**

Get the parameter names

**Usage**

`Engine.getParameterNames(e)`

**Arguments**

e	An engine object instance
---	---------------------------

**Details**

Returns the names of all declared model's parameters in the `parameter.xml` file in the scenario directory.

**Value**

A collection of parameter names

Engine.getParameterType

*Engine.getParameterType*

---

### **Description**

Returns the declared type of a Repast model parameter

### **Usage**

Engine.getParameterType(e, k)

### **Arguments**

e	An engine object instance
k	The parameter name

### **Value**

The parameter type string

---

Engine.LoadModel

*Engine.LoadModel*

---

### **Description**

Loads the model's scenario files

### **Usage**

Engine.LoadModel(e, f)

### **Arguments**

e	An engine object instance
f	The full path of scenario directory

### **Details**

This function loads the scenario of a Repast Model and initialize de model.



---

Engine.resetModelOutput  
*Engine.resetModelOutput*

---

**Description**

Resets the the model output holder

**Usage**

Engine.resetModelOutput(e)

**Arguments**

e                    An engine object instance

---

Engine.RunModel            *Engine.RunModel*

---

**Description**

Performs the execution of Repast model

**Usage**

Engine.RunModel(e)

**Arguments**

e                    An engine object instance

---

Engine.SetAggregateDataSet  
*Engine.SetAggregateDataSet*

---

**Description**

Sets the model's dataset

**Usage**

Engine.SetAggregateDataSet(e, k)

**Arguments**

e	An engine object instance
k	The repast model's data set name

**Details**

Configure a dataset with the desired output values to be "drained" by the function `Engine.GetModelOutput`.

**Examples**

```
## Not run:  
d<- "C:/usr/models/your-model-directory"  
m<- Model(d)  
setAggregateDataSet(m,"dataset-name")  
## End(Not run)
```

---

`Engine.setParameter`     *Engine.setParameter*

---

**Description**

Set the value of model parameter

**Usage**

```
Engine.setParameter(e, k, v)
```

**Arguments**

e	An engine object instance
k	The parameter name
v	The parameter value

---

enginestats.calls	<i>enginestats.calls</i>
-------------------	--------------------------

---

**Description**

Return the current calls to the 'Engine.RunModel' function

**Usage**

```
enginestats.calls(increment = FALSE)
```

**Arguments**

increment      A flag telling to increment and update the counter

**Value**

The number of calls to 'Engine.RunModel'

---

enginestats.reset	<i>enginestats.reset</i>
-------------------	--------------------------

---

**Description**

Reset internal statistics

**Usage**

```
enginestats.reset()
```

---

getExperimentDataset	<i>Helper function to get experiment dataset</i>
----------------------	--

---

**Description**

The RunExperiment function returns a list holding the paramset, output and dataset collection. The paramset collection contains the parameters used for running the experimental setup. The output has the results from user provided calibration function. The dataset collection has the raw output of 'Repast' aggregated dataset.

**Usage**

```
getExperimentDataset(e)
```

**Arguments**

e                    The experiement object returned by [RunExperiment](#)

**Value**

The reference to dataset container.

---

getExperimentOutput    *Helper function to get experiment output*

---

**Description**

The RunExperiment function returns a list holding the paramset, output and dataset collection. The paramset collection contains the parameters used for running the experimental setup. The output has the results from user provided calibration function. The dataset collection has the raw output of 'Repast' aggregated dataset.

**Usage**

getExperimentOutput(e)

**Arguments**

e                    The experiement object returned by [RunExperiment](#)

**Value**

The reference to output container.

---

getExperimentParamSet    *Helper function to get experiment paramset*

---

**Description**

The RunExperiment function returns a list holding the paramset, output and dataset collection. The paramset collection contains the parameters used for running the experimental setup. The output has the results from user provided calibration function. The dataset collection has the raw output of 'Repast' aggregated dataset.

**Usage**

getExperimentParamSet(e)

**Arguments**

e                    The experiement object returned by [RunExperiment](#)

**Value**

The reference to output container.

