

Package ‘sybil’

July 19, 2017

Type Package

Title Efficient Constrained Based Modelling

Version 2.0.4

Date 2017-07-18

Maintainer C. Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

Depends R (>= 2.14.2), Matrix, lattice

Imports methods

Suggests glpkAPI (>= 1.2.8), cplexAPI (>= 1.2.4), clpAPI (>= 1.2.4),
lpSolveAPI (>= 5.5.2.0), parallel, grid

URL [http://www.cs.hhu.de/lehrstuehle-und-arbeitsgruppen/
computational-cell-biology/software/sybil.html](http://www.cs.hhu.de/lehrstuehle-und-arbeitsgruppen/computational-cell-biology/software/sybil.html)

Description This Systems Biology Package (Gelius-Dietrich et. al. (2012) <doi:10.1186/1752-0509-7-125>) implements algorithms for constraint based analyses of metabolic networks, e.g. flux-balance analysis (FBA), minimization of metabolic adjustment (MOMA), regulatory on/off minimization (ROOM), robustness analysis and flux variability analysis. The package is easily extendable for additional algorithms. Most of the current LP/MILP solvers are supported via additional packages.

LazyLoad yes

License GPL-3 | file LICENSE

Collate generics.R validmodelorg.R validoptsol.R validreactId.R
validreactId_Exch.R validsysBiolAlg.R addAlgorithm.R
addExchReact.R addReact.R addSolver.R blockedReact.R
bracket_pairs.R ceilValues.R changeBounds.R changeGPR.R
changeObjFunc.R checkAlgorithm.R checkDefaultMethod.R
checkEmptyField.R checkReactId.R check_brackets.R
createReactionString.R deadEndMetabolite.R doInRound.R
doubleFluxDel.R doubleGeneDel.R doubleReact.R editEnvir.R
findExchReact.R floorValues.R fluxVar.R geneDel.R
geneDeletion.R generateFluxdels.R generateModKey.R generateWT.R
getsybilenv.R makeLPcompatible.R mod2irrev.R modelorg2ExPA.R
modelorg2text.R modelorg2tsv.R multiDel.R oneFluxDel.R
oneGeneDel.R onlyChangeGPR.R onlyCheckGPR.R optObj_basicfunc.R

optObj_lpSolveAPIcompat.R optimizer.R parseBoolean.R phpp.R
 ppProcessing.R prepareSubSysMatrix.R printLogComment.R
 printNamedList.R progress.R promptSysBiolAlg.R recodeMatrix.R
 readTEXTmod.R readTSVmod.R reassignFwBwMatch.R rmReact.R
 robAna.R settings.R singletonMetabolite.R sybilStack.R ypd.R
 zzz.R modelorgClass.R modelorg_irrevClass.R optObj_pointer.R
 optObjClass.R optObj_clpAPIClass.R optObj_cplexAPIClass.R
 optObj_glpkAPIClass.R optObj_lpSolveAPIClass.R
 sybilErrorClass.R ppProcClass.R netFluxClass.R
 fluxDistributionClass.R reactIdClass.R reactId_ExchClass.R
 optsolClass.R optsol_blockedReactClass.R
 optsol_optimizeProbClass.R optsol_fluxVarClass.R
 optsol_fluxdelClass.R optsol_robAnaClass.R optsol_phppClass.R
 optsol_genedelClass.R checksolClass.R summaryOptsolClass.R
 sysBiolAlgClass.R sysBiolAlg_fbaClass.R
 sysBiolAlg_fbaEasyConstraintClass.R sysBiolAlg_fvClass.R
 sysBiolAlg_lmomaClass.R sysBiolAlg_momaClass.R
 sysBiolAlg_mtfClass.R sysBiolAlg_mtfEasyConstraintClass.R
 sysBiolAlg_roomClass.R sybilLogClass.R upgradeModelorg.R

Author C. Jonathan Fritzscheier [cre, ctb],
 Gabriel Gelius-Dietrich [aut],
 Rajen Piernikarczyk [ctb],
 Marc Andre Daxer [ctb],
 Benjamin Braasch [ctb],
 Abdelmoneim Desouki [ctb],
 Martin J. Lercher [ctb]

NeedsCompilation no

Repository CRAN

Date/Publication 2017-07-19 16:22:47 UTC

R topics documented:

sybil-package	5
addAlgorithm	7
addCols-methods	7
addColsToProb-methods	8
addExchReact	10
addReact	11
addRows-methods	13
addRowsCols-methods	14
addRowsToProb-methods	15
addSolver	17
applyChanges-methods	18
backupProb-methods	20
blockedReact	21
changeBounds	22

changeColsBnds-methods	23
changeColsBndsObjCoefs-methods	24
changeGPR	25
changeMatrixRow-methods	26
changeObjCoefs-methods	27
changeObjFunc	28
changeRowsBnds-methods	29
changeUptake-methods	30
checkAlgorithm	31
checkDefaultMethod	31
checkOptSol-methods	33
checkReactId	34
checksol-class	35
checkVersion-methods	36
deadEndMetabolites-methods	37
delProb-methods	38
doubleFluxDel	39
doubleGeneDel	40
doubleReact	42
Ec_core	43
editEnvir	44
findExchReact	45
fluxDistribution-class	46
fluxVar	47
geneDel	48
geneDeletion	49
getColPrim-methods	51
getColsLowBnds-methods	52
getColsNames-methods	53
getColsUppBnds-methods	54
getFluxDist-methods	55
getNumCols-methods	56
getNumNnz-methods	57
getNumRows-methods	58
getObjCoefs-methods	59
getObjDir-methods	60
getObjVal-methods	61
getRedCosts-methods	62
getRowsLowBnds-methods	63
getRowsNames-methods	64
getRowsUppBnds-methods	65
getSolStat-methods	66
getSolverParm-methods	67
getsybilenv	68
initProb-methods	69
loadLPprob-methods	70
loadQobj-methods	75
makeOptsolMO	76

mod2irrev	76
modelorg-class	77
modelorg2ExPA	81
modelorg2tsv	82
modelorg_irrev-class	86
multiDel	87
netFlux-class	88
oneFluxDel	89
oneGeneDel	91
onlyChangeGPR	92
onlyCheckGPR	93
optimizeProb-methods	94
optimizer	99
optObj	102
optObj-class	103
optObj_clpAPI-class	106
optObj_cplexAPI-class	107
optObj_glpkAPI-class	108
optObj_lpSolveAPI-class	109
optsol-class	110
optsol_blockedReact-class	113
optsol_fluxdel-class	114
optsol_fluxVar-class	117
optsol_genedel-class	119
optsol_optimizeProb-class	121
optsol_phpp-class	122
optsol_robAna-class	124
phpp	126
ppProc-class	128
printMetabolite-methods	129
printReaction-methods	130
promptSysBiolAlg	131
reactId-class	132
reactId_Exch-class	134
readProb-methods	136
readTSVmod	137
resetChanges-methods	144
rmReact	145
robAna	146
scaleProb-methods	147
sensitivityAnalysis-methods	148
setColsNames-methods	149
setObjDir-methods	150
setRhsZero-methods	152
setRowsNames-methods	153
setSolverParm-methods	154
shrinkMatrix-methods	155
singletonMetabolites-methods	156

solveLp-methods	157
summaryOptsol	158
summaryOptsol-class	159
sybil-deprecated	161
sybilError-class	162
sybilLog-class	163
sybilStack	165
SYBIL_SETTINGS	167
sysBiolAlg	169
sysBiolAlg-class	171
sysBiolAlg_fba-class	174
sysBiolAlg_fbaEasyConstraint-class	176
sysBiolAlg_fv-class	178
sysBiolAlg_lmoma-class	181
sysBiolAlg_moma-class	184
sysBiolAlg_mtf-class	186
sysBiolAlg_room-class	189
upgradeModelorg	192
writeProb-methods	192
ypd	194

Index**197**

sybil-package

*sybil – Efficient Constrained Based Modelling in R***Description**

The package **sybil** is a collection of functions designed for in silico analysis—in particular constrained based analysis—of metabolic networks.

Details

The package `sybil` is designed to read metabolic networks from csv files. This is done by the function `readTSVmod`. The function returns an object of the class `modelorg`.

Read csv files (example files included):

```
mpath <- system.file(package = "sybil", "extdata")
model <- readTSVmod(prefix = "Ec_core",
                    fpath = mpath, quote = "\\")
```

Perform flux balance analysis (FBA):

```
ec_f <- optimizeProb(model)
```

Perform single gene deletion analysis:

```
ec_g <- oneGeneDel(model)
```

Plot the values of the objective function after optimization in a histogram:

```
plot(ec_g)
```

Perform flux variability analysis:

```
ec_v <- fluxVar(model)
```

Plot the result:

```
plot(ec_v)
```

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

References

Gelius-Dietrich, G., Desouki, A. A., Fritzemeier, C. J., and Lercher, M. J. (2013). sybil – Efficient constraint-based modelling in R. *BMC Systems Biology* **7**, 125.

The BiGG database <http://bigg.ucsd.edu/>.

Schellenberger, J., Park, J. O., Conrad, T. C., and Palsson, B. Ø., (2010) BiGG: a Biochemical Genetic and Genomic knowledgebase of large scale metabolic reconstructions. *BMC Bioinformatics* **11**, 213.

The openCOBRA project <https://opencobra.github.io/>.

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

See Also

Package **sybilSBML** and there the function `readSBMLmod` to read metabolic models written in SBML language.

Examples

```
data(Ec_core)
Ec_ofd <- oneGeneDel(Ec_core)
plot(Ec_ofd)
```

addAlgorithm	<i>Add a New Algorithm Name to sybil</i>
--------------	--

Description

Certain simulations can be run using different algorithms. For example, genetic perturbations can be studied with FBA, MOMA or the like. With this function you can add a new algorithm to an existing kind of simulation.

Usage

```
addAlgorithm(alg, purpose)
```

Arguments

alg	A single character string containing the name of the new algorithm.
purpose	Purpose of the new algorithm.

Value

Returns NULL invisibly.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkAlgorithm](#), [getsybilenv](#)

addCols-methods	<i>Add Columns to an Optimization Problem</i>
-----------------	---

Description

Add columns to an optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'  
addCols(lp, ncols)  
  
## S4 method for signature 'optObj_cplexAPI,numeric'  
addCols(lp, ncols)  
  
## S4 method for signature 'optObj_glpkAPI,numeric'  
addCols(lp, ncols)  
  
## S4 method for signature 'optObj_lpSolveAPI,numeric'  
addCols(lp, ncols)
```

Arguments

lp An object extending class `optObj`.
ncols Number of columns (variables) to add to the problem object.

Methods

signature(lp = "optObj_clpAPI", ncols = "numeric") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI", ncols = "numeric") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI", ncols = "numeric") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI", ncols = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>
Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

addColsToProb-methods *Add New Columns (Variables) to an Optimization Problem*

Description

Add new columns (variables) to an optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
addColsToProb(lp, j, obj, lb, ub, rind, nzval)  
  
## S4 method for signature 'optObj_cplexAPI'  
addColsToProb(lp, j, obj, lb, ub, rind, nzval)  
  
## S4 method for signature 'optObj_glpkAPI'  
addColsToProb(lp, j, obj, lb, ub, rind, nzval)  
  
## S4 method for signature 'optObj_lpSolveAPI'  
addColsToProb(lp, j, obj, lb, ub, rind, nzval)
```

Arguments

lp	An object extending class <code>optObj</code> .
j	A numeric vector containing the new column indices.
obj	A numeric vector containing the objective coefficients of the new variables.
lb	A numeric vector containing the lower bounds of the new variables.
ub	A numeric vector containing the upper bounds of the new variables.
rind	A list containing the row indices of the new non-zero elements.
nzval	A list containing the new non-zero elements.

Methods

```
signature(lp = "optObj_clpAPI") method to use with package optObj_clpAPI.  
signature(lp = "optObj_cplexAPI") method to use with package optObj_cplexAPI.  
signature(lp = "optObj_glpkAPI") method to use with package optObj_glpkAPI.  
signature(lp = "optObj_lpSolveAPI") method to use with package optObj_lpSolveAPI.
```

Note

Arguments j, obj, lb, lu, rind and nzval must have the same length.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

addExchReact *Add Exchange Reactions to a Model*

Description

The function `addExchReact` adds exchange reactions for a set of metabolites to a metabolic model.

Usage

```
addExchReact(model, met, lb, ub)
```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>met</code>	A vector of character strings containing the metabolite id's to add exchange reactions for.
<code>lb</code>	A vector of numeric values of the same length as <code>met</code> containing the lower bounds for the exchange reactions. Default: <code>rep(0, length(met))</code> .
<code>ub</code>	A vector of numeric values of the same length as <code>met</code> containing the upper bounds for the exchange reactions. Default: <code>rep(SYBIL_SETTINGS("MAXIMUM"), length(met))</code> .

Details

If `lb[i] < 0`, the exchange reaction for the metabolite in `met[i]` is considered to be reversible, otherwise irreversible. A reaction id is generated for each exchange reaction by prepending the metabolite id's with the string "Ex_".

Value

An object of class `modelorg`

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

See Also

[modelorg](#) and [addReact](#)

Examples

```
# add exchange reactions (allowing input) for the metabolites
# malate and oxalacetate
data(Ec_core)
mod <- addExchReact(Ec_core,
                    met = c("mal_L[c]", "oaa[c]"),
                    lb = c(-20, -20))
findExchReact(mod)
```

 addReact

Add/Change Reactions in a Model

Description

The function `addReact` adds one reaction to a metabolic model, or changes one reaction in a metabolic model.

Usage

```
addReact(model,
          id,
          met,
          Scoef,
          reversible = FALSE,
          lb = 0,
          ub = SYBIL_SETTINGS("MAXIMUM"),
          obj = 0,
          subSystem = NA,
          gprAssoc = NA,
          reactName = NA,
          metName = NA,
          metComp = NA)
```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>id</code>	A single character string containing a reaction id (see details below).
<code>met</code>	A vector of character strings containing the metabolite id's used in the reaction given in <code>Scoef</code> .
<code>Scoef</code>	A numeric vector of the same length as <code>met</code> of stoichiometric coefficients for the metabolites in <code>met</code> . The value in <code>Scoef[i]</code> is the stoichiometric coefficient of the metabolite in <code>met[i]</code> .

reversible	A Boolean value, indicating if the reaction is reversible or not. Default: FALSE.
lb	A single numeric value giving the lower bound of the reaction. Default: 0.
ub	A single numeric value giving the upper bound of the reaction. Default: SYBIL_SETTINGS("MAXIMUM").
obj	A single numeric value giving the objective coefficient of the reaction. Default: 0.
subSystem	A vector of character strings containing the sub systems to which the reaction belongs. All values must be available in <code>subSys(model)</code> . If NA, the reaction will not be associated to any sub system. Default: NA.
gprAssoc	A single character string giving the gpr association for the reaction. If NA, no gpr association is created. Default: NA.
reactName	A single character string giving the name for the reaction. If NA, the value of argument <code>id</code> is used. Default: NA.
metName	A vector of character strings of the same length as <code>met</code> containing the metabolites names for the metabolites given in argument <code>met</code> . If set to NA, the metabolite id's are used. Default: NA.
metComp	A vector of character strings or integers of the same length as <code>met</code> containing a compartment name (as in <code>mod_compart(model)</code>) or an index pointing to a value in <code>mod_compart(model)</code> (as in <code>met_comp(model)</code>). If NA, the metabolites will not be associated to any compartment. Default: NA.

Details

The function `addReact` can be used to add reactions and/or metabolites to a given metabolic model, or to change parameters of a reaction already present in a given metabolic model. If the reaction `id` in argument `id` is already present in the given model, this reaction will be changed, no new column will be added to the stoichiometric matrix. If any of the metabolite id's of argument `met` are not present in the model, they will be added (new rows in the stoichiometric matrix will be added).