**Examples**

```
## Not run:  
d<- "C:/usr/models/your-model-directory"  
m<- Model(d)  
...  
e<- RunExperiment(e,r=1,exp.design,my.cost)  
p<- getExperimentParamSet(e)  
## End(Not run)
```

---

GetFactorLevels

*GetFactorLevels*

---

**Description**

Returns the factor's levels

**Usage**

```
GetFactorLevels(factors, name)
```

**Arguments**

factors	The current factor collection
name	The factor name

**Value**

Levels

**Examples**

```
## Not run:  
f<- AddFactor0(name="Age", levels=c(25,30,40,65))  
f<- AddFactor0(factors=f, name="Weight", levels=c(60,70,80,90))  
  
GetFactorLevels(factors=f, "Age")  
## End(Not run)
```

---

GetFactorsSize	<i>Get the number of factors</i>
----------------	----------------------------------

---

**Description**

Returns the total number of factors

**Usage**

```
GetFactorsSize(factors)
```

**Arguments**

factors            A collection of factors created with AddFactor

**Value**

The number of parameters in factors collection

---

getId	<i>Gets the model name</i>
-------	----------------------------

---

**Description**

Provides the name of the model currently instantiated.

**Usage**

```
getId()
```

---

getKeyRandom	<i>Gets Repast randomSeed name</i>
--------------	------------------------------------

---

**Description**

Returns the Repast randomSeed parameter name.

**Usage**

```
getKeyRandom()
```

**Value**

A string value holding the randomSeed name.

---

getLogDir	<i>getLogDir()</i>
-----------	--------------------

---

**Description**

Returns the value for log directory

**Usage**

```
getLogDir()
```

---

GetOutput	<i>Gets the output</i>
-----------	------------------------

---

**Description**

Returns the results of a model a data.frame from the last RUN. Should be used only if model replication is equal to 1, otherwise GetResults must be used.

**Usage**

```
GetOutput(e)
```

**Arguments**

e                    An engine object instance

**Value**

Returns a data.frame with output data

**Examples**

```
## Not run:  
d<- "C:/usr/models/your-model-directory"  
m<- Model(d)  
...  
data<- GetOutput(m)  
## End(Not run)
```

getOutputDir            *Gets output directory*

---

**Description**

Returns the value of module variable for storing the current output directory.

**Usage**

getOutputDir()

---

getpkgcores            *getpkgcores*

---

**Description**

Returns the maximum number of cores to be used in parallel computations

**Usage**

getpkgcores()

**Value**

The number of cores

---

getpkgdefaultcores    *getpkgdefaultcores*

---

**Description**

Provides the package default parallelism level which is 80% of total cores available

**Usage**

getpkgdefaultcores()

**Value**

Cores used by R/Repast



---

GetResults	<i>Returns the model results</i>
------------	----------------------------------

---

**Description**

Returns the model results

**Usage**

GetResults()

---

GetResultsParameters	<i>Gets the parameters</i>
----------------------	----------------------------

---

**Description**

Returns the current set of paramters used for the last model run.

**Usage**

GetResultsParameters()

**Value**

A data.frame with parameters of the model.

---

GetSimulationParameters	<i>Gets the simulation parameters</i>
-------------------------	---------------------------------------

---

**Description**

Returns a dataframe with the current set of input parameters for the last model run.

**Usage**

GetSimulationParameters(e)

**Arguments**

e                    An engine object instance

**Value**

A data frame with simulation parameters

GetSimulationParameterType  
*GetSimulationParameterType*

---

**Description**

Returns the declared parameter type.

**Usage**

```
GetSimulationParameterType(e, k)
```

**Arguments**

e	An instance of 'Engine' object
k	The parameter name

**Value**

The parameter type as string

---

GoToPreviousDir      *GoToPreviousDir*

---

**Description**

Returns to the saved work directory

**Usage**

```
GoToPreviousDir()
```

---

GoToWorkDir      *GoToWorkDir*

---

**Description**

Changes the current work directory saving the previous one which is used in [GoToPreviousDir](#). This function is called by [Easy.Setup](#)

**Usage**

```
GoToWorkDir()
```

---

hybrid.distance	<i>hybrid.distance</i>
-----------------	------------------------

---

**Description**

Calculates the distance between some value a reference target value. It is an hybrid distance because when the value falls within a reference range the distance is 0, otherwise the distance between the value and the reference value is calculated using the user provided distance function.

**Usage**

```
hybrid.distance(value, reference, FUN = AoE.NRMSD)
```

**Arguments**

value	The value which will be compared against the reference
reference	The reference value. It should be a list holding the value, the range of values.
FUN	The distance function. The default is the NRMSD

**Value**

The distance metric

---

hybrid.value	<i>hybrid.value</i>
--------------	---------------------

---

**Description**

A simple helper function for generating the input list for the function 'hybrid.distance'. This list must hold the value and a range centered over the value.

**Usage**

```
hybrid.value(value, distance)
```

**Arguments**

value	The reference value
distance	The distance interval.

**Value**

The list holding the value and the interval 'min — value — max'

jarfile                    *jarfile*

---

**Description**

The jarfile returns the full path to some jar file available inside rrpast package

**Usage**

```
jarfile(fjar)
```

**Arguments**

fjar                    The name of jar file

**Value**

The full path to jar file

---

jvm.enablejmx            *jvm.enablejmx*

---

**Description**

Enable jmx for the current R/rJava session

**Usage**

```
jvm.enablejmx()
```

**Details**

Configures the JMX subsystem for the current session of R/rJava. This function must be called before any other function which initializes r/Java such as [Easy.Setup](#) or [Model](#) otherwise it will have no effect.

**Examples**

```
## Not run:  
  jvm.enablejmx()  
## End(Not run)
```

---

jvm.getruntime	<i>jvm.getruntime</i>
----------------	-----------------------

---

**Description**

A wrapper for System.getRuntime()

**Usage**

```
jvm.getRuntime()
```

**Details**

A simple wrapper for System.getRuntime() java method

---

jvm.get_parameters	<i>jvm.get_parameters</i>
--------------------	---------------------------

---

**Description**

Returns the current java virtual machine parameters

**Usage**

```
jvm.get_parameters()
```

**Value**

A string with JVM parameters.

---

jvm.init	<i>Init R/JVM environment</i>
----------	-------------------------------

---

**Description**

Initialize rJava and repast environment with classpath. This function is called internally and it is not meant to be used directly.

**Usage**

```
jvm.init()
```

**Details**

The default parameters can be changed as needed calling the primitive `jvm.set_parameters` before instantiating the model engine.

**References**

[1] rJava: Low-Level R to Java Interface. Low-level interface to Java VM very much like `.C/.Call` and friends. Allows creation of objects, calling methods and accessing fields.

**Examples**

```
## Not run:
  jvm.init()
## End(Not run)
```

---

jvm.memory	<i>jvm.memory</i>
------------	-------------------

---

**Description**

JVM memory state

**Usage**

```
jvm.memory()
```

**Details**

Provides information about the memory used by the JVM subsystem

---

jvm.resetOut	<i>jvm.resetOut</i>
--------------	---------------------

---

**Description**

Reset the `System.out` filed value to console output

**Usage**

```
jvm.resetOut()
```

**Examples**

```
## Not run:
  jvm.resetOut()
## End(Not run)
```

---

jvm.runtimegc	<i>jvm.runtimegc</i>
---------------	----------------------

---

**Description**

A wrapper for `Runtime.gc()`

**Usage**

```
jvm.runtimegc()
```

**Details**

Forces the execution of the JVM garbage collector

---

jvm.setOut	<i>jvm.setOut</i>
------------	-------------------

---

**Description**

Set the `System.out` filed to a file

**Usage**

```
jvm.setOut(f)
```

**Arguments**

f	The output file name
---	----------------------

**Examples**

```
## Not run:  
jvm.setOut("/tmp/SysteOut.log")  
## End(Not run)
```

---

```
jvm.set_parameters      jvm.set_parameters
```

---

**Description**

Configures the jvm parameters

**Usage**

```
jvm.set_parameters(s)
```

**Arguments**

s                    The paramter string to be passed to the underlying JVM

**Details**

Set the underlying parameters for java virtual machine. The default values are "-server -Xms1024m -Xmx1024m". These defaults can be changed to fit the model requirements.