Arguments `subSystem`, `gprAssoc` and `reactName` are only used, if a new reaction is added to the model (if `id` is not in `react_id(model)`, exact matching is used).

Value

An object of class `modelorg`, or `modelorg_irrev`, if `model` is of class `modelorg_irrev`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmadian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

See Also

[modelorg](#) and [rmReact](#)

Examples

```
data(Ec_core)

# add reaction A + 2 B <-> C to the model
modelNew <- addReact(Ec_core, id="newReact", met=c("A", "B", "C"),
  Scoef=c(-1, -2, 1), reversible=TRUE,
  lb=-1000, ub=1000, obj=0)

# view the new reaction
shrinkMatrix(modelNew, j="newReact")
```

addRows-methods

Add Rows to an Optimization Problem

Description

Add rows to an optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'
addRows(lp, nrows)

## S4 method for signature 'optObj_cplexAPI,numeric'
addRows(lp, nrows)

## S4 method for signature 'optObj_glpkAPI,numeric'
addRows(lp, nrows)

## S4 method for signature 'optObj_lpSolveAPI,numeric'
addRows(lp, nrows)
```

Arguments

lp An object extending class `optObj`.
 nrows Number of rows (constraints) to add to the problem object.

Methods

signature(lp = "optObj_clpAPI", nrows = "numeric") method to use with package **optObj_clpAPI**.
 signature(lp = "optObj_cplexAPI", nrows = "numeric") method to use with package **optObj_cplexAPI**.
 signature(lp = "optObj_glpkAPI", nrows = "numeric") method to use with package **optObj_glpkAPI**.
 signature(lp = "optObj_lpSolveAPI", nrows = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

addRowsCols-methods *Add Rows and Columns to an Optimization Problem*

Description

Add rows and columns to an optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric,numeric'
addRowsCols(lp, nrows, ncols)
```

```
## S4 method for signature 'optObj_cplexAPI,numeric,numeric'
addRowsCols(lp, nrows, ncols)
```

```
## S4 method for signature 'optObj_glpkAPI,numeric,numeric'
addRowsCols(lp, nrows, ncols)
```

```
## S4 method for signature 'optObj_lpSolveAPI,numeric,numeric'
addRowsCols(lp, nrows, ncols)
```

Arguments

lp	An object extending class <code>optObj</code> .
nrows	Number of rows (constraints) to add to the problem object.
ncols	Number of columns (variables) to add to the problem object.

Methods

signature(lp = "optObj_clpAPI", nrows = "numeric", ncols = "numeric") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI", nrows = "numeric", ncols = "numeric") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI", nrows = "numeric", ncols = "numeric") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI", nrows = "numeric", ncols = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

addRowsToProb-methods *Add New Rows (Constraints) to an Optimization Problem*

Description

Add new rows (constraints) to an optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'
addRowsToProb(lp, i, type, lb, ub, cind, nzval, rnames = NULL)
```

```
## S4 method for signature 'optObj_cplexAPI'
addRowsToProb(lp, i, type, lb, ub, cind, nzval, rnames = NULL)
```

```
## S4 method for signature 'optObj_glpkAPI'
addRowsToProb(lp, i, type, lb, ub, cind, nzval, rnames = NULL)
```

```
## S4 method for signature 'optObj_lpSolveAPI'
addRowsToProb(lp, i, type, lb, ub, cind, nzval, rnames = NULL)
```

Arguments

lp	An object extending class <code>optObj</code> .
i	A numeric vector containing the new row indices.
type	A character vector giving the constraint type: "F": free constraint (optObj_glpkAPI only), "L": \geq (lower bound), "U": \leq (upper bound) or "D": $lb \leq r \leq ub$ (double bound) or "E": = (equality). If <code>type[k]</code> is not F, "L", "U", "D" or "E", the value of <code>type[k]</code> will be set to "E".
lb	A numeric vector containing the lower bound of the new constraints.
ub	A numeric vector containing the upper bound of the new constraints.
cind	A list containing the column indices of the new non-zero elements.
nzval	A list containing the new non-zero elements.
rnames	A character vector containing names for the new rows/constraints. Default: NULL.

Methods

`signature(lp = "optObj_clpAPI")` method to use with package **optObj_clpAPI**. Parameter `rnames` is currently unused.

`signature(lp = "optObj_cplexAPI")` method to use with package **optObj_cplexAPI**.

`signature(lp = "optObj_glpkAPI")` method to use with package **optObj_glpkAPI**.

`signature(lp = "optObj_lpSolveAPI")` method to use with package **optObj_lpSolveAPI**.

Note

Arguments `i`, `type`, `lb`, `cind`, `nzval` and `rnames` (if not NULL) must have the same length.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

`addSolver`*Add a New Mathematical Programming Solver to sybil*

Description

Make a new mathematical programming solver available to sybil via the `SYBIL_SETTINGS` command.

Usage

```
addSolver(solver, method, probType)
```

Arguments

<code>solver</code>	A single character string giving the name of the desired solver.
<code>method</code>	A character vector of algorithms supported by the solver given in <code>solver</code> .
<code>probType</code>	A list of the same length as <code>method</code> containing a vector of character strings for each method which types of problems can be solved with that method: <code>method[i]</code> of <code>solver</code> can solve problems of type <code>probType[[i]]</code> . Problem types could be "lp": linear programming, "mip": mixed integer programming or "qp": quadratic programming.

Details

The parameters to the algorithms given in `method` are set to NA, which means, the default parameters of the solver software will be used. If a solver already exists, an error message will be given.

Value

The function returns NULL invisibly.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

[SYBIL_SETTINGS](#)

applyChanges-methods *Generic Function to Apply Changes to Objects of Class sysBiolAlg*

Description

Use method `applyChanges` to apply changes in objects of class `sysBiolAlg`. Changes can be coefficients of the objective function, variable bounds or the optimization direction.

Usage

```
## S4 method for signature 'sysBiolAlg'
applyChanges(object, del, obj, ld,
             react = NULL,
             lb = NULL,
             ub = NULL,
             obj_coef = NULL,
             fldind = TRUE,
             lpdir = NULL)

## S4 method for signature 'sysBiolAlg_room'
applyChanges(object, del, obj, ld,
             react = NULL,
             lb = NULL,
             ub = NULL,
             obj_coef = NULL,
             fldind = TRUE,
             lpdir = NULL)
```

Arguments

<code>object</code>	An object of class <code>sysBiolAlg</code> .
<code>del</code>	A logical value indicating whether variable bounds should be altered or not.
<code>obj</code>	A logical value indicating whether objective coefficients should be altered or not.
<code>ld</code>	A logical value indicating whether the direction of optimization should be altered or not.
<code>react</code>	A numeric vector containing indices to reactions which should be changed (in terms of variable bounds or objective coefficients). Default: NULL.
<code>lb</code>	Numeric vector of the same length as <code>react</code> , containing the new lower variable bounds. Default: NULL.
<code>ub</code>	Numeric vector of the same length as <code>react</code> , containing the new upper variable bounds. Default: NULL.

obj_coef	Numeric vector of the same length as react, containing the new objective coefficients. Default: NULL.
fldind	Boolean value. If set to TRUE, (default) indices in "react" are used only for reactions. If set to FALSE, indices in "react" are used for all variables during optimization, e.g. also for additional variables introduced by the <code>mtf</code> algorithm. Currently unused by class <code>sysBioAlg_room</code> . Default: TRUE.
lpdir	A single character value indicating the new direction of optimization. Default: NULL.

Value

Returns a list containing the original values in order to undo the changes with `resetChanges`:

fi	A numeric vector containing variable id's to apply changes to.
lb	A numeric vector of the same length as react containing the original variable lower bounds.
ub	A numeric vector of the same length as react containing the original variable upper bounds.
obj_coef	A numeric vector of the same length as react containing the original objective coefficients.
lpdir	A single character value giving the original optimization direction.
ri	A numeric vector of the same length as react containing row indices of the stoichiometric matrix required to apply changes in variable bounds when algorithm "room" is used. (only used by the <code>sysBioAlg_room</code> method).
ci	A numeric vector of the same length as react containing column indices of the stoichiometric matrix required to apply changes in variable bounds when algorithm "room" is used. (only used by the <code>sysBioAlg_room</code> method).

Methods

`signature(object = "sysBioAlg")` Method used with objects extending class `sysBioAlg`

`signature(object = "sysBioAlg_room")` Method used with objects of class `sysBioAlg_room`

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Class `sysBioAlg` and `resetChanges`

backupProb-methods *Copies a Problem Object to a New Problem Object*

Description

Copies a problem object into a new problem object.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
backupProb(lp)  
  
## S4 method for signature 'optObj_cplexAPI'  
backupProb(lp)  
  
## S4 method for signature 'optObj_glpkAPI'  
backupProb(lp)  
  
## S4 method for signature 'optObj_lpSolveAPI'  
backupProb(lp)
```

Arguments

lp An object extending class `optObj`.

Value

An object of the same class as given in argument `lp` (extending class `optObj`).

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**. The new problem object will be in the same CPLEX environment like the original one.
signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**. Building a new problem object will reset all parameters to their default. After backing up, set all parameters which are not at their default values again.
signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

blockedReact	<i>Find Blocked Reactions in a Metabolic Network</i>
--------------	--

Description

A blocked Reaction in a metabolic network can not be used by the network, given the stiochiometric matrix of the network and a set of input and output fluxes.

Usage

```
blockedReact(model,
             tol = SYBIL_SETTINGS("TOLERANCE"),
             exex = TRUE,
             fld = FALSE,
             retOptSol = FALSE,
             verboseMode = 2,
             ...)
```

Arguments

model	An object of class modelorg .
tol	Tolerance value. Default: SYBIL_SETTINGS("TOLERANCE").
exex	Boolean, if set to TRUE, exchange reactions found by findExchReact are excluded from the analysis. Default: TRUE.
fld	Boolean. Save the resulting flux distributions. Default: FALSE
retOptSol	Boolean. Return an object of class optsol_blockedReact or just a list containing the results. Default: FALSE.
verboseMode	An integer value indicating the amount of output to stdout: 0: nothing, 1: status messages, 2: like 1 plus a progress indicator. Default: 2.
...	Further arguments passed to sysBiolAlg . Argument solverParm is a good candidate.

Details

A reaction i is considered to be 'blocked', if its calculated reaction rate v_i is $-tol < v_i < tol$. Reaction rates are calculated via linear optimization: maximizing and minimizing each reaction rate. If the difference of the maximum and the minimum is not larger than tol , that particular reaction is blocked, given the current side conditions (exchange fluxes).

Value

If argument `retOptSol` is set to `TRUE`, an object of class `optsol_blockedReact` is returned, otherwise a logical vector with length equal to the number of reactions of the network. If element `i` equals `TRUE`, reaction `i` is blocked.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

`modelorg`, `optsol_blockedReact` and `SYBIL_SETTINGS`.

changeBounds

Change Variable Bounds in a Metabolic Network

Description

The function changes the upper and/or lower bounds of a given metabolic network model to new values.

Usage

```
changeBounds(model, react, lb = NULL, ub = NULL)
```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>react</code>	An object of class <code>reactId</code> , character or integer. Specifies the fluxes (variables) for which to change the upper and/or lower bounds.
<code>lb</code>	Numeric vector giving the lower bounds for the fluxes mentioned in <code>react</code> . If missing, lower bounds are set to zero. If <code>lb</code> has a length of 1, the value of <code>lb</code> will be used for all reactions in <code>react</code> .
<code>ub</code>	Numeric vector giving the upper bounds for the fluxes mentioned in <code>react</code> . If missing, upper bounds are set to zero. If <code>ub</code> has a length of 1, the value of <code>ub</code> will be used for all reactions in <code>react</code> .

Details

The argument `react` will be evaluated by the function `checkReactId`.

Value

Returns the given model (an object of the same class as the argument `lpmodel`) containing the new objective function.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

[checkReactId](#)

Examples

```
## change the E.coli core model to lactate input:
data(Ec_core)
Ec_new <- changeBounds(Ec_core,
                       c("EX_glc", "EX_lac"),
                       lb = c(0, -20), ub = 1000)
```

changeColsBnds-methods

Change Column (Variable) Bounds in the Optimization Problem

Description

Change column (variable) bounds in the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'
changeColsBnds(lp, j, lb, ub)

## S4 method for signature 'optObj_cplexAPI'
changeColsBnds(lp, j, lb, ub)

## S4 method for signature 'optObj_glpkAPI'
changeColsBnds(lp, j, lb, ub)

## S4 method for signature 'optObj_lpSolveAPI'
changeColsBnds(lp, j, lb, ub)
```

Arguments

lp	An object extending class <code>optObj</code> .
j	A numeric vector containing the column indices of the variables to change.
lb	A numeric vector of the same length as j containing the lower bounds of the variables to change.
ub	A numeric vector of the same length as j containing the upper bounds of the variables to change.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

changeColsBndsObjCoefs-methods

Change Column (Variable) Bounds and Objective Coefficients in the Optimization Problem

Description

Change column (variable) bounds and objective coefficients in the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'
changeColsBndsObjCoefs(lp, j, lb, ub, obj_coef)
```

```
## S4 method for signature 'optObj_cplexAPI'
changeColsBndsObjCoefs(lp, j, lb, ub, obj_coef)
```

```
## S4 method for signature 'optObj_glpkAPI'
changeColsBndsObjCoefs(lp, j, lb, ub, obj_coef)
```

```
## S4 method for signature 'optObj_lpSolveAPI'
changeColsBndsObjCoefs(lp, j, lb, ub, obj_coef)
```

Arguments

lp	An object extending class optObj .
j	A numeric vector containing the column indices of the variables to change.
lb	A numeric vector of the same length as j containing the lower bounds of the variables to change.

ub	A numeric vector of the same length as j containing the upper bounds of the variables to change.
obj_coef	A numeric vector of the same length as j containing the objective coefficients of the variables to change.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

changeGPR

Check and Change the GPR Rules

Description

Checks and Changes the GPR Rules for the chosen reactions

Usage

```
changeGPR(model, react, gprRules = "logicalExpression", verboseMode = 1)
```

Arguments

model	An object of class modelorg
react	An object of class reactId , a numeric vector, or a character vector containing reaction id's.
gprRules	character: contains logical expressions.
verboseMode	integer: verbosity level.

Details

The function changes the expressions for the chosen reactions.

The function stops if any logic expressions is not correct. Then the changes are executed.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

changeMatrixRow-methods

Change a Row in the Constraint Matrix of the Optimization Problem

Description

Change a row in the constraint matrix of the optimization problem.

Usage

```
## S4 method for signature 'optObj_cplexAPI'
changeMatrixRow(lp, i, j, val)
```

```
## S4 method for signature 'optObj_glpkAPI'
changeMatrixRow(lp, i, j, val)
```

```
## S4 method for signature 'optObj_lpSolveAPI'
changeMatrixRow(lp, i, j, val)
```

Arguments

lp	An object extending class optObj .
i	A single numeric value giving the row index of the constraint matrix to change.
j	A numeric vector containing the column indices of the new non-zero elements.
val	A numeric vector of the same length as j containing the new non-zero elements.

Methods

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**. Only the columns given in argument j will be changed. All other columns stay the same.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**. The row given in argument i will be reset completely.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**. The row given in argument i will be reset completely.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

changeObjCoefs-methods

Change Column (Variable) Objective Coefficients in the Optimization Problem

Description

Change column (variable) objective coefficients in the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
changeObjCoefs(lp, j, obj_coef)  
  
## S4 method for signature 'optObj_cplexAPI'  
changeObjCoefs(lp, j, obj_coef)  
  
## S4 method for signature 'optObj_glpkAPI'  
changeObjCoefs(lp, j, obj_coef)  
  
## S4 method for signature 'optObj_lpSolveAPI'  
changeObjCoefs(lp, j, obj_coef)
```

Arguments

lp	An object extending class <code>optObj</code> .
j	A numeric vector containing the column indices of the variables to change.
obj_coef	A numeric vector of the same length as j containing the objective coefficients of the variables to change.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>
Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

changeObjFunc	<i>Sets/changes the Objective Function</i>
---------------	--

Description

The function `changeObjFunc` changes or sets the objective function for a specified model.

Usage

```
changeObjFunc(model, react, obj_coef = rep(1, length(react)))
```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>react</code>	An object of class <code>reactId</code> , character or integer. Specifies the fluxes (variables) for which to change the objective coefficients.
<code>obj_coef</code>	A numerical vector with length equal to the number of reaction id's given in argument <code>react</code> containing the objective coefficients. Default: a value of one for each reaction given in argument <code>react</code> .

Details

The argument `react` will be evaluated by the function `checkReactId`. The return value is used to change the objective function.

All reactions not given in argument `react` will get an objective value of zero.

Value

Returns the given model containing the new objective function.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkReactId](#)

Examples

```
## sets the objective function to the ATP maintenance reaction:  
data(Ec_core)  
Ec_new <- changeObjFunc(Ec_core, "ATPM")
```

changeRowsBnds-methods

Change Row Bounds in the Optimization Problem

Description

Change row bounds in the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
changeRowsBnds(lp, i, lb, ub)
```

```
## S4 method for signature 'optObj_cplexAPI'  
changeRowsBnds(lp, i, lb, ub)
```

```
## S4 method for signature 'optObj_glpkAPI'  
changeRowsBnds(lp, i, lb, ub)
```

```
## S4 method for signature 'optObj_lpSolveAPI'  
changeRowsBnds(lp, i, lb, ub)
```

Arguments

lp	An object extending class <code>optObj</code> .
i	A numeric vector containing the row indices of the constraints to change.
lb	A numeric vector of the same length as <code>i</code> containing the lower bounds of the constraints to change.
ub	A numeric vector of the same length as <code>i</code> containing the upper bounds of the constraints to change.

Methods

```
signature(lp = "optObj_clpAPI") method to use with package optObj_clpAPI.  
signature(lp = "optObj_cplexAPI") method to use with package optObj_cplexAPI.  
signature(lp = "optObj_glpkAPI") method to use with package optObj_glpkAPI.  
signature(lp = "optObj_lpSolveAPI") method to use with package optObj_lpSolveAPI.
```

Note

Changing row bounds does not change the constraint type.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

changeUptake-methods *Change Uptake Reactions*

Description

Switch uptake reactions in metabolic networks on and off.

Usage

```
## S4 method for signature 'modelorg'  
changeUptake(object, off = NULL, on = NULL,  
              rate = SYBIL_SETTINGS("MAXIMUM") * -1)
```

Arguments

object	An object of class modelorg .
off	A numeric or character vector or an object of class reactId_Exch containing the metabolite id's of metabolites to not use for uptake. If they have an exchange reaction with a lower bound less than zero, this lower bound is set to 0. If off is set to NULL, all uptake reactions will be deactivated. If off is set to FALSE, no uptake reaction will be deactivated. If you just want to add an uptake reaction, set off to FALSE. Default: NULL.
on	A numeric or character vector or an object of class reactId_Exch containing the metabolite id's of metabolites to use for uptake. Default: NULL.
rate	A numeric vector containing the uptake rates for metabolites given in on. Default: SYBIL_SETTINGS("MAXIMUM") * -1.

Value

An object of class [modelorg](#).

Methods

`signature(object = "modelorg")` method to use with objects of class [modelorg](#).

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Class [modelorg](#)

checkAlgorithm	<i>Check Algorithm</i>
----------------	------------------------

Description

Test, if a given algorithm can has a certain purpose.

Usage

```
checkAlgorithm(alg, purpose)
```

Arguments

alg	A single character string containing the name of the algorithm.
purpose	Purpose of the new algorithm.

Value

Returns TRUE if successful, otherwise FALSE.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[addAlgorithm](#), [getsybilenv](#)

checkDefaultMethod	<i>Validate Solver and Method</i>
--------------------	-----------------------------------

Description

The function `checkDefaultMethod` returns the default method for a desired solver, or a default solver – method pair. A “solver” is always the name of a R package offering facilities for solving optimization problems.

Usage

```
checkDefaultMethod(solver, method, probType, loadPackage = TRUE)
```

Arguments

<code>solver</code>	A single character string, containing the solver name (must be identical to the name of an R-package), see SYBIL_SETTINGS .
<code>method</code>	A single character string, containing the method name, see SYBIL_SETTINGS .
<code>probType</code>	A single character string, containing the problem type, see optObj .
<code>loadPackage</code>	A single Boolean value. If set to TRUE, load the given solver package via require .

Details

In order to run simulations (optimizations) with `sybil`, additional software offering facilities for solving optimization problems is required. Supported R packages are described in [SYBIL_SETTINGS](#). At first, the function checks if argument `solver` contains a valid solver. If that is not the case, a corresponding library will be loaded, if one exists (this library must have the same name as given in `solver`). If this fails too, the default solver will be returned (see [SYBIL_SETTINGS](#)). Next the same is done for the argument `method`, regarding the current value of `solver`. Additionally, it will be checked, whether or not the given problem type can be solved using the given method and solver.

Value

<code>sol</code>	Validated solver name.
<code>met</code>	Validated method name.
<code>parm</code>	Default parameter set for the validated method.

Note

Arguments `"glpk"`, `"cplex"` and `"clp"` not used anymore; valid arguments must be the name of the desired solver package like `"glpkAPI"`, `"cplexAPI"` and `"cplAPI"`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

[SYBIL_SETTINGS](#) and [getsybilenv](#)

checkOptSol-methods *Summarized Information About an Object of Class OptSol*

Description

The function `checkOptSol` evaluates the results of the solution of optimizations; the returned objects e.g. from `optimizeProb`.

Usage

```
## S4 method for signature 'optsol'  
checkOptSol(opt, onlywarn = FALSE)
```

Arguments

<code>opt</code>	An object of class <code>optsol</code> .
<code>onlywarn</code>	A single Boolean value. If set to TRUE, the method will check, if all optimizations ended successfully. Default: FALSE.

Details

The function `checkOptSol` is used by functions performing a linear optimization (e.g. `optimizeProb`). In that case, the argument `onlywarn` is set to TRUE. If the optimization ends unsuccessful, a warning will be produced.

It is also possible to use the function directly, with `onlywarn` set to FALSE (the default). In that case, an object of class `checksol` will be returned. This object contains a summary with the exit status of the optimization.

Value

TRUE or FALSE if `onlywarn` is set to TRUE, otherwise an object of class `checksol`.

Methods

`signature(opt = "optsol")` method to use with objects of class `optsol`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

`checksol`, `optimizeProb` and `oneGeneDel`

Examples

```
data(Ec_core)
Ec_f <- optimizeProb(Ec_core, retOptSol = TRUE)
Ec_check <- checkOptSol(Ec_f)
```

checkReactId	<i>Check if a Reaction Id is Valid</i>
--------------	--

Description

The function `checkReactId` evaluates a vector of reaction id's if they are unique and appear in a given model.

Usage

```
checkReactId(model, react)
```

Arguments

<code>model</code>	A model. An object of class <code>modelorg</code> , or a problem object of a lp solver.
<code>react</code>	Character vector containing reaction id's, or a numerical vector containing indices of reaction id's.

Details

If argument `react` is numeric, the maximum value will be inspected, if it is larger than the number of reactions in the model.

In case of a character vector, `react` is matched to the reaction id's residing in the model. If they are not found, `grep` is used.

If argument `react` is of class `reactId`, it will be returned without checking.

Value

An object of class `reactId` or NULL if argument `react` contains any reactions not in `model`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

[reactId](#)

Examples

```

data(Ec_core)

## Example with react as character vector
ids <- c("ATPM", "ACK")
idc <- checkReactId(Ec_core, ids)

## Example with react as numerical vector
ids <- c(1:4)
idc <- checkReactId(Ec_core, ids)

```

checksol-class *Structure of the Class "checksol"*

Description

Structure of the class "checksol". Objects of that class are returned by the function [checkOptSol](#).

Objects from the Class

Objects can be created by calls of the form `new("checksol")`.

Slots

`exit_code`: Object of class "integer" containing the exit code of the lp solver.
`exit_num`: Object of class "integer" containing the number of appearance of a specific exit code.
`exit_meaning`: Object of class "character" containing the meaning of the exit code.
`num_of_prob`: Object of class "integer" indicating the number of optimization problems.
`status_code`: Object of class "integer" containing the solution status of the lp problem.
`status_num`: Object of class "integer" containing the number of appearance of a specific solution status.
`status_meaning`: Object of class "character" containing the meaning of the solution status.

Methods

`exit_code<-`: signature(object = "checksol"): sets the `exit_code` slot.
`exit_code`: signature(object = "checksol"): gets the `exit_code` slot.
`exit_meaning<-`: signature(object = "checksol"): sets the `exit_meaning` slot.
`exit_meaning`: signature(object = "checksol"): gets the `exit_meaning` slot.
`exit_num<-`: signature(object = "checksol"): sets the `exit_num` slot.
`exit_num`: signature(object = "checksol"): gets the `exit_num` slot.
`num_of_prob<-`: signature(object = "optsol"): sets the `num_of_prob` slot.
`num_of_prob`: signature(object = "optsol"): gets the `num_of_prob` slot.

```

show: signature(object = "checksol"): prints some details specific to the instance of class
      checksol.
status_code<-: signature(object = "checksol"): sets the status_code slot.
status_code: signature(object = "checksol"): gets the status_code slot.
status_meaning<-: signature(object = "checksol"): sets the status_meaning slot.
status_meaning: signature(object = "checksol"): gets the status_meaning slot.
status_num<-: signature(object = "checksol"): sets the status_num slot.
status_num: signature(object = "checksol"): gets the status_num slot.

```

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

[checkOptSol](#)

Examples

```
showClass("checksol")
```

checkVersion-methods *checks Version of modelorg*

Description

Checks the Version of the modelorg.

Usage

```
## S4 method for signature 'modelorg'
checkVersion(object)
```

Arguments

object An object of class [modelorg](#) or of class [summaryOptsol](#).

Details

This method checks whether this instance of a modelorg-Class is of the currently used version. All methods of sybil create the correct version of modelorg, but if objects saved to disk may be of an older version. Current version can be obtained by SYBIL_SETTINGS("VERSION").

Value

Returns TRUE if the version is correct. Otherwise returns a character stating the reason.

Methods

signature(object = "modelorg") method to use with objects of class `modelorg`.

Author(s)

Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Class `modelorg`, method `upgradeModelorg` and `SYBIL_SETTINGS`

deadEndMetabolites-methods

Identify Dead End Metabolites

Description

Search a metabolic network for metabolites, which are produced, but not consumed and vice versa.

Usage

```
## S4 method for signature 'modelorg'  
deadEndMetabolites(object,retIds)
```

Arguments

object	An object of class <code>modelorg</code> .
retIds	Boolean. If set to TRUE, a list containing metabolite id's will be returned, otherwise a list of logical vectors. Default: TRUE.

Value

A list will be returned:

dem	dead end metabolites
der	reactions containing dead end metabolites

Methods

signature(object = "modelorg") method to use with class `modelorg`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Class [modelorg](#) and [readTSVmod](#).

delProb-methods

Free Memory Associated to the Pointer to the Problem Object

Description

Delete (free) memory associated to the pointer to the problem object.

Usage

```
## S4 method for signature 'optObj_clpAPI'
delProb(lp, ...)

## S4 method for signature 'optObj_cplexAPI'
delProb(lp, closeEnv = TRUE)

## S4 method for signature 'optObj_glpkAPI'
delProb(lp, ...)

## S4 method for signature 'optObj_lpSolveAPI'
delProb(lp, ...)
```

Arguments

lp	An object extending class optObj .
closeEnv	A Boolean value. If set to TRUE, the CPLEX environment associated with the problem object will be closed also. Otherwise not. Default: TRUE.
...	Further arguments passed to the deletion function of the solver package.

Methods

```
signature(lp = "optObj_clpAPI") method to use with package optObj_clpAPI.
signature(lp = "optObj_cplexAPI") method to use with package optObj_cplexAPI.
signature(lp = "optObj_glpkAPI") method to use with package optObj_glpkAPI.
signature(lp = "optObj_lpSolveAPI") method to use with package optObj_lpSolveAPI.
```

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

doubleFluxDel	<i>Double Flux Deletion Experiment</i>
---------------	--

Description

Double reaction (flux) deletion analysis.

Usage

```
doubleFluxDel(model, react1, react2, lb = NULL, ub = NULL,
              allComb = FALSE, exex = FALSE, checkOptSolObj = FALSE, ...)
```

Arguments

model	An object of class modelorg .
react1	An object of class reactId or character or integer containing reaction id's to constrain to zero. Default: <code>react_id(model)</code> .
react2	An object of class reactId or character or integer containing reaction id's to constrain to zero. Default: <code>react_id(model)</code> .
lb	A numeric vector containing the lower bounds for the reaction rates of reactions (variables) given in arguments <code>react1</code> and <code>react2</code> . If set to <code>NULL</code> , all reactions will be constrained to zero. Default: <code>NULL</code> .
ub	A numeric vector containing the upper bounds for the reaction rates of reactions (variables) given in arguments <code>react1</code> and <code>react2</code> . If set to <code>NULL</code> , all reactions will be constrained to zero. Default: <code>NULL</code> .
allComb	A single Boolean value. If set to <code>TRUE</code> , every possible pairwise combination of reactions given in arguments <code>react1</code> and <code>react2</code> will be constrained to zero flux. If set to <code>FALSE</code> , arguments <code>react1</code> and <code>react2</code> must have the same length. The deletions will be computed pair-wise: first <code>react1[1]</code> and <code>react2[1]</code> , second <code>react1[2]</code> and <code>react2[2]</code> and so on. Default: <code>FALSE</code> .
exex	A single Boolean value. If set to <code>TRUE</code> , exchange reactions will be excluded from the analysis. They are identified by the function findExchReact . Default: <code>FALSE</code> .
checkOptSolObj	A single logical value. If set to <code>TRUE</code> , a warning will be generated, if not all optimizations ended successful. Default: <code>FALSE</code> .
...	Further arguments passed to optimizer . Important ones are <code>algorithm</code> in order to set the algorithm to use or <code>solverParm</code> in order to set parameter values for the optimization software.

Details

The function `doubleFluxDel` studies the effect of double flux deletions on the phenotype of the metabolic network. The function performs n optimizations with n being either the number of reaction id's in argument `react1` times the number of reaction id's in argument `react2`, if argument `allComb` is set to `TRUE`, or the length of one of these vectors if argument `allComb` is set to `FALSE`. Each optimization corresponds to the simultaneous deletion of two fluxes.

Value

An object of class `optsol_fluxdel`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

`modelorg`, `optsol`, `optsol_fluxdel`, `checkOptSol`, `optimizer` and `SYBIL_SETTINGS`.

Examples

```
data(Ec_core)
Ec_dfd <- doubleFluxDel(Ec_core)
```

doubleGeneDel

Double Gene Deletion Experiment

Description

Predict the metabolic phenotype of of double-gene knock out mutants.