**Examples**

```
## Not run:
  jvm.set_parameters("-server -Xms512m -Xmx2048m")
## End(Not run)
```

---

```
lcontains      lcontains
```

---

**Description**

Cheks if a list contains a name

**Usage**

```
lcontains(l, n)
```

**Arguments**

l                    The list object  
n                    The item name

**Value**

Boolean TRUE if name is found on list



---

lget	<i>get</i>
------	------------

---

**Description**

Retrieve the value for a list item

**Usage**

```
lget(l, n)
```

**Arguments**

l	The list object
n	The item name

**Value**

The item value

---

Load	<i>The Scenario loader</i>
------	----------------------------

---

**Description**

Loads the model's scenario. This function must be called before running the model.

**Usage**

```
Load(e)
```

**Arguments**

e	An engine object instance
---	---------------------------

**Examples**

```
## Not run:  
d<- "C:/usr/models/your-model-directory"  
m<- Model(d)  
Load(m)  
## End(Not run)
```

---

`Logger.setLevelInfo`    *Set the log level to INFO*

---

**Description**

Configures the underlying logging system

**Usage**

`Logger.setLevelInfo()`

---

`Logger.setLevelWarning`  
                                   *Set the log level to WARNING*

---

**Description**

Configures the underlying logging system

**Usage**

`Logger.setLevelWarning()`

---

`Model`                            *The easy API for model initialization*

---

**Description**

Instantiate a repast model from the model dir without loading the scenario file.

**Usage**

`Model(modeldir = "", maxtime = 300, dataset = "none", load = FALSE)`

**Arguments**

<code>modeldir</code>	The installation directory of some repast model
<code>maxtime</code>	The total simulated time
<code>dataset</code>	The name of any model aggregate dataset
<code>load</code>	If true instantiate model and load scenario

**Details**

This is the entry point for model execution. Typically any model execution will start with this function which encapsulates all low level calls for model initialization. In order to perform simulations with repast from R code only Model and a few more function calls are required: [Load](#), [Run](#). Finally the output of model is managed with functions [GetResults](#) and [SaveSimulationData](#).

**Value**

Returns the instance of repast model

**References**

[1] North, M.J., N.T. Collier, and J.R. Vos, "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit," ACM Transactions on Modeling and Computer Simulation, Vol. 16, Issue 1, pp. 1-25, ACM, New York, New York, USA (January 2006).

**Examples**

```
## Not run:
  d<- "C:/usr/models/your-model-directory"
  m<- Model(d)
## End(Not run)
```

---

ParallelClose	<i>ParallelClose</i>
---------------	----------------------

---

**Description**

Finalize the parallel execution environment for R/Repast

**Usage**

```
ParallelClose()
```

---

ParallelInit	<i>ParallelInit</i>
--------------	---------------------

---

**Description**

Initialize the parallel execution environment for R/Repast

**Usage**

```
ParallelInit()
```

---

parallelize

*parallelize*


---

**Description**

Tells R/Repast to use multicore. Default is using just one core.

**Usage**

```
parallelize(v = NULL)
```

**Arguments**

v	A Boolean value telling if use multiple cores. When null just returns the current setting
---	---

**Value**

Boolean with current state

---

ParallelRunExperiment

*ParallelRunExperiment*


---

**Description**

Run the model multiple times for different parameters given by design matrix function parameter.

**Usage**

```
ParallelRunExperiment(modelldir, datasource, maxtime, r = 1, design, FUN,
  default = NULL)
```

**Arguments**

modelldir	The installation directory of some repast model
datasource	The name of any model aggregate dataset
maxtime	The total simulated time
r	The number of experiment replications
design	The desing matrix holding parameter sampling
FUN	THE calibration function.
default	The alternative values for parameters which should be kept fixed

**Details**

The FUN function must return zero for perfect fit and values greater than zero otherwise.

**Value**

A list with output and dataset

**Examples**

```
## Not run:
my.cost<- function(params, results) { # your best fit calculation, being 0 the best metric. }
d<- "/usr/models/your-model-directory"
f<- AddFactor(name="cyclePoint",min=40,max=90)
f<- AddFactor(factors=f, name="conjugationCost",min=1,max=80)
d<- AoE.LatinHypercube(factors=f)
v<- ParallelRunExperiment()
## End(Not run)
```

---

ParallelRun

*ParallelRun*


---

**Description**

Run simulations in parallel. This function executes the time steps of an instantiated model. The number of replications of model runs can be specified by the function parameter. The seed parameter may be omitted and will be generated internally. If provided, the seed collection, must contain the same number of r parameter.