Usage

```
doubleGeneDel(model, geneList1, geneList2, lb = NULL, ub = NULL,
              allComb = FALSE, exLethal = TRUE,
              tol = SYBIL_SETTINGS("TOLERANCE"),
              checkOptSolObj = FALSE, ...)
```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>geneList1</code>	A character vector containing the set of genes to be deleted. Default: <code>allGenes(model)</code> .
<code>geneList2</code>	A character vector containing the set of genes to be deleted. Default: <code>allGenes(model)</code> .

lb	A numeric vector containing the lower bounds for the reaction rates of reactions (variables) affected by the genes given in arguments <code>geneList1</code> and <code>geneList2</code> . If set to NULL, all reactions affected will be constrained to zero. Default: NULL.
ub	A numeric vector containing the upper bounds for the reaction rates of reactions (variables) affected by the genes given in arguments <code>geneList1</code> and <code>geneList2</code> . If set to NULL, all reactions affected will be constrained to zero. Default: NULL.
allComb	A single Boolean value. If set to TRUE, every possible pairwise combination of genes given in arguments <code>geneList1</code> and <code>geneList2</code> will be knocked-out. If set to FALSE, arguments <code>geneList1</code> and <code>geneList2</code> must have the same length. The knock-outs will be computed pair-wise: first <code>geneList1[1]</code> and <code>geneList2[1]</code> , second <code>geneList1[2]</code> and <code>geneList2[2]</code> and so on. Default: FALSE.
exLethal	A single Boolean value. If set to TRUE, lethal genes are removed from the analysis. A unique set of genes in <code>geneList1</code> and <code>geneList2</code> will be scanned for lethal genes. A particular gene i is considered as lethal, if the deletion of this gene results in a zero flux rate in the objective function given in <code>model</code> . Default: TRUE.
tol	A single numeric value, containing an absolute threshold value for a gene being lethal or not. Default: <code>SYBIL_SETTINGS("TOLERANCE")</code> .
checkOptSolObj	A single logical value. If set to TRUE, a warning will be generated, if not all optimizations ended successful. Default: FALSE.
...	Further arguments passed to <code>optimizer</code> . Important ones are <code>algorithm</code> in order to set the algorithm to use or <code>solverParm</code> in order to set parameter values for the optimization software.

Details

The function `doubleGeneDel` studies the effect of genetic perturbations by double gene deletions on the phenotype of the metabolic network. The function performs n optimizations with n being either the length of the character vector in argument `geneList1` times the length of the character vector in argument `geneList2`, if argument `allComb` is set to TRUE, or the length of one of these vectors if argument `allComb` is set to FALSE. For each gene deletion i, j the set of fluxes effected by the simultaneous deletion of genes i and j is constrained to zero flux. If the deletion of a certain pair of genes has an effect, is tested with the function `geneDel`. Each optimization corresponds to the simultaneous deletion of two genes.

Value

An object of class `optsol_genedel`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[modelorg](#), [optsol](#), [optsol_genedel](#), [checkOptSol](#), [optimizer](#) and [SYBIL_SETTINGS](#).

Examples

```
## Not run:
## compute all possible pairwise gene deletions
# load example data set
data(Ec_core)

# compute all possible pairwise gene deletions via
# FBA (default)
Ec_dgd <- doubleGeneDel(Ec_core, allComb = TRUE)

# or MOMA (linearized version)
Ec_dgd <- doubleGeneDel(Ec_core,
                        allComb = TRUE,
                        algorithm = "lmoma")

## End(Not run)
```

doubleReact

Identifies Identical Reactions

Description

The function `doubleReact` identifies identical reactions (isoenzymes) in a model.

Usage

```
doubleReact(model, checkRev = TRUE, linInd = FALSE)
```

Arguments

<code>model</code>	An object of class modelorg .
<code>checkRev</code>	A single logical value. If set to TRUE, two reactions are identical, if, additionally to the stoichiometric coefficients, the direction of the reactions is the same (the corresponding value of slot react_rev of the model). Default: TRUE.
<code>linInd</code>	A single logical value. If set to TRUE, two reactions are identical, if the vectors of stoichiometric coefficients are linear dependent. For example, two reactions with coefficients $(1, 1, -1)$ and $(2, 2, -2)$ are linear dependent. If the coefficients have different signs, for example $(-1, 1)$ and $(1, -1)$ (the first reaction being forward direction and the second one being backward direction), they are not identical. If <code>linInd</code> is set to FALSE, the stoichiometric must be identical, for two reactions considered to be identical. Default: FALSE.

Details

In the first step, the stoichiometric matrix S is divided into groups of reactions containing the same number of metabolites. After that, the row indices of the non-zero elements of these matrices are compared. If identical pairs are found, we check the corresponding values in S . If they are also identical, the reversibility of the reactions are examined. If they are the same, the two reactions are called identical.

Value

If no identical reactions were found, the return value is FALSE. Otherwise a list is returned, ordered by the number of metabolites used in each reaction. Each element is a numerical vector containing the indices (column number for the stoichiometric matrix) of identical reactions.

Note

At the moment, the directions of a pair of reactions is not compared. Meaning, that if concerning to the values in S the reaction is in forward direction, but not when including the flux values, `doubleReact` will not find it.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

Examples

```
data(Ec_core)
Ec_dr <- doubleReact(Ec_core)
```

Ec_core

Escherichia coli Core Metabolic Model

Description

The dataset is a network representation of the *E. coli* core metabolism. It consists of 95 internal reactions, 20 exchange reactions and a biomass objective function.

Usage

```
data(Ec_core)
```

Format

An object of class `modelorg`

Source

<http://gcrp.ucsd.edu/Downloads/EcoliCore>

References

Bernhard Ø. Palsson (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press.

Orth, J. D., Fleming, R. M. T. and Palsson, B. Ø. (2010). Reconstruction and Use of Microbial Metabolic Networks: the Core Escherichia coli Metabolic Model as an Educational Guide *in* EcoSal Chapter 10.2.1.

editEnvir

Environment Editor for Metabolic Networks

Description

Environment editor for metabolic networks. The function `editEnvir` opens the exchange reactions of a metabolic network in R's data editor. Changes in upper and lower bounds will be set in the given model.

Usage

```
editEnvir(model, newKey = FALSE, ...)
```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>newKey</code>	If set to TRUE, a new model key will be generated.
<code>...</code>	Further arguments passed to <code>edit</code> .

Value

An object of class `modelorg`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

[checkReactId](#)

Examples

```
## Not run:  
## change environment of E.coli core model:  
data(Ec_core)  
mod <- editEnvir(Ec_core)  
  
## End(Not run)
```

findExchReact	<i>Find Exchange Reactions</i>
---------------	--------------------------------

Description

This function identifies reactions in a metabolic network which transport metabolites across the network boundary. Only the stoichiometric matrix is taken into account, so the identified reactions are basically those, having only one non-zero entry in their column of the stoichiometric matrix. In order to work, the network must be “open”, it must not contain boundary metabolites.

Usage

```
findExchReact(model)
```

Arguments

model An object of class `modelorg`, `Matrix` or `matrix`.

Details

A exchange reaction j for a particular metabolite i has exactly one non-zero entry in the stoichiometric matrix $S_{ij} \in \{-1, 1\}$. If $S_{ij} = -1$, reaction j is considered to be an uptake (source) reaction.

Value

If `model` is of class `modelorg` an object of class `reactId_Exch` is returned. Otherwise, if `model` is of class `matrix` or of class `Matrix`, a logical vector is returned. If element i equals TRUE, column i of `model` is an exchange reaction. The function returns `NULL` and gives a warning, if no exchange reaction can be found.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

Examples

```

data(Ec_core)
ex <- findExchReact(Ec_core)

# run FBA
opt <- optimizeProb(Ec_core)

# get flux distribution of exchange reactions
getFluxDist(opt, ex)

```

fluxDistribution-class

Class "fluxDistribution"

Description

Structure of the class "fluxDistribution". Objects of that class are used by class "optsol" in order to store flux distributions. Flux distributions are stored column by column; each flux corresponds to one row and the optimizations correspond to the columns.

Objects from the Class

Objects can be created by calls of the form `test <- fluxDistribution(fluxes, nrow = 1, ncol = 1)`. If argument `fluxes` is of class `Matrix` or `matrix`, `num_of_fluxes` is set to `ncol(fluxes) * nrow(fluxes)`. If argument `fluxes` is a vector, a matrix will be generated according to `nrow` and `ncol`.

Slots

`fluxes`: Object of class "Matrix" containing fluxdistributions column by column.
`num_of_fluxes`: Object of class "integer" containing the number of elements in `fluxes`.

Methods

`[` signature(`x = "fluxDistribution"`): subsetting operator for the matrix of flux distributions.
`fluxes` signature(`object = "fluxDistribution"`): gets the `fluxes` slot.
`fluxes<-` signature(`object = "fluxDistribution"`): sets the `fluxes` slot.
`num_of_fluxes` signature(`object = "fluxDistribution"`): gets the `num_of_fluxes` slot.
`nnzero` signature(`object = "fluxDistribution"`): gets the number of non-zero elements in slot `fluxes`.
`nvar` signature(`object = "fluxDistribution"`): gets the number of fluxes in the fluxdistribution in slot `fluxes` (the number of rows of slot `fluxes`).
`plot` signature(`x = "fluxDistribution"`, `y = "missing"`): heatmap like plotting method for fluxdistributions. Not finished yet.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

Examples

```
showClass("fluxDistribution")
```

fluxVar

Flux Variability Analysis

Description

Performs flux variability analysis for a given model.

Usage

```
fluxVar(model, react = c(1:react_num(model)), exex = FALSE, ...)
```

Arguments

model	An object of class modelorg .
react	An object of class reactId , character or integer. Specifies the fluxes (variables) to analyse. Default: all reactions present in model.
exex	Boolean. Exclude exchange reactions from analysis. If set to TRUE, argument react will be ignored. All reactions present in model will be used, except for the exchange reactions. Default: FALSE
...	Further arguments passed to optimizer . Argument algorithm is set to "fv", further possible arguments are fld, arguments for pre and post processing commands, verboseMode and further arguments passed to the constructor for objects of class sysBiolAlg_fv , see there for details.

Details

The algorithm is described in [sysBiolAlg_fv](#).

Value

An object of class [optsol_fluxVar](#). The first 1 to n (with n being the number of elements in argument react) solutions are from the minimizations, and the last $n + 1$ to $2n$ solutions are from the maximizations.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

Bernhard Ø. Palsson (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press.

Examples

```
data(Ec_core)
fv <- fluxVar(Ec_core)
plot(fv)
```

geneDel

Get Gene-Reaction Association

Description

The function geneDel returns the fluxes which are effected by a particular combination of genes.

Usage

```
geneDel(model, genes, checkId = FALSE)
```

Arguments

model	An object of class <code>modelorg</code> .
genes	A vector of character strings of gene id's used in <code>model</code> , or an integer vector with indices to gene id's in <code>allGenes(model)</code> .
checkId	Boolean. If set to TRUE, argument <code>genes</code> will be checked wether it fits to <code>model</code> (e.g. are all genes existing). If set to FALSE, <code>genes</code> must contain indices of gene id's in <code>model</code> , e.g. in calls from <code>optimizer</code> .

Details

The function geneDel checks for a set of gene id's in gene on which fluxes a deletion of this set of genes has an effect.

Value

An numeric vector of pointers to reaction id's in model or NULL, if no fluxes are effected by the gene deletion.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Edwards, J. S., Ibarra, R. U. and Palsson, B. Ø. (2001) In silico predictions of *Escherichia coli* metabolic capabilities are consistent with experimental data. *Nat Biotechnol* **19**, 125–130.

See Also

[optimizer](#)

geneDeletion

Gene Deletion Experiments

Description

The function geneDeletion studies the effect of n in silico gene deletions on the phenotype of a metabolic network. The value of n is the number of genes knocked-out simultaneously.

Usage

```
geneDeletion(model, genes, combinations = 1,
             lb = NULL, ub = NULL, checkOptSolObj = FALSE, ...)
```

Arguments

model	An object of class modelorg .
genes	Character or Integer: the genes to delete (see Details below).
combinations	A single integer value. If combinations > 1 and genes is not a matrix, combinations is the number of elements from genes taken at a time while building all combinations of the elements in genes (see Details below). Default: 1.
lb	A numeric vector containing the lower bounds for the reaction rates of reactions (variables) affected by the genes given in argument genes. If set to NULL, all reactions affected will be constrained to zero. Default: NULL.
ub	A numeric vector containing the upper bounds for the reaction rates of reactions (variables) affected by the genes given in argument genes. If set to NULL, all reactions affected will be constrained to zero. Default: NULL.

checkOptSolObj A single logical value. If set to TRUE, a warning will be generated, if not all optimizations ended successful.
Default: FALSE.

... Further arguments passed to [optimizer](#). Important ones are `algorithm` in order to set the algorithm to use or `solverParm` in order to set parameter values for the optimization software.

Details

If argument `genes` is a matrix of character values (gene id's) or integers (pointers to gene id's), each column is treated as one deletion experiment. If the matrix is made up of integers, a zero entry means no gene.

If argument `genes` is a character vector or integer, the argument `combinations` gives the number of gene id's taken each time in order to build all possible combinations of genes. A matrix is constructed using [combn](#). The value of argument `combinations` gives the number of genes, which are knocked-out simultaneously. The default value 1 performs a single gene deletion experiment, like the function [oneGeneDel](#) does. A value of 2 performs a double gene deletion as described in [doubleGeneDel](#). A value of n performs an n gene deletion experiment. Keep in mind, that the number of optimizations will get very high for increasing values of combinations.

If argument `genes` is empty, the number of unique genes present in `model` is used.

The required length of arguments `lb` and `ub` (if not NULL) depends on the values given in arguments `genes` and `combinations`. If `genes` is a matrix, `lb` and `ub` must be of length equal to the number of columns in `genes`. If `genes` is a vector, `lb` and `ub` must be of length equal to `length(genes) * combinations`.

Value

An object of class [optsol_genedel](#).

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[modelorg](#), [optsol](#), [optsol_genedel](#), [checkOptSol](#), [oneGeneDel](#), [optimizer](#), [optimizeProb](#), [combn](#) and [SYBIL_SETTINGS](#).

Examples

```
## load the dataset
data(Ec_core)

## perform a single gene deletion analysis
## (delete every gene one by one) via FBA
gd <- geneDeletion(Ec_core)

## or via MOMA (linearized version)
```

```
gd <- geneDeletion(Ec_core, algorithm = "lmoma")

## triple gene deletion analysis using the first ten genes
gd <- geneDeletion(Ec_core, genes = 10, combinations = 3)

## Not run:
## perform a double gene deletion analysis
##(delete all possible pairwise combinations of all genes)
gd <- geneDeletion(Ec_core, combinations = 2)

## perform a triple gene deletion analysis
## (very high number of optimizations)
gd <- geneDeletion(Ec_core, combinations = 3)

## End(Not run)
```

getColPrim-methods *Get Primal Value of Variables After Optimization*

Description

Get primal value of variables after optimization.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'
getColPrim(lp, j)

## S4 method for signature 'optObj_cplexAPI,numeric'
getColPrim(lp, j)

## S4 method for signature 'optObj_glpkAPI,numeric'
getColPrim(lp, j)

## S4 method for signature 'optObj_lpSolveAPI,numeric'
getColPrim(lp, j)
```

Arguments

lp An object extending class `optObj`.

j A numeric vector containing the column (variable) indices.

Value

A numeric vector containing the desired primal values.

Methods

signature(lp = "optObj_clpAPI", j = "numeric") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI", j = "numeric") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI", j = "numeric") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI", j = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getColsLowBnds-methods

Get Lower Bounds of the Columns (Variables) of the Optimization Problem

Description

Get lower bounds of the columns (variables) of the optimization Problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'
getColsLowBnds(lp, j)

## S4 method for signature 'optObj_cplexAPI,numeric'
getColsLowBnds(lp, j)

## S4 method for signature 'optObj_glpkAPI,numeric'
getColsLowBnds(lp, j)

## S4 method for signature 'optObj_lpSolveAPI,numeric'
getColsLowBnds(lp, j)
```

Arguments

lp An object extending class [optObj](#).
j A numeric vector containing the column (variable) indices.

Value

A numeric vector containing the desired column bounds.

Methods

signature(lp = "optObj_clpAPI", j = "numeric") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI", j = "numeric") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI", j = "numeric") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI", j = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

getColsNames-methods *Retrieve Variable Names*

Description

Get names of variables (columns) used in a optimization problem.