**Usage**

```
ParallelRun(modeldir, datasource, maxtime, r = 1, seed = c(),
  design = NULL, default = NULL)
```

**Arguments**

modeldir	The installation directory of some repast model
datasource	The name of any model aggregate dataset
maxtime	The total simulated time
r	The number of experiment replications
seed	The random seed collection
design	The desing matrix holding parameter sampling
default	The alternative values for parameters which should be kept fixed

**Value**

The model output dataset

**Examples**

```
## Not run:  
md<- "/usr/models/your-model-directory"  
output<- ParallelRun(modeldir= md, maxtime = 360, dataset= ds, r=4)  
## End(Not run)
```

---

PB.close

*PB.close*

---

**Description**

Close the progress bar descriptor

**Usage**

PB.close()

---

PB.disable

*PB.disable*

---

**Description**

Disable the progress bar visualization

**Usage**

PB.disable()

---

PB.enable

*PB.enable*

---

**Description**

Enables the progress bar visualization

**Usage**

PB.enable()

---

PB.get	<i>PB.get</i>
--------	---------------

---

**Description**

Gets the the progress bar descriptor

**Usage**

PB.get()

---

PB.init	<i>PB.init</i>
---------	----------------

---

**Description**

Initialize progress bar for model execution.

**Usage**

PB.init(psets, replications)

**Arguments**

psets                   – The total number of paramter sets being simulated  
replications       – The number of replications per simulation round

---

PB.isEnabled	<i>PB.isEnabled</i>
--------------	---------------------

---

**Description**

Returns the global value indicating if progress bar is enabled.

**Usage**

PB.isEnabled()

**Value**

Boolean TRUE if progress bar must be shown

---

PB.pset	<i>PB.pset</i>
---------	----------------

---

**Description**

Update pset value

**Usage**

PB.pset(v)

**Arguments**

v	The current parameter set being simulated
---	---

---

PB.rnum	<i>PB.rnum</i>
---------	----------------

---

**Description**

Update run number value

**Usage**

PB.rnum(v)

**Arguments**

v	The current run number
---	------------------------

---

PB.set	<i>PB.set</i>
--------	---------------

---

**Description**

Ses the progress bar descriptor

**Usage**

PB.set(obj)

**Arguments**

obj	– The progress bar descriptor
-----	-------------------------------



---

PB.update

*PB.update*

---

### Description

Update progress bar

### Usage

```
PB.update(r = NULL)
```

### Arguments

r                    The current replication number

---

pick.fittest

*pick.fittest*

---

### Description

Choose the best solutions minimizing the objective function

### Usage

```
pick.fittest(out, goals = c(), n = 4)
```

### Arguments

out                    The output data set holding the values of goals  
goals                  The column names which must be used as goal  
n                        The number of solutions

### Value

The n rows holding the best results

---

Plot.Calibration	<i>Plot of calibration</i>
------------------	----------------------------

---

**Description**

Generate plot for parameter sets providing best fit

**Usage**

```
Plot.Calibration(obj, key, title = NULL)
```

**Arguments**

obj	An instance of calibration Object
key	The column name
title	Chart title, may be null

**Value**

The resulting ggplot2 plot object

---

Plot.Morris	<i>Plot of Morris output</i>
-------------	------------------------------

---

**Description**

Generate plot for Morris's screening method

**Usage**

```
Plot.Morris(obj, type, title = NULL)
```

**Arguments**

obj	An instance of Morris Object <a href="#">AoE.Morris</a>
type	The chart type ( $\mu \cdot \sigma$ or $\mu \cdot \sigma^2$ )
title	Chart title, may be null

**Value**

The resulting ggplot2 plot object

---

Plot.Sobol	<i>Plot of Sobol output</i>
------------	-----------------------------

---

**Description**

Generate plot for Sobol's GSA

**Usage**

```
Plot.Sobol(obj, type, title = NULL)
```

**Arguments**

obj	An instance of Sobol Object <a href="#">AoE.Sobol</a>
type	The chart type
title	Chart title, may be null

**Value**

The resulting ggplot2 plot object

---

Plot.Stability	<i>Plot stability of output</i>
----------------	---------------------------------

---

**Description**

Generate plot for visually access the stability of coefficient of variation as function of simulation sample size.