Usage

```
## S4 method for signature 'optObj_cplexAPI,numeric'
getColsNames(lp, j)

## S4 method for signature 'optObj_glpkAPI,numeric'
getColsNames(lp, j)

## S4 method for signature 'optObj_lpSolveAPI,numeric'
getColsNames(lp, j)
```

Arguments

lp An object extending class `optObj`.
j A numeric vector of column indices.

Value

A character vector of column names, if names are existing.

Methods

signature(lp = "optObj_cplexAPI", j = "numeric") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI", j = "numeric") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI", j = "numeric") method to use with package **optObj_lpSolveAPI**.

Note

For the `optObj_glpkAPI` method: the result vector may be shorter than `j`, if some names are missing.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

getColsUppBnds-methods

Get Upper Bounds of the Columns (Variables) of the Optimization Problem

Description

Get upper bounds of the columns (variables) of the optimization Problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'  
getColsUppBnds(lp, j)  
  
## S4 method for signature 'optObj_cplexAPI,numeric'  
getColsUppBnds(lp, j)  
  
## S4 method for signature 'optObj_glpkAPI,numeric'  
getColsUppBnds(lp, j)  
  
## S4 method for signature 'optObj_lpSolveAPI,numeric'  
getColsUppBnds(lp, j)
```

Arguments

`lp` An object extending class `optObj`.
`j` A numeric vector containing the column (variable) indices.

Value

A numeric vector containing the desired column bounds.

Methods

signature(lp = "optObj_clpAPI", j = "numeric") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI", j = "numeric") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI", j = "numeric") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI", j = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getFluxDist-methods *Retrieve Flux Distribution*

Description

Get all primal values of variables after optimization (the resulting flux distribution).

Usage

```
## S4 method for signature 'optObj_clpAPI'  
getFluxDist(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'  
getFluxDist(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'  
getFluxDist(lp)
```

```
## S4 method for signature 'optObj_lpSolveAPI'  
getFluxDist(lp)
```

```
## S4 method for signature 'optsol'  
getFluxDist(lp, react = NULL, opt = NULL, drop = TRUE)
```

Arguments

lp An object extending class [optObj](#) or class [optsol](#).

react Numeric vector or object of class [reactId](#) indicating the reactions (rows of the flux distribution) to return.
Default: NULL.

opt	Numeric vector indicating the optimizations (columns of the flux distribution) to return. Default: NULL.
drop	Used for array subsetting like in <code>[]</code> . Default: TRUE.

Value

A numeric matrix or vector containing all primal values (the flux distribution).

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

signature(lp = "optsol") method to use with objects of class **optsol**. Returns a subset of the flux distribution stored in slot `fluxdist` as object of class **Matrix**. If arguments `react` and `opt` are both set to NULL (default), the flux distribution corresponding to the variable indices in slot `fldind` will be returned.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass **optObj** and constructor function **optObj**.

getNumCols-methods *Get Number of Columns (Variables) of the Optimization Problem*

Description

Get number of columns (variables) of the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'
getNumCols(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'
getNumCols(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'
```



```
getNumCols(lp)

## S4 method for signature 'optObj_lpSolveAPI'
getNumCols(lp)
```

Arguments

`lp` An object extending class `optObj`.

Value

A single numeric value.

Methods

```
signature(lp = "optObj_clpAPI") method to use with package optObj_clpAPI.
signature(lp = "optObj_cplexAPI") method to use with package optObj_cplexAPI.
signature(lp = "optObj_glpkAPI") method to use with package optObj_glpkAPI.
signature(lp = "optObj_lpSolveAPI") method to use with package optObj_lpSolveAPI.
```

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

getNumNnz-methods *Retrieve the Number of Non-Zero Elements of the Constraint Matrix*

Description

Retrieve the number of non-zero elements in the constraint matrix of the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'
getNumNnz(lp)

## S4 method for signature 'optObj_cplexAPI'
getNumNnz(lp)

## S4 method for signature 'optObj_glpkAPI'
getNumNnz(lp)
```

Arguments

lp An object extending class [optObj](#).

Value

A single numeric value.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getNumRows-methods *Get Number of Rows (Constraints) of the Optimization Problem*

Description

Get number of rows (constraints) of the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'
getNumRows(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'
getNumRows(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'
getNumRows(lp)
```

```
## S4 method for signature 'optObj_lpSolveAPI'
getNumRows(lp)
```

Arguments

lp An object extending class [optObj](#).

Value

A single numeric value.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getObjCoefs-methods *Get Objective Coefficients of the Optimization Problem*

Description

Get objective coefficients of the optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'  
getObjCoefs(lp, j)  
  
## S4 method for signature 'optObj_cplexAPI,numeric'  
getObjCoefs(lp, j)  
  
## S4 method for signature 'optObj_glpkAPI,numeric'  
getObjCoefs(lp, j)  
  
## S4 method for signature 'optObj_lpSolveAPI,numeric'  
getObjCoefs(lp, j)
```

Arguments

lp An object extending class [optObj](#).
j A numeric vector containing the column (variable) indices.

Value

A numeric vector containing the desired objective coefficients.

Methods

signature(lp = "optObj_clpAPI", j = "numeric") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI", j = "numeric") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI", j = "numeric") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI", j = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getObjDir-methods *Get Direction of Optimization.*

Description

Get direction of optimization.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
getObjDir(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'  
getObjDir(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'  
getObjDir(lp)
```

```
## S4 method for signature 'optObj_lpSolveAPI'  
getObjDir(lp)
```

Arguments

lp An object extending class [optObj](#).

Value

Returns a single character string indicating the direction of optimization: "max": maximization, or "min": minimization.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getObjVal-methods

Get Value of the Objective Function After Optimization

Description

Get value of the objective function after optimization.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
getObjVal(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'  
getObjVal(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'  
getObjVal(lp)
```

```
## S4 method for signature 'optObj_lpSolveAPI'  
getObjVal(lp)
```

Arguments

lp An object extending class [optObj](#).

Value

Returns a single numeric value.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**. For problems of type "mip": if no solution exists, the **cplexAPI** function `getBestObjValCPLEX` will be used.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getRedCosts-methods *Get Reduced Costs of all Variables After Optimization*

Description

Get reduced costs of all variables after optimization.

Usage

```
## S4 method for signature 'optObj_clpAPI'
getRedCosts(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'
getRedCosts(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'
getRedCosts(lp)
```

```
## S4 method for signature 'optObj_lpSolveAPI'
getRedCosts(lp)
```

Arguments

lp An object extending class [optObj](#).

Value

A numeric vector containing the reduced costs of all variables.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getRowsLowBnds-methods

Get Lower Bounds of the Rows (Constraints) of the Optimization Problem

Description

Get lower bounds of the rows (constraints) of the optimization Problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'  
getRowsLowBnds(lp, i)
```

```
## S4 method for signature 'optObj_cplexAPI,numeric'  
getRowsLowBnds(lp, i)
```

```
## S4 method for signature 'optObj_glpkAPI,numeric'  
getRowsLowBnds(lp, i)
```

```
## S4 method for signature 'optObj_lpSolveAPI,numeric'  
getRowsLowBnds(lp, i)
```

Arguments

lp An object extending class [optObj](#).

i A numeric vector containing the row indices.

Value

A numeric vector containing the desired row bounds.

Methods

signature(lp = "optObj_clpAPI", i = "numeric") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI", i = "numeric") method to use with package **optObj_cplexAPI**.

This method returns always FALSE.

signature(lp = "optObj_glpkAPI", i = "numeric") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI", i = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getRowsNames-methods *Retrieve Constraint Names*

Description

Get names of constraints (rows) used in a optimization problem.

Usage

```
## S4 method for signature 'optObj_cplexAPI,numeric'
getRowsNames(lp, i)
```

```
## S4 method for signature 'optObj_glpkAPI,numeric'
getRowsNames(lp, i)
```

```
## S4 method for signature 'optObj_lpSolveAPI,numeric'
getRowsNames(lp, i)
```

Arguments

lp An object extending class [optObj](#).

i A numeric vector of row indices.

Value

A character vector of row names, if names are existing.

Methods

signature(lp = "optObj_cplexAPI", i = "numeric") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI", i = "numeric") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI", i = "numeric") method to use with package **optObj_lpSolveAPI**.

Note

For the `optObj_glpkAPI` method: the result vector may be shorter than `i`, if some names are missing.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

getRowsUppBnds-methods

Get Upper Bounds of the Rows (Constraints) of the Optimization Problem

Description

Get upper bounds of the rows (constraints) of the optimization Problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric'
getRowsUppBnds(lp, i)

## S4 method for signature 'optObj_cplexAPI,numeric'
getRowsUppBnds(lp, i)

## S4 method for signature 'optObj_glpkAPI,numeric'
getRowsUppBnds(lp, i)

## S4 method for signature 'optObj_lpSolveAPI,numeric'
getRowsUppBnds(lp, i)
```

Arguments

`lp` An object extending class `optObj`.
`i` A numeric vector containing the row indices.

Value

A numeric vector containing the desired row bounds.

Methods

signature(lp = "optObj_clpAPI", i = "numeric") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI", i = "numeric") method to use with package **optObj_cplexAPI**.
This method returns always FALSE.

signature(lp = "optObj_glpkAPI", i = "numeric") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI", i = "numeric") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getSolStat-methods *Get Solution Status After Optimization*

Description

Get solution status after optimization.

Usage

```
## S4 method for signature 'optObj_clpAPI'
getSolStat(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'
getSolStat(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'
getSolStat(lp)
```

```
## S4 method for signature 'optObj_lpSolveAPI'
getSolStat(lp)
```

Arguments

lp An object extending class [optObj](#).

Value

Returns a single numeric value indicating the solution status after optimization.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**. This method returns NA. Package **lpSolveAPI** does not provide a solution status.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Function [getMeanStatus](#) and superclass [optObj](#) and constructor function [optObj](#).

getSolverParm-methods *Retrieve Current Parameter Settings Used By The Optimization Software*

Description

Retrieve current parameter settings used by the optimization software.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
getSolverParm(lp)
```

```
## S4 method for signature 'optObj_cplexAPI'  
getSolverParm(lp)
```

```
## S4 method for signature 'optObj_glpkAPI'  
getSolverParm(lp)
```

```
## S4 method for signature 'optObj_lpSolveAPI'  
getSolverParm(lp)
```

Arguments

lp An object extending class [optObj](#).

Value

Returns a list containing the current parameter settings or zero/non-zero.

Methods

`signature(lp = "optObj_clpAPI")` method to use with package **optObj_clpAPI**. This method is currently unused. It is not possible to provide parameters for package **clpAPI**. Always FALSE will be returned.

`signature(lp = "optObj_cplexAPI")` method to use with package **optObj_cplexAPI**. This method writes the current parameter settings to the file "cplex_parameters.prm". The method returns zero if successful, otherwise non-zero.

`signature(lp = "optObj_glpkAPI")` method to use with package **optObj_glpkAPI**.

`signature(lp = "optObj_lpSolveAPI")` method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

getsybilenv

Print sybil Environment

Description

Prints current settings in the sybil environment.

Usage

```
getsybilenv(part)
```

Arguments

<code>part</code>	A character vector containing names of elements in the sybil environment. Possible values are: <ul style="list-style-type: none"> "solvers" supported R packages for solving optimization problems. "methods" methods to solve optimization problems included in the R packages. "ptype" methods required for a particular problem type. "purpose" algorithms used in systems biology to use with a particular purpose.
-------------------	--

Details

Typical usages are

```
getsybilenv(part)
getsybilenv()
```

If argument `part` is not given, all elements described above will be printed.

Value

Returns NULL invisibly.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

[addSolver](#), [checkDefaultMethod](#) and [SYBIL_SETTINGS](#).

initProb-methods	<i>Initialize Problem Object</i>
------------------	----------------------------------

Description

Initialize Problem Object.

Usage

```
## S4 method for signature 'optObj_clpAPI'
initProb(lp, to = NULL, ...)

## S4 method for signature 'optObj_cplexAPI'
initProb(lp, to = FALSE, ...)

## S4 method for signature 'optObj_glpkAPI'
initProb(lp, to = FALSE, ...)

## S4 method for signature 'optObj_lpSolveAPI'
initProb(lp, to = NULL, nrows, ncols)
```

Arguments

<code>lp</code>	An object extending class <code>optObj</code> .
<code>to</code>	A single boolean, numeric or character value, controlling the amount of terminal output of the solver software. Default: FALSE or NULL.
<code>nrows</code>	Number of rows (constraints) of the new problem object.
<code>ncols</code>	Number of columns (variables) of the new problem object.
<code>...</code>	Further arguments passed to the initialization function of the solver package.

Methods

`signature(lp = "optObj_clpAPI")` method to use with package **optObj_clpAPI**, argument to can be a single numeric value: 0 – “none”, 1 – “just final”, 2 – “just factorizations”, 3 – “as 2 plus a bit more”, code4 – “verbose”. See COIN-OR Clp documentation for more details.

`signature(lp = "optObj_cplexAPI")` method to use with package **optObj_cplexAPI**, argument to can be TRUE or FALSE. Setting CPLEX parameter CPX_PARAM_SCRIND to CPX_ON or CPX_OFF has the same effect.

`signature(lp = "optObj_glpkAPI")` method to use with package **optObj_glpkAPI**, argument to can be TRUE or FALSE, setting GLPK function termOutGLPK to GLP_ON or GLP_OFF. The amount of output is controlled by the GLPK parameter MSG_LEV.

`signature(lp = "optObj_lpSolveAPI")` method to use with package **optObj_lpSolveAPI**, argument to can be a single character value, see **lpSolveAPI** documentation for more details (`lp.control.options`, section `verbose`).

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

Description

Load data to the problem object (extending class `optObj`). Use this method to generate problem objects.

Usage

```
## S4 method for signature 'optObj_clpAPI'
loadLPprob(lp,
            nCols, nRows, mat, ub, lb, obj, rlb, rtype,
            lpdir = "max", rub = NULL, ctype = NULL,
            cnames = NULL, rnames = NULL, pname = NULL,
            defLowerBnd = SYBIL_SETTINGS("MAXIMUM") * -1,
            defUpperBnd = SYBIL_SETTINGS("MAXIMUM")
)

## S4 method for signature 'optObj_cplexAPI'
loadLPprob(lp,
            nCols, nRows, mat, ub, lb, obj, rlb, rtype,
            lpdir = "max", rub = NULL, ctype = NULL,
            cnames = NULL, rnames = NULL, pname = NULL)

## S4 method for signature 'optObj_glpkAPI'
loadLPprob(lp,
            nCols, nRows, mat, ub, lb, obj, rlb, rtype,
            lpdir = "max", rub = NULL, ctype = NULL,
            cnames = NULL, rnames = NULL, pname = NULL)

## S4 method for signature 'optObj_lpSolveAPI'
loadLPprob(lp,
            nCols, nRows, mat, ub, lb, obj, rlb, rtype,
            lpdir = "max", rub = NULL, ctype = NULL,
            cnames = NULL, rnames = NULL, pname = NULL)
```

Arguments

lp	An object of class <code>optObj_clpAPI</code> , <code>optObj_cplexAPI</code> , <code>optObj_glpkAPI</code> or <code>optObj_lpSolveAPI</code> .
nCols	Number of columns (variables) of the constraint matrix.
nRows	Number of rows (constraints) of the constraint matrix.
mat	An object of class <code>Matrix</code> . The constraint matrix of the problem object. The number of columns in <code>mat</code> must be <code>nCols</code> and the number of rows in <code>mat</code> must be <code>nRows</code> .
ub	A numeric vector of length <code>nCols</code> giving the upper bounds of the variables of the problem object.
lb	A numeric vector of length <code>nCols</code> giving the lower bounds of the variables of the problem object.
obj	A numeric vector of length <code>nCols</code> giving the objective coefficients of the variables of the problem object.
rlb	A numeric vector of length <code>nRows</code> giving the right hand side of the problem object. If argument <code>rub</code> is not <code>NULL</code> , <code>rlb</code> contains the lower bounds of the constraints of the problem object. See Details.

rtype A character vector of length nRows giving the constraint type:

"F":	free constraint (GLPK only)	$-\infty < x < \infty$
"L":	constraint with lower bound	$lb \leq x < \infty$
"U":	constraint with upper bound	$-\infty < x \leq ub$
"D":	double-bounded (ranged) constraint	$lb \leq x \leq ub$
"E":	fixed (equality) constraint	$lb = x = ub$

If `rtype[i]` is not one of "F", "L", "U", "D" or "E", the value of `rtype[i]` will be set to "E". See Details.

<code>lpdir</code>	Single character string containing the direction of optimization. Can be set to "min" or "max". Default: "max".										
<code>rub</code>	A numeric vector of length <code>nRows</code> giving the right hand side of the problem object. If not NULL, it contains the upper bounds of the constraints of the problem object. See Details. Default: NULL.										
<code>ctype</code>	A character vector of length <code>nCols</code> giving the variable type. If set to NULL, no specific variable type is set, which usually means, all variables are treated as continuous variables. Default: NULL. <table> <tr> <td>"C":</td> <td>continuous variable</td> </tr> <tr> <td>"B":</td> <td>binary variable</td> </tr> <tr> <td>"I":</td> <td>integer variable</td> </tr> <tr> <td>"S":</td> <td>semi-continuous variable</td> </tr> <tr> <td>"N":</td> <td>semi-integer variable</td> </tr> </table>	"C":	continuous variable	"B":	binary variable	"I":	integer variable	"S":	semi-continuous variable	"N":	semi-integer variable
"C":	continuous variable										
"B":	binary variable										
"I":	integer variable										
"S":	semi-continuous variable										
"N":	semi-integer variable										
	Values "S" and "N" are not available for every solver software. Check documentation of the solver software if semi-continuous and semi-integer variables are supported. If <code>ctype[j]</code> is not "C", "B", "I", "S", or "N", the value of <code>ctype[j]</code> will be set to "C".										
<code>cnames</code>	A character vector of length <code>nCols</code> containing symbolic names for the variable of the problem object. Default: NULL.										
<code>rnames</code>	A character vector of length <code>nRows</code> containing symbolic names for the constraints of the problem object. Default: NULL.										
<code>pname</code>	A single character string containing a name for the problem object. Default: NULL.										
<code>defLowerBnd</code>	For the <code>optObj_cIpAPI</code> method only: a single numeric value containing a default value for an lower bound to a constraint in an optimization problem. Default: <code>SYBIL_SETTINGS("MAXIMUM") * -1</code> .										
<code>defUpperBnd</code>	For the <code>optObj_cIpAPI</code> method only: a single numeric value containing a default value for an upper bound to a constraint in an optimization problem. Default: <code>SYBIL_SETTINGS("MAXIMUM")</code> .										

Details

Method `loadLPprob` can be used any time after a problem object is initialized by `initProb`.

In order to set constraints, usually only parameter `r1b` is required and parameter `rub` can be left at `NULL` (which is the default). If `rub` is not `NULL`, `r1b` and `rub` must have the same length. Parameter `rub` is required, if a particular constraint is a ranged or double bounded constraint. The general idea is, for any constraint i , the value in `r1b[i]` gives the lower bound and the value in `rub[i]` gives the upper bound. If the constraints of the optimization problem do only have one bound (type "L", "U" and "E"), all bounds can be set via `r1b` and `rub` is not required. If any constraint is of type "D" (a double-bounded or ranged constraint) additionally `rub` is required. It is of course also possible to use `r1b` strictly for all lower bounds and `rub` for all upper bounds. Again, if both `r1b` and `rub` are given (not `NULL`), they must have the same length. For equality constraints (type "E"), always the value in `r1b` is used.

For the `optObj_cplexAPI` method: CPLEX uses so called ranged constraints for double bounded constraints. The values in `r1b` and `rub` will be transformed into range values for ranged constraints. The range for a ranged constraint i is given as $\text{abs}(\text{rub}[i] - \text{r1b}[i])$, so that the valid interval is denoted as $[\text{r1b}[i], \text{r1b}[i] + \text{range}]$.

For the `optObj_glpkAPI` method: if `cnames` or `rnames` is not `NULL`, an index will be created.

For the `optObj_clpAPI` method: if `cnames` is not `NULL`, `rnames` must be also not `NULL` and vice versa.

For the `optObj_lpSolveAPI` method: if `cnames` is not `NULL`, `rnames` must be also not `NULL` and vice versa. Round brackets ("(" and ")") will be replaced by underscores "_".

Methods

`signature(lp = "optObj_clpAPI")` method to use with package **clpAPI**.

`signature(lp = "optObj_cplexAPI")` method to use with package **cplexAPI**.

`signature(lp = "optObj_glpkAPI")` method to use with package **glpkAPI**.

`signature(lp = "optObj_lpSolveAPI")` method to use with package **lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

loadQobj-methods	<i>Load Quadratic Part of the Objective Function to the Optimization Problem</i>
------------------	--

Description

load quadratic part of the objective function to the optimization problem.

Usage

```
## S4 method for signature 'optObj_cplexAPI,Matrix'  
loadQobj(lp, mat)  
## S4 method for signature 'optObj_cplexAPI,numeric'  
loadQobj(lp, mat)
```

Arguments

lp	An object extending class <code>optObj</code> .
mat	An object of class <code>Matrix</code> or a numeric vector containing the quadratic objective Matrix Q .

Methods

signature(lp = "optObj_cplexAPI", mat = "Matrix") method to use with package `optObj_cplexAPI` and if mat is of class `Matrix`.

signature(lp = "optObj_cplexAPI", mat = "numeric") method to use with package `optObj_cplexAPI` and if mat is a numeric vector.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

makeOptsolMO	<i>Constructor Function for Objects of Class <code>optsol_optimizeProb</code>.</i>
--------------	--

Description

This function is a constructor function generating objects of class `optsol_optimizeProb`.

Usage

```
makeOptsolMO(mod, sol)
```

Arguments

mod	An object of class <code>modelorg</code> .
sol	A list returned by function <code>optimizer</code> .

Value

An object of class `optsol_optimizeProb`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Class `optsol_optimizeProb`, class `modelorg` and function `optimizer`.

mod2irrev	<i>Produces a Model in Irreversible Format</i>
-----------	--

Description

The function `mod2irrev` produces a model with all reactions moving in positive direction.

Usage

```
mod2irrev(model, exex = FALSE)
```

Arguments

model	An object of class <code>modelorg</code> .
exex	Boolean. Exclude exchange fluxes (default: FALSE).

Details

The returned model consists only of reactions moving in positive direction. Reactions with a negative direction in the original model are transferred to positive direction; the corresponding reaction id gets extended by “_r”.

Reversible reactions are split into two reactions. The corresponding reaction ids gets extended by “_f”, or “_b” indicating the original direction.

If `exex = TRUE`, the exchange reactions were obtained by `findExchReact`.

Value

An object of class `modelorg_irrev`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

See Also

`modelorg_irrev`

Examples

```
data(Ec_core)
Ec_ir <- mod2irrev(Ec_core)
```

modelorg-class

Structure of Class "modelorg"

Description

Structure of the class “modelorg”. Objects of that class are returned by functions like `readTSVmod`.

Objects from the Class

Objects can be created by calls of the function `modelorg`:

```
test <- modelorg(id = "foo", name = "bar", subSys = NULL, compartment = NULL).
```

id: a single character string giving the model id.

name: a single character string giving the model name.

subSys: an optional single character string giving the metabolic subsystems of the model. Default: NULL

compartment: an optional single character string giving the compartments of the model. Default: NULL

This constructor also generates the model key used in slot `mod_key`.

Slots

mod_desc: Object of class "character" containing a description of the model.

mod_name: Object of class "character" indicating the model name.

version: Object of class "character" indicating the model version.

mod_id: Object of class "character" indicating the model id.

mod_key: Object of class "character" containing a single character string functioning as a unique key to a model object.

mod_attr: Object of class "data.frame" to store additional attributes of the model.

mod_compart: Object of class "character" containing the model compartments.

comp_attr: Object of class "data.frame" to store additional attributes for each compartment.

met_num: Object of class "integer" indicating the number of metabolites.

met_id: Object of class "character" containing the metabolite id's.

met_name: Object of class "character" containing the metabolite names.

met_comp: Object of class "integer" containing the metabolites compartment.

met_attr: Object of class "data.frame" to store additional attributes for each metabolite.

met_single: Object of class "logical" with length `met_num`. Element `i` is TRUE, if metabolite `i` appears only once in `S`.

met_de: Object of class "logical" with length `met_num`. Element `i` is TRUE, if metabolite `i` is a dead end metabolite.

react_num: Object of class "integer" indicating the number of reactions.

react_rev: Object of class "logical" indicating whether a reaction is reversible or not.

react_id: Object of class "character" containing the reaction id's.

react_name: Object of class "character" containing the reaction names.

react_attr: Object of class "data.frame" to store additional attributes for each reaction.

react_single: Object of class "logical" with length `react_num`. Element `i` is TRUE, if reaction `i` uses metabolites appearing only once in `S`.

`react_de`: Object of class "logical" with length `react_num`. Element `i` is TRUE, if reaction `i` uses dead end metabolites.
`S`: Object of class "matrix" containing the stoichiometric matrix.
`lowbnd`: Object of class "numeric" containing the reactions lower bounds.
`uppbnd`: Object of class "numeric" containing the reactions upper bounds.
`obj_coef`: Object of class "numeric" containing the objective coefficients.
`gprRules`: Object of class "character" containing the gene-reaction association rules in computable form.
`genes`: Object of class "list" containing the genes corresponding to each reaction. Every list element is a vector of the type character.
`gpr`: Object of class "character" containing the gene-reaction association rules for each reaction.
`allGenes`: Object of class "character" containing a unique list of all genes.
`rxnGeneMat`: Object of class "matrix" containing a reaction to gene mapping.
`subSys`: Object of class "matrix" giving one or more subsystem name for each reaction.

Methods

`allGenes<-:` signature(object = "modelorg"): sets the `allGenes` slot.
`allGenes:` signature(object = "modelorg"): gets the `allGenes` slot.
`dim:` signature(object = "modelorg"): gets the dimension attribute of slot `S`.
`genes<-:` signature(object = "modelorg"): sets the `genes` slot.
`genes:` signature(object = "modelorg"): gets the `genes` slot.
`gpr<-:` signature(object = "modelorg"): sets the `gpr` slot.
`gpr:` signature(object = "modelorg"): gets the `gpr` slot.
`gprRules<-:` signature(object = "modelorg"): sets the `gprRules` slot.
`gprRules:` signature(object = "modelorg"): gets the `gprRules` slot.
`lowbnd<-:` signature(object = "modelorg"): sets the `lowbnd` slot.
`lowbnd:` signature(object = "modelorg"): gets the `lowbnd` slot.
`met_comp<-:` signature(object = "modelorg"): sets the `met_comp` slot.
`met_comp:` signature(object = "modelorg"): gets the `met_comp` slot.
`met_de<-:` signature(object = "modelorg"): sets the `met_de` slot.
`met_de:` signature(object = "modelorg"): gets the `met_de` slot.
`met_id<-:` signature(object = "modelorg"): sets the `met_id` slot.
`met_id:` signature(object = "modelorg"): gets the `met_id` slot.
`met_name<-:` signature(object = "modelorg"): sets the `met_name` slot.
`met_name:` signature(object = "modelorg"): gets the `met_name` slot.
`met_num<-:` signature(object = "modelorg"): sets the `met_num` slot.
`met_num:` signature(object = "modelorg"): gets the `met_num` slot.
`met_single<-:` signature(object = "modelorg"): sets the `met_single` slot.

met_single: signature(object = "modelorg"): gets the met_single slot.
mod_compart<-: signature(object = "modelorg"): sets the mod_compart slot.
mod_compart: signature(object = "modelorg"): gets the mod_compart slot.
mod_desc<-: signature(object = "modelorg"): sets the mod_desc slot.
mod_desc: signature(object = "modelorg"): gets the mod_desc slot.
mod_id<-: signature(object = "modelorg"): sets the mod_id slot.
mod_id: signature(object = "modelorg"): gets the mod_id slot.
mod_key<-: signature(object = "modelorg"): sets the mod_key slot.
mod_key: signature(object = "modelorg"): gets the mod_key slot.
mod_name<-: signature(object = "modelorg"): sets the mod_name slot.
mod_name: signature(object = "modelorg"): gets the mod_name slot.
obj_coef<-: signature(object = "modelorg"): sets the obj_coef slot.
obj_coef: signature(object = "modelorg"): gets the obj_coef slot.
printObjFunc: signature(object = "modelorg"): prints the objective function in a human readable way.
react_de<-: signature(object = "modelorg"): sets the react_de slot.
react_de: signature(object = "modelorg"): gets the react_de slot.
react_id<-: signature(object = "modelorg"): sets the react_id slot.
react_id: signature(object = "modelorg"): gets the react_id slot.
react_name<-: signature(object = "modelorg"): sets the react_name slot.
react_name: signature(object = "modelorg"): gets the react_name slot.
react_num<-: signature(object = "modelorg"): sets the react_num slot.
react_num: signature(object = "modelorg"): gets the react_num slot.
react_rev<-: signature(object = "modelorg"): sets the react_rev slot.
react_rev: signature(object = "modelorg"): gets the react_rev slot.
react_single<-: signature(object = "modelorg"): sets the react_single slot.
react_single: signature(object = "modelorg"): gets the react_single slot.
rxnGeneMat<-: signature(object = "modelorg"): sets the rxnGeneMat slot.
rxnGeneMat: signature(object = "modelorg"): gets the rxnGeneMat slot.
show: signature(object = "modelorg"): prints some details specific to the instance of class modelorg.
Snnz: signature(object = "modelorg"): prints the number of non-zero elements in S.
S<-: signature(object = "modelorg"): sets the S slot as matrix, see Details below.
S: signature(object = "modelorg"): gets the S slot as matrix.
subSys<-: signature(object = "modelorg"): sets the subSys slot.
subSys: signature(object = "modelorg"): gets the subSys slot.
uppbnd<-: signature(object = "modelorg"): sets the uppbnds slot.
uppbnd: signature(object = "modelorg"): gets the uppbnd slot.
version<-: signature(object = "modelorg"): sets the version slot.
version: signature(object = "modelorg"): gets the version slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[modelorg_irrev](#) for models in irreversible format.

Examples

```
showClass("modelorg")

## print human readable version of the objective function
data(Ec_core)
printObjFunc(Ec_core)

## change objective function and print
Ec_objf <- changeObjFunc(Ec_core, c("EX_etoh(e)", "ETOht2r"), c(1, 2))
printObjFunc(Ec_objf)
```

modelorg2ExPA

Write an Instance of Class modelorg to File in ExPA Format

Description

The function `modelorg2ExPA` writes the content of an instance of class `modelorg` to text files in a format which can be read by the program ExPA to compute extreme pathways.

Usage

```
modelorg2ExPA(model, fname = NULL, exIntReact = NULL,
              filepath = ".", suffix = "expa",
              tol = SYBIL_SETTINGS("TOLERANCE"))
```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>fname</code>	An single character string giving the filename to write to. Default: <model_id>.expa.
<code>exIntReact</code>	An object of class <code>reactId</code> , character or integer, giving id's of internal reactions to exclude in the ExPA file. Default: NULL.
<code>filepath</code>	A single character string giving the path to a certain directory in which the output files will be stored. Default: ".".
<code>suffix</code>	A single character string giving the file name extension. Default: "expa".

`tol` A single numeric value giving the limit of tolerance. An element S_{ij} of the stoichiometric matrix is treated as non-zero, if $|S_{ij}| > \text{tol}$ is true. Default: "expa".

Details

The function `modelorg2ExPA` produces input files for the program ExPA. With ExPA, it is possible to calculate extreme pathways in metabolic networks.

The function produces a warning, if a reaction contains non-integer stoichiometric values, because they are not compatible with the ExPA program.