**Usage**

```
Plot.Stability(obj, title = NULL)
```

**Arguments**

obj	An instance of Morris Object <a href="#">AoE.Morris</a>
title	Chart title, may be null

**Value**

The resulting ggplot2 plot object

---

`Results.GetCharts`      *Results.GetCharts*

---

**Description**

Simplify the access to the charts member

**Usage**

`Results.GetCharts(obj)`

**Arguments**

`obj`                      An instance of the object returned by Easy methods

**Value**

The charts element inside results

---

`Results.GetExperiment`      *Results.GetExperiment*

---

**Description**

Simplify the access to the experiment member

**Usage**

`Results.GetExperiment(obj)`

**Arguments**

`obj`                      An instance of the object returned by Easy methods

**Value**

The experiment element inside results

---

Results.GetObject	<i>Results.GetObject</i>
-------------------	--------------------------

---

**Description**

Simplify the access to the object member

**Usage**

Results.GetObject(obj)

**Arguments**

obj	An instance of the object returned by Easy methods
-----	--

**Value**

The object element inside results

---

Run	<i>Run simulations</i>
-----	------------------------

---

**Description**

This function executes the time steps of an instantiated model. The number of replications of model runs can be specified by the function parameter. The seed parameter may be omitted and will be generated internally. If provided, the seed collection, must contain the same number of r parameter.

**Usage**

Run(e, r = 1, seed = c())

**Arguments**

e	An engine object instance
r	The number of experiment replications
seed	The random seed collection

**Value**

The model output dataset

**Examples**

```
## Not run:
d<- "C:/usr/models/your-model-directory"
m<- Model(d)
Load(m)
Run(m,r=2) # or Run(m,r=2,seed=c(1,2))
## End(Not run)
```

---

RunExperiment

*Run an experimental setup*


---

**Description**

Run the model multiple times for different parameters given by design matrix function parameter.

**Usage**

```
RunExperiment(e, r = 1, design, FUN)
```

**Arguments**

e	An engine object instance
r	The number of experiment replications
design	The desing matrix holding parameter sampling
FUN	THE calibration function.

**Details**

The FUN function must return zero for perfect fit and values greater than zero otherwise.

**Value**

A list with output and dataset

**Examples**

```
## Not run:
my.cost<- function(params, results) { # your best fit calculation, being 0 the best metric. }
d<- "c:/usr/models/your-model-directory"
m<- Model(d,dataset="ds::Output")
Load(m)
f<- AddFactor(name="cyclePoint",min=40,max=90)
f<- AddFactor(factors=f, name="conjugationCost",min=1,max=80)
d<- LatinHypercube(factors=f)
p<- GetSimulationParameters(e)
exp.design<- BuildParameterSet(d,p)
v<- RunExperiment(e,r=1,exp.design,my.cost)
## End(Not run)
```

---

SaveSimulationData	<i>Saving simulation output</i>
--------------------	---------------------------------

---

**Description**

Saves the simulation results of last call to Run(e) function.

**Usage**

```
SaveSimulationData(as = "csv", experiment = NULL)
```

**Arguments**

as	The desired output type, must be csv or xls
experiment	The experiment output

**Details**

The model must have been initialized or user must call setId explicitly.

**Value**

The id of saved data

---

SequenceItem	<i>SequenceItem</i>
--------------	---------------------

---

**Description**

Generate a sequence from min to max using an increment based on the number of elements in v

**Usage**

```
SequenceItem(v, min, max)
```

**Arguments**

v	A column of n x k design matrix
min	The lower boundary of range
max	The upper boundary of range

**Value**

A sequence between min and max value

---

setId	<i>Sets the model name</i>
-------	----------------------------

---

**Description**

Set the name of the model currently instantiated.

**Usage**

setId(s)

**Arguments**

s	The model name
---	----------------

---

setKeyRandom	<i>Sets Repast randomSeed name</i>
--------------	------------------------------------

---

**Description**

Configures a non-default value for Repast randomSeed parameter name.

**Usage**

setKeyRandom(k)

**Arguments**

k	The string with an alternative name for randomSeed
---	--

---

setOutputDir	<i>Sets output directory</i>
--------------	------------------------------

---

**Description**

Configure the desired directory to save model output data.