Value

Returns TRUE invisibly on success.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Bell, S. L. and Palsson, B. Ø. (2005) Expa: a program for calculating extreme pathways in biochemical reaction networks. *Bioinformatics* **21**, 1739–1740.

The ExPA homepage <http://gcr.g.ucsd.edu/Downloads/ExtremePathwayAnalysis>.

modelorg2tsv

Write an Instance of Class modelorg to File

Description

The function `modelorg2tsv` writes the content of an instance of class `modelorg` to text files in a character-separated value format adopted from the BiGG database output.

Usage

```
modelorg2tsv(model, prefix, suffix, extMetFlag = "b",
             fielddelim = "\t", entrydelim = ", ",
             makeClosedNetwork = FALSE,
             onlyReactionList = FALSE,
             minimalSet = FALSE,
             fpath = SYBIL_SETTINGS("PATH_TO_MODEL"), ...)
```

Arguments

model	An object of class <code>modelorg</code> .
prefix	A single character string giving the prefix for three possible output files (see Details below).
suffix	A single character string giving the file name extension. If missing, the value of suffix depends on the argument <code>fielddelim</code> , see Details below. Default: "tsv".
extMetFlag	A single character string giving the identifier for metabolites which are outside the system boundary. Only necessary, if the model is a closed one. Default: "b".
fielddelim	A single character string giving the value separator. Default: "\t".
entrydelim	A single character string giving the a separator for values containing more than one entry. Default: ", ".
makeClosedNetwork	Boolean. If set to TRUE, external metabolites (which are outside the system boundary) will be added to the model. These metabolites participate in reactions, transporting metabolites across the system boundary. The metabolite id will be the same as for the metabolite inside the system, but the compartment type is set to the value of argument <code>extMetFlag</code> . For example, most models contain a transport reaction for glucose: $glc[c] \rightleftharpoons$ If <code>makeClosedNetwork</code> is set to TRUE, this reaction will be written as $glc[c] \rightleftharpoons glc[b]$ with the letter b being the default value for <code>extMetFlag</code> . Default: FALSE.
onlyReactionList	Boolean. If set to TRUE, only one file containing all reaction equations will be produced (output file has one column). Default: FALSE.
minimalSet	Boolean. If set to TRUE, only one file containing the fields "abbreviation", "equation", "lowbnd", "uppbnd" and "obj_coef" will be produced (output file has five columns). Default: FALSE.
fpath	A single character string giving the path to a certain directory in which the output files will be stored. Default: <code>SYBIL_SETTINGS("PATH_TO_MODEL")</code> .
...	Further arguments passed to <code>write.table</code> , e.g. the Boolean argument <code>quote</code> can be used here.

Details

The function `modelorg2tsv` produces three output files: a reactions list, a metabolites list and a model description file.

The reactions list has the following columns:

```

"abbreviation"  react_id(model)
"name"          react_name(model)
"equation"      the reaction equations
"reversible"    react_rev(model)
"compartment"   reaction compartment(s)
"lowbnd"        lowbnd(model)
"uppbnd"        uppbnd(model)
"obj_coef"      obj_coef(model)
"rule"          gpr(model)
"subsystem"     subSys(model)

```

The metabolites list has the following columns:

```

"abbreviation"  met_id(model)
"name"          met_name(model)
"compartment"   met_comp(model)

```

The model description file has the following columns:

```

"name"          mod_name(model)
"id"            mod_id(model)
"description"    mod_desc(model)
"compartment"   mod_compart(model)
"abbreviation"  unique compartment abbreviations
"Nmetabolites"  number of metabolites
"Nreactions"    number of reactions
"Ngenes"        number of independent genes
"Nnnz"          number of non-zero elements in the stoichiometric matrix

```

If `onlyReactionList` is set to `TRUE`, only the reactions list containing the column "equation" is produced.

Please read the package vignette for detailed information about file formats and examples.

All fields in the output files are in double quotes. In order to read them in with `readTSVmod`, set argument `quoteChar` to `"\""`.

Value

Returns `TRUE` on success.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

References

The BiGG database <http://bigg.ucsd.edu/>.

See Also

[read.table](#), [modelorg2tsv](#), [modelorg](#).

modelorg_irrev-class *Class for Metabolic Networks in Irreversible Format.*

Description

Structure of the class "modelorg_irrev". Objects of that class are returned by the function [mod2irrev](#).

Objects from the Class

Objects can be created by calls of the function `modelorg_irrev`:

```
test <- modelorg_irrev(id = "foo", name = "bar").
```

Slots

`irrev`: Object of class "logical" indicating if the model is in irreversible format.

`matchrev`: Object of class "integer" matching of forward and backward reactions of a reversible reaction.

`rev2irrev`: Object of class "matrix" containing the reaction id's of the corresponding reactions in irreversible format.

`irrev2rev`: Object of class "integer" containing the reaction id's of the corresponding reaction in reversible format.

Extends

Class "[modelorg](#)", directly.

Methods

`irrev<-`: signature(object = "modelorg_irrev"): sets the irrev slot.

`irrev`: signature(object = "modelorg_irrev"): gets the irrev slot.

`matchrev<-`: signature(object = "modelorg_irrev"): sets the matchrev slot.

`matchrev`: signature(object = "modelorg_irrev"): gets the matchrev slot.

`rev2irrev<-`: signature(object = "modelorg_irrev"): sets the rev2irrev slot.

`rev2irrev`: signature(object = "modelorg_irrev"): gets the rev2irrev slot.

`irrev2rev<-`: signature(object = "modelorg_irrev"): sets the irrev2rev slot.

`irrev2rev`: signature(object = "modelorg_irrev"): gets the irrev2rev slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[modelorg](#)

Examples

```
showClass("modelorg_irrev")
```

multiDel

Parallel Support for sybil

Description

Parallel computation support for the functions [oneGeneDel](#), [doubleGeneDel](#), [oneFluxDel](#), [doubleFluxDel](#) and [fluxVar](#).

Usage

```
multiDel(model, nProc = 2, todo = "oneGeneDel", del1 = NA, del2 = NA, ...)
```

Arguments

model	An object of class modelorg .
nProc	Number of cores (processes) to use.
todo	A single character value giving the function name, which should be parallelised. Can be one of "oneGeneDel", "doubleGeneDel", "oneFluxDel", "doubleFluxDel" or "fluxVar".
del1	Vector of genes/reactions to consider.
del2	Vector of genes/reactions to consider (for use with doubleGeneDel or doubleFluxDel).
...	Further arguments passed to oneGeneDel , doubleGeneDel , oneFluxDel , doubleFluxDel or fluxVar .

Details

The function loads the package **parallel** if available. Argument nProc should be the number of cores to use. This number is verified via a call to detectCores (of **parallel**) and is set to the return value of detectCores, if nProc > detectCores() evaluates to TRUE. Arguments del1 and del2 are split into lists, each list element containing nProc/del1 elements. These are passed to [mclapply](#).

Value

A list of length nProc (or less, depending of the numbers of available cores), each element containing the return value of the function called (on object of a class extending [optsol](#)).

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[mclapply](#), [optsol](#), [oneGeneDel](#), [doubleGeneDel](#), [oneFluxDel](#), [doubleFluxDel](#) and [fluxVar](#).

Examples

```
## Not run:
## The examples here require the packages glpkAPI and parallel to be
## installed.

## perform single gene deletion analysis using the E. coli core
## metabolic model
data(Ec_core)
ad <- multiDel(Ec_core)
mapply(checkOptSol, ad)

## End(Not run)
```

netFlux-class

Class "netFlux"

Description

Class "netFlux" groups exchange reaction rates according to their sign in uptake, excretion and unused reactions.

Objects from the Class

Objects can be created by calls of the form `getNetFlux(rates, tol)`, with argument `rates` being a named numeric vector containing reaction rates of exchange fluxes and corresponding reaction id's. Argument `rates` can be obtained by a call to [optimizeProb](#). The second argument `tol` is a tolerance value (default: `SYBIL_SETTINGS("TOLERANCE")`). Reaction rates less than `tol * -1` are uptake reactions, reaction rates greater than `tol` are excretion reactions and all others (`abs(rates) < tol`) are unused reactions.

Slots

uptake: Object of class "logical" indicating uptake reactions.
product: Object of class "logical" indicating excretion reactions.
unused: Object of class "logical" indicating unused reactions.
react_id: Object of class "character" containing the reaction id's of the exchange reactions.
rate: Object of class "numeric" containing the reaction rates of the exchange reactions.

Methods

length signature(x = "netFlux"): number of exchange reactions.
rate signature(object = "netFlux"): gets the rate slot.
react_id signature(object = "netFlux"): gets the react_id slot.
react_id<- signature(object = "netFlux"): sets the react_id slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>
Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[optimizeProb](#), [getFluxDist](#)

Examples

```
data(Ec_core)
# retrieve all exchange reactions
ex <- findExchReact(Ec_core)
# perform flux balance analysis
opt <- optimizeProb(Ec_core, algorithm = "fba")
# get flux distribution of all exchange reactions
fd <- getFluxDist(opt, ex)
# group exchange reactions
getNetFlux(fd)
```

oneFluxDel

Single Flux Deletion Experiment

Description

Single reaction (flux) deletion analysis.

Usage

```
oneFluxDel(model, react = c(1:react_num(model)),
           lb = rep(0, length(react)),
           ub = rep(0, length(react)),
           checkOptSolObj = FALSE, ...)
```

Arguments

model	An object of class <code>modelorg</code> .
react	An object of class <code>reactId</code> or character or integer containing reaction id's to constrain to zero one by one. Default: all reactions present in argument model.
lb	A numeric vector of the same length as react containing the lower bounds for the reaction rates of reactions (variables) given in argument react. Default: 0 for all reactions in react, zero flux through all reactions.
ub	A numeric vector of the same length as react containing the lower bounds for the reaction rates of reactions (variables) given in argument react. Default: 0 for all reactions in react, zero flux through all reactions.
checkOptSolObj	A single logical value. If set to TRUE, a warning will be generated, if not all optimizations ended successful. Default: FALSE.
...	Further arguments passed to <code>optimizer</code> . Important ones are <code>algorithm</code> in order to set the algorithm to use or <code>solverParm</code> in order to set parameter values for the optimization software.

Details

The function `oneFluxDel` studies the effect of constraining single fluxes to zero flux rates on the phenotype of the metabolic network. The function performs n optimizations with n being the number of reaction id's given in argument `react`. Each optimization corresponds to the removal of one reaction.

Value

An object of class `optsol_fluxdel`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

`modelorg`, `optsol`, `optsol_fluxdel`, `checkOptSol`, `optimizer` and `SYBIL_SETTINGS`.

Examples

```
data(Ec_core)
Ec_ofd <- oneFluxDel(Ec_core)
```

oneGeneDel *Single Gene Deletion Experiment*

Description

Predict the metabolic phenotype of single-gene knock out mutants.

Usage

```
oneGeneDel(model, geneList,
           lb = rep(0, length(geneList)),
           ub = rep(0, length(geneList)),
           checkOptSolObj = FALSE, ...)
```

Arguments

model	An object of class modelorg .
geneList	A character vector containing the set of genes to be deleted one by one. Default: <code>allGenes(model)</code> .
lb	A numeric vector of the same length as <code>geneList</code> containing the lower bounds for the reaction rates of reactions (variables) affected by the genes given in argument <code>geneList</code> . Default: 0 for all genes in <code>geneList</code> , simulating knock-out mutants.
ub	A numeric vector of the same length as <code>geneList</code> containing the upper bounds for the reaction rates of reactions (variables) affected by the genes given in argument <code>geneList</code> . Default: 0 for all genes in <code>geneList</code> , simulating knock-out mutants.
checkOptSolObj	A single logical value. If set to TRUE, a warning will be generated, if not all optimizations ended successful. Default: FALSE.
...	Further arguments passed to optimizer . Important ones are <code>algorithm</code> in order to set the algorithm to use or <code>solverParm</code> in order to set parameter values for the optimization software.

Details

The function `oneGeneDel` studies the effect of genetic perturbations by single gene deletions on the phenotype of the metabolic network. The function performs n optimizations with n being the length of the character vector in argument `geneList`. For each gene deletion j the set of fluxes effected by the deletion of gene given in `geneList[j]` is constrained to zero flux. If the deletion of a certain gene has an effect, is tested with the function [geneDel](#). Each optimization corresponds to the deletion of one gene.

Value

An object of class `optsol_genedel`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

`modelorg`, `optsol`, `optsol_genedel`, `checkOptSol`, `optimizer` and `SYBIL_SETTINGS`.

Examples

```
# load example data set
data(Ec_core)

# compute phenotypes of genetic perturbations via
# FBA (default)
Ec_ogd <- oneGeneDel(Ec_core)

# or MOMA (linearized version)
Ec_ogd <- oneGeneDel(Ec_core, algorithm = "lmoma")
```

onlyChangeGPR

Change the GPR Rules

Description

Changes the GPR Rules for the chosen reactions

Usage

```
onlyChangeGPR(model, gprRules, reactNr, verboseMode = 0)
```

Arguments

<code>model</code>	An object of class <code>modelorg</code>
<code>gprRules</code>	character: contains logical expressions.
<code>reactNr</code>	An object of class <code>reactId</code> , a numeric vector, or a character vector containing reaction id's.
<code>verboseMode</code>	integer: verbosity level.

Details

The function changes the expressions for the chosen reactions.

Use `onlyCheckGPR` first to check the expressions.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

onlyCheckGPR

Check the GPR Rules

Description

Checks the GPR Rules for the chosen reactions

Usage

```
onlyCheckGPR(model, gprRules, reactNr, verboseMode = 1)
```

Arguments

model	An object of class modelorg
gprRules	character: contains logical expressions.
reactNr	An object of class reactId , a numeric vector, or a character vector containing reaction id's.
verboseMode	integer: verbosity level.

Details

The function checks the expressions for the chosen reactions.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

optimizeProb-methods *Optimize Problem Object*

Description

The generic optimizeProb performs the optimization of a mathematical programming object.