**Usage**

setOutputDir(s)

**Arguments**

s	The full path for output directory
---	------------------------------------



---

setpkgcores	<i>setpkgcores</i>
-------------	--------------------

---

**Description**

Configures the maximum number of cores to be used in parallel computations

**Usage**

```
setpkgcores(v)
```

**Arguments**

v	The number of cores
---	---------------------

---

SetResults	<i>Stores a data.frame</i>
------------	----------------------------

---

**Description**

Stores a data.frame

**Usage**

```
SetResults(d)
```

**Arguments**

d	A data frame containing one replication data
---	--

---

SetResultsParameters	<i>Sets the parameters</i>
----------------------	----------------------------

---

**Description**

Save the current set of parameters used for the last model run.

**Usage**

```
SetResultsParameters(d)
```

**Arguments**

d	A data.frame with parameter values
---	------------------------------------

SetSimulationParameter

*SetSimulationParameter*

---

### **Description**

Modify model's default parameter collection

### **Usage**

SetSimulationParameter(e, key, value)

### **Arguments**

e	An engine object instance
key	The parameter name
value	The parameter value

---

SetSimulationParameters

*Set parameters for running model*

---

### **Description**

Modify the repeat model parameters with values provided in parameter 'p' which is a data frame with just one row.

### **Usage**

SetSimulationParameters(e, p)

### **Arguments**

e	An engine object instance
p	A data frame with simulation parameters

---

ShowClassPath	<i>ShowClassPath</i>
---------------	----------------------

---

**Description**

Shows the current classpath

**Usage**

```
ShowClassPath()
```

**Value**

the current setting of JVM classpath

**Examples**

```
## Not run:  
  ShowClassPath()  
## End(Not run)
```

---

ShowCores	<i>ShowUsedCores</i>
-----------	----------------------

---

**Description**

Prints the number of cores used

**Usage**

```
ShowCores()
```

---

ShowModelPaths	<i>ShowModelPaths</i>
----------------	-----------------------

---

**Description**

Prints the paths. Shows the directories currently used to load model scenario and lib. The output of this function is informational only and can be used to check whether model data is being loaded properly from correct locations.

**Usage**

```
ShowModelPaths()
```

**Examples**

```
## Not run:
  ShowModelPaths()
## End(Not run)
```

---

UpdateDefaultParameters	<i>UpdateDefaultParameters</i>
-------------------------	--------------------------------

---

**Description**

Modify the value of the default parameters which should be kept fixed

**Usage**

```
UpdateDefaultParameters(e, p)
```

**Arguments**

e	An engine object instance
p	The collection of model fixed paramters to change

**Examples**

```
## Not run:
  d<- "C:/usr/models/your-model-directory"
  m<- Model(d)
  Load(m)

  p<- c(name1=value1, name2=2)
  UpdateDefaultParameters(m,p)
## End(Not run)
```

---

WrapperRun	<i>WrapperRun</i>
------------	-------------------

---

**Description**

Wrapper for the Run and ParallelRun functions

**Usage**

```
WrapperRun(modelldir, datasource, maxtime, r = 1, seed = c(),
  design = NULL, default = NULL, multi = TRUE)
```

**Arguments**

modelldir	The installation directory of some repast model
datasource	The name of any model aggregate dataset
maxtime	The total simulated time
r	The number of experiment replications
seed	The random seed collection
design	The desing matrix holding parameter sampling
default	The alternative values for parameters which should be kept fixed
multi	allows forcing single core execution, default is using multi-core

**Value**

The model output dataset

---

WrapperRunExperiment	<i>WrapperRunExperiment</i>
----------------------	-----------------------------

---

**Description**

Wrapper for the RunExperiment and ParallelRunExperiment functions

**Usage**

```
WrapperRunExperiment(modelldir, datasource, maxtime, r = 1, design, FUN,
  default = NULL)
```

**Arguments**

modeldir	The installation directory of some repast model
datasource	The name of any model aggregate dataset
maxtime	The total simulated time
r	The number of experiment replications
design	The desing matrix holding parameter sampling
FUN	The objective function.
default	The alternative values for parameters which should be kept fixed

**Value**

The model output dataset

# Index

AddFactor, 5  
AddFactor0, 5  
AddResults, 6  
AoE.Base, 6  
AoE.ColumnCoV, 7  
AoE.CoV, 7  
AoE.FullFactorial, 8  
AoE.GetMorrisOutput, 8  
AoE.LatinHypercube, 9  
AoE.MAE, 9  
AoE.Morris, 10, 58, 59  
AoE.NRMSD, 10  
AoE.RandomSampling, 11  
AoE.RMSD, 12  
AoE.Sobol, 12, 59  
AoE.Stability, 13  
ApplyFactorRange, 13  
  
BuildParameterSet, 14  
  
Calibration.GetMemberKeys, 14  
Calibration.GetMemberList, 15  
check.integration, 15  
check.scenario, 16  
ClearResults, 16  
col.sum, 17  
config.check, 17  
config.copypilib, 18  
config.scenario, 18  
createOutputDir, 19  
  
df2matrix, 19  
dffilterby, 20  
dfround, 20  
dfsumcol, 21  
  
Easy.Calibration, 21  
Easy.getChart, 22  
Easy.getPlot, 23  
Easy.Morris, 23  
  
Easy.Run, 24  
Easy.RunExperiment, 24  
Easy.Setup, 25, 42, 44  
Easy.ShowModelParameters, 25  
Easy.Sobol, 26  
Easy.Stability, 26  
Engine, 27  
Engine.endAt, 28  
Engine.Finish, 28  
Engine.getId, 28  
Engine.GetModelOutput, 29  
Engine.getParameter, 29  
Engine.getParameterAsDouble, 30  
Engine.getParameterAsNumber, 30  
Engine.getParameterAsString, 31  
Engine.getParameterNames, 31  
Engine.getParameterType, 32  
Engine.LoadModel, 32  
Engine.resetModelOutput, 33  
Engine.RunModel, 33  
Engine.SetAggregateDataSet, 33  
Engine.setParameter, 34  
enginestats.calls, 35  
enginestats.reset, 35  
  
getExperimentDataset, 35  
getExperimentOutput, 36  
getExperimentParamSet, 36  
GetFactorLevels, 37  
GetFactorsSize, 38  
getId, 38  
getKeyRandom, 38  
getLogDir, 39  
GetOutput, 39  
getOutputDir, 40  
getpkgcores, 40  
getpkgdefaultcores, 40  
GetResults, 41, 51  
GetResultsParameters, 41  
GetSimulationParameters, 41

GetSimulationParameterType, 42  
GoToPreviousDir, 42, 42  
GoToWorkDir, 42

hybrid.distance, 43  
hybrid.value, 43

jarfile, 44  
jvm.enablejmx, 44  
jvm.get\_parameters, 45  
jvm.getruntime, 45  
jvm.init, 45  
jvm.memory, 46  
jvm.resetOut, 46  
jvm.runtimegc, 47  
jvm.set\_parameters, 46, 48  
jvm.setOut, 47

lcontains, 48  
lget, 49  
Load, 49, 51  
Logger.setLevelInfo, 50  
Logger.setLevelWarning, 50

Model, 44, 50

ParallelClose, 51  
ParallelInit, 51  
parallelize, 52  
ParallelRunExperiment, 52  
ParallelRun, 53  
PB.close, 54  
PB.disable, 54  
PB.enable, 54  
PB.get, 55  
PB.init, 55  
PB.isEnabled, 55  
PB.pset, 56  
PB.rnum, 56  
PB.set, 56  
PB.update, 57  
pick.fittest, 57  
Plot.Calibration, 58  
Plot.Morris, 58  
Plot.Sobol, 59  
Plot.Stability, 59

Results.GetCharts, 60  
Results.GetExperiment, 60  
Results.GetObject, 61  
Run, 51, 61  
RunExperiment, 36, 62

SaveSimulationData, 51, 63  
SequenceItem, 63  
setId, 64  
setKeyRandom, 64  
setOutputDir, 64  
setpkgcores, 65  
SetResults, 65  
SetResultsParameters, 65  
SetSimulationParameter, 66  
SetSimulationParameters, 66  
ShowClassPath, 67  
ShowCores, 67  
ShowModelPaths, 68

UpdateDefaultParameters, 68

WrapperRun, 69  
WrapperRunExperiment, 69