Usage

```
## S4 method for signature 'modelorg'  
optimizeProb(object,  
             algorithm = SYBIL_SETTINGS("ALGORITHM"),  
             gene = NULL,  
             react = NULL,  
             lb = NULL,  
             ub = NULL,  
             retOptSol = TRUE,  
             obj_coef = NULL,  
             lpdir = NULL,  
             mtfobj = NULL,  
             fldind = TRUE,  
             prCmd = NA,  
             poCmd = NA,  
             prCil = NA,  
             poCil = NA,  
             ...)
```

```
## S4 method for signature 'sysBiolAlg'  
optimizeProb(object,  
             react = NULL,  
             lb = NULL,  
             ub = NULL,  
             obj_coef = NULL,  
             lpdir = NULL,  
             fldind = TRUE,  
             resetChanges = TRUE,  
             prCmd = NA,  
             poCmd = NA,  
             prCil = NA,  
             poCil = NA)
```

Arguments

object	An object of class <code>modelorg</code> or <code>sysBiolAlg</code> .
algorithm	A single character string giving the name of the algorithm to use. See parameter "ALGORITHM" in <code>SYBIL_SETTINGS</code> for possible values. Default: <code>SYBIL_SETTINGS("ALGORITHM")</code> .

gene	A character or integer vector containing gene id's or indices of gene id's in <code>allGenes(model)</code> . If arguments <code>lb</code> and/or <code>ub</code> are additionally used (not <code>NULL</code>), upper and lower bounds will be applied to all fluxes on which the deletion of the genes given in <code>gene</code> have an effect. In this case, the first value in <code>lb</code> and <code>ub</code> is used. Default: <code>NULL</code> .
react	An object of class <code>reactId</code> , character or integer. Specifies the fluxes (variables) for which to change the upper and lower bound (see also arguments <code>lb</code> and <code>ub</code>) or objective coefficients (see also argument <code>obj_coef</code>). For class <code>sysBiolAlg</code> , it must be numeric. For class <code>modelorg</code> , setting <code>react</code> as no effect, if <code>gene</code> is also not <code>NULL</code> . Default: <code>NULL</code> .
lb	Numeric vector, must have the same length as <code>react</code> . Contains the new values for the lower bounds of fluxes (variables) mentioned in <code>react</code> . If set to <code>NULL</code> , lower bounds for variables in <code>react</code> will be left unchanged. For class <code>modelorg</code> : if <code>lb</code> is of length one, <code>lb</code> is used for all elements in <code>react</code> . Default: <code>NULL</code> .
ub	Same functionality as <code>lb</code> , but for upper bounds. Default: <code>NULL</code> .
obj_coef	Numeric vector, must have the same length as <code>react</code> . Contains the new values for the objective coefficients of fluxes (variables) mentioned in <code>react</code> . All other objective coefficients stay untouched. If set to <code>NULL</code> , objective coefficients for variables in <code>react</code> will be left unchanged. For class <code>modelorg</code> : if <code>obj_coef</code> is of length one, <code>obj_coef</code> is used for all elements in <code>react</code> . Default: <code>NULL</code> .
lpdir	Character value, direction of optimization. Can be set to "min" for minimization or "max" for maximization. Default: <code>SYBIL_SETTINGS("OPT_DIRECTION")</code> .
mtfobj	Only used, if argument <code>algorithm</code> is set to "mtf". A single numeric value giving a previously calculated optimized value of the objective function given in the model. The objective function of the model will be fixed to this value during optimization. If set to <code>NULL</code> , it will be computed by means of the "fba" algorithm. If additionally arguments <code>solver</code> and <code>method</code> are set, they will be used here too. Default: <code>NULL</code> .
fldind	Boolean value. If set to <code>TRUE</code> , (default) indices in "react" are used only for reactions. If set to <code>FALSE</code> , indices in "react" are used for all variables during optimization, e.g. also for additional variables introduced by the <code>mtf</code> algorithm. Currently unused by class <code>sysBiolAlg_room</code> . Default: <code>TRUE</code> .
resetChanges	Boolean value. If set to <code>TRUE</code> , (default) modifications of the problem object will be reset to their original values (e.g. changing upper and lower bounds for certain reactions). If set to <code>FALSE</code> , modifications will stay in the model. Default: <code>TRUE</code> .
prCmd	A list of preprocessing commands. See Details below. Default: <code>NA</code> .

poCmd	A list of postprocessing commands. See Details below. Default: NA.
prCil	Can be used if optimizeProb is called several times (like in optimizer). The argument prCil gets the value of the loop variable and passes it to the preprocessing function. There, one can access it via the keyword “LOOP_VAR”. See also optimizer . Default: NA.
poCil	Same as prCil, but for postprocessing. Default: NA.
retOptSol	Boolean. Return an object of class optsol_optimizeProb or just a list containing the results. Default: TRUE.
...	Only for the modelorg-method: further arguments passed to sysBiolAlg . See Details below.

Details

The arguments prCmd and poCmd can be used to execute R commands working on the problem object. All commands in prCmd are executed immediately before solving the problem; all commands in poCmd are executed after the problem has been solved. In all other aspects, the arguments work the same. The value of prCmd or poCmd are lists of character vectors (each list element is one command). Each command is a character vector and should be built as follows:

- The first element is the name of the function to call.
- All other elements are arguments to the function named in the first element.
- If any argument is character, enclose it in single quotes ' '.
- Use the keyword LP_PROB in order to refer to the variable name of the problem object (object of class [optObj](#)).
- If the length of the character vector is one, it is treated as a function call with the problem object (object of class [optObj](#)) as single argument.

The result will be an object of class [ppProc](#). A few examples for arguments prCmd or poCmd (all arguments must be lists, see examples section below):

```
sensitivityAnalysis
```

will be translated to the command

```
sensitivityAnalysis(LP_PROB)
```

with LP_PROB being the placeholder for the variable name of the problem object. The vector

```
c("writeProb", "LP_PROB", "'Ec_core.lp'", "'lp'")
```

will be translated to the command


```
writeProb(LP_PROB, 'Ec_core.lp', 'lp')
```

The first element will be the function name and the others the arguments to that function. The list of commands

```
list("sensitivityAnalysis",
     c("getDjCPLEX", "LP_PROB@oobj@env",
       "LP_PROB@oobj@lp", "0", "react_num(Ec_core)-1"
     )
)
```

will be translated to the commands

```
sensitivityAnalysis(LP_PROB)
getDjCPLEX(LP_PROB@oobj@env, LP_PROB@oobj@lp,
           0, react_num(Ec_core)-1)
```

For more information on the usage of prCmd and poCmd, see the examples section below.

The method optimizeProb for class `modelorg` generates a subclass of class `sysBio1Alg` and calls optimizeProb for that object again. Argument MoreArgs is used to transport arguments to the second optimizeProb call. Argument ... instead is used to transport arguments to the constructor function `sysBio1Alg`, for example algorithm, solver, method and solverParm. See `SYBIL_SETTINGS` for possible values.

Arguments gene, react, lb, ub and react cause changes in the problem object (object of class `optObj`, slot problem of class `sysBio1Alg`). These changes will be reset immediately after optimization if argument resetChanges is set to TRUE, otherwise changes will persist.

Value

Calls to optimizeProb returns either an object of class `optsol_optimizeProb` of length one if argument retOptSol is set to TRUE and object is of class `modelorg`, or a list containing the results of the optimization:

ok	Return value of the optimizer (e.g. "solution process was successful" or "time limit exceeded").
obj	Value of the objective function after optimization.
stat	Status value of the optimization (e.g. "solution is optimal" or "no feasible solution exists").
fluxes	The resulting flux distribution.
fldind	Pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the solution object represents reaction i in the original network.
preP	An object of class <code>ppProc</code> if a preprocessing command was given.
postP	An object of class <code>ppProc</code> if a postprocessing command was given.

Methods

signature(object = "modelorg") Translates the object of class `modelorg` into an object of class `sysBiolAlg` and calls `optimizeProb` again.

signature(object = "sysBiolAlg") Run optimization with the given problem object.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

`modelorg`, `applyChanges` and `sysBiolAlg`.

Examples

```
## Not run:
## The examples here require the package glpkAPI to be
## installed. If that package is not available, you have to set
## the argument 'solver' (the default is: solver = SYBIL_SETTINGS("SOLVER")).

## load the example data set
data(Ec_core)

## run optimizeProb(), Ec_sf will be an object of
## class optsol_optimizeProb
Ec_sf <- optimizeProb(Ec_core)

## run optimizeProb(), Ec_sf will be a list
Ec_sf <- optimizeProb(Ec_core, retOptSol = FALSE)

## do FBA, change the upper and lower bounds for the reactions
## "ATPM" and "PFK".
optimizeProb(Ec_core, react = c("ATPM", "PFK"),
             lb = c(3, -3), ub = c(5, 6))

## do FBA, perform sensitivity analysis after optimization
optimizeProb(Ec_core, poCmd = list("sensitivityAnalysis"))

## do FBA, write the problem object to file in lp-format
optimizeProb(Ec_core,
             poCmd = list(c("writeProb", "LP_PROB",
                           "'Ec_core.lp'", "'lp'")))

## do FBA, use "cplexAPI" as lp solver. Get all lower bounds before
## solving the problem. After solving, perform a sensitivity
## analysis and retrieve the reduced costs
opt <- optimizeProb(Ec_core, solver = "cplexAPI",
                   prCmd = list(c("getColsLowBnds", "LP_PROB", "1:77")),
                   poCmd = list("sensitivityAnalysis",
                                c("getDjCplex",
```

```

                                "LP_PROB@obj@env",
                                "LP_PROB@obj@lp",
                                "0", "react_num(Ec_core)-1"))
## get lower bounds
preProc(opt)
## get results of sensitivity analysis
postProc(opt)

## End(Not run)

```

optimizer

Performs Series of Optimizations

Description

The function `optimizer` is a wrapper to the `sysBiolAlg`-method `optimizeProb`. While `optimizeProb` runs one optimization, `optimizer` is designed to run a series of optimization by re-optimizing a given problem object (successive calls to `optimizeProb`).

Usage

```

optimizer(model, react, lb, ub, obj_coef, lpdire,
          algorithm = SYBIL_SETTINGS("ALGORITHM"),
          mtfobj = NULL,
          setToZero = FALSE,
          rebuildModel = FALSE,
          fld = "none",
          prCmd = NA, poCmd = NA,
          prDIR = NULL, poDIR = NULL,
          verboseMode = 2,
          ...)

```

Arguments

<code>model</code>	An object of class <code>modelorg</code> .
<code>react</code>	A list of numeric vectors. Each value must point to a reaction id present in <code>model</code> . The length of the list in <code>react</code> determines the number of optimizations to run. Each list element can be used in conjunction with arguments <code>lb</code> and <code>ub</code> or <code>obj_coef</code> and <code>lpdire</code> . The parameters given in this arguments will be set temporarily for each optimization.
<code>lb</code>	A numeric vector or list of the same length as <code>react</code> or a matrix with the number of rows equal to the length of <code>react</code> containing the lower bounds for the reaction rates of reactions (variables) given in argument <code>react</code> . If set to <code>NULL</code> , no lower bounds will be changed. If <code>lb</code> is a vector, <code>lb[k]</code> is used as lower bound for all reactions given in <code>react[k]</code> . If <code>lb</code> is a list, <code>lb[k]</code> must have the same length as <code>react[k]</code> . If <code>lb</code> is a matrix, each row serves as lower bound for the reactions given in each element of <code>react</code> (all elements in <code>react</code> must have the

	<p>same length).</p> <p>Default: NULL.</p>
ub	<p>A numeric vector or list of the same length as react or a matrix with the number of rows equal to the length of react containing the upper bounds for the reaction rates of reactions (variables) given in argument react. If set to NULL, no upper bounds will be changed. If ub is a vector, ub[k] is used as upper bound for all reactions given in react[k]. If ub is a list, ub[k] must have the same length as react[k]. If ub is a matrix, each row serves as upper bound for the reactions given in each element of react (all elements in react must have the same length).</p> <p>Default: NULL.</p>
obj_coef	<p>A numeric vector or list of the same length as react or a matrix with the number of rows equal to the length of react containing the objective coefficients for the reactions (variables) given in argument react. If set to NULL, no objective coefficients will be changed. If obj_coef is a vector, obj_coef[k] is used as objective coefficients for all reactions given in react[k]. If obj_coef is a list, obj_coef[k] must have the same length as react[k]. If obj_coef is a matrix, each row serves as objective coefficient for the reactions given in each element of react (all elements in react must have the same length).</p> <p>Default: NULL.</p>
lpdir	<p>A character vector of the same length as react containing the direction of optimization for each optimization. Possible values are "min" for minimization, or "max" for maximization. If set to NULL, optimization direction will not change.</p> <p>Default: NULL.</p>
algorithm	<p>A single character value giving the algorithm to compute genetic perturbations. Can be "fba": flux-balance analysis, "mtf": minimization of absolute total flux (see Details below), "moma": minimization of metabolic adjustment (MOMA), "lmoma": linear version of MOMA, "room": regulatory on/off minimization (ROOM) or "fv": flux variability analysis.</p> <p>Default: SYBIL_SETTINGS("ALGORITHM").</p>
mtfobj	<p>Only used, if argument algorithm is set to "mtf". A numeric vector of the same length as react containing previously calculated optimized values of the objective function given in the model. The objective function of the model will be fixed to this values in each optimization. If set to NULL, they will be computed by means of the "fba" algorithm. If additionally arguments solver and method are set, they will be used here too.</p> <p>Default: NULL.</p>
setToZero	<p>Logical: If the mathematical programming software returns a solution status which is not optimal, set the corresponding objective value to zero.</p> <p>Default: FALSE.</p>
rebuildModel	<p>Logical. If set to TRUE, the problem object will be rebuilt prior each round of optimization. Default: FALSE.</p>
fld	<p>Type of flux distribution to return. If set to "none", no flux distribution will be returned. If set to "fluxes", only the real flux distribution is returned, meaning all variable values after optimization representing a flux (reaction) in the model. If set to "all", all variable values are returned. If algorithm is set to "mtf"</p>

and fld equals "none", argument fld will be changed to "fluxes".
Default: "none".

prCmd	A list of preprocessing commands passed to <code>optimizeProb</code> . See there for details. Default: NA.
poCmd	A list of postprocessing commands passed to <code>optimizeProb</code> . See there for details. Default: NA.
prDIR	A numeric or character vector, indicating in which round of optimization the preprocessing command(s) will be executed. <code>prDIR = c(2, 5, 10)</code> executes the commands in <code>prCmd</code> before the second, 5th and 10th optimization. If <code>prDIR</code> is a character vector, for example <code>prDIR = c("10")</code> , the preprocessing commands given in <code>prCmd</code> will be executed every 10th round of optimization. If <code>prDIR</code> is character and has length 2, the first element is an offset to the following elements. <code>prDIR = c("-2", "10")</code> will do the preprocessing on every 10th round of optimization, beginning in round number $10 - 2 = 8$. Default: NULL.
poDIR	The same as <code>prDIR</code> , but for postprocessing. Default: NULL.
verboseMode	Single integer value, giving the amount of output to the console. Use <code>sink</code> to redirect output to a file. If <code>verboseMode == 1</code> status messages will be printed, if <code>verboseMode == 2</code> additionally a progress bar will be produced. If <code>verboseMode > 2</code> , intermediate results will be printed. Use <code>suppressMessages</code> to disable any output to the console. Default: 2.
...	Further arguments passed to <code>sysBiolAlg</code> .

Value

A list containing the results of the optimization:

solver	A single character string indicating the used mathematical programming software.
method	A single character string indicating the used optimization method by the mathematical programming software.
algorithm	A single character string indicating the used algorithm.
lp_num_cols	Number of columns (variables) in the problem object.
lp_num_rows	Number of rows (constraints) in the problem object.
obj	A numeric vector containing the values of the objective function after optimization.
ok	A numeric vector containing the return values of the optimizer (e.g. "solution process was successful" or "time limit exceeded").
stat	A numeric vector containing the status value of the optimization (e.g. "solution is optimal" or "no feasible solution exists").
lp_dir	A factor variable indicating the direction of optimization for each optimization.

fldind	Pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the solution object represents reaction i in the original network.
fluxdist	The resulting flux distribution.
prAna	An object of class <code>ppProc</code> if a preprocessing command was given.
poAna	An object of class <code>ppProc</code> if a postprocessing command was given.
alg_par	A named list of algorithm specific parameters.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

See Also

Class `sysBioAlg`, and constructor function `sysBioAlg`, `optimizeProb` and `SYBIL_SETTINGS`.

optObj

General Constructor Function For Objects of Class optObj

Description

This function serves as a user constructor function for objects of class `optObj`.

Usage

```
optObj(solver = SYBIL_SETTINGS("SOLVER"),
      method = SYBIL_SETTINGS("METHOD"),
      pType = "lp", prefix = "optObj", sep = "_")
```

Arguments

solver	A single character string giving the name of the solver package to use. See SYBIL_SETTINGS for possible values. Default: <code>SYBIL_SETTINGS("SOLVER")</code> .
method	A single character string containing the name of the method used by solver. See SYBIL_SETTINGS for possible values. If missing or not available, the default method for solver is used (see also checkDefaultMethod). Default: <code>SYBIL_SETTINGS("METHOD")</code> .
pType	A single character string containing the type of optimization problem. Can be "lp": linear programming, "mip": mixed integer programming or "qp": quadratic programming. Default: "lp".
prefix	A single character string containing a prefix for the new class name. Default: "optObj".
sep	A single character string containing a separator for prefix and solver. Default: "_".

Details

If argument `solver` is set to "foo" and `prefix` is set to "optObj" (default), `optObj` will try to build an instance of class `optObj_foo`. If `solver` does not contain a valid name of a solver package (this is checked by [checkDefaultMethod](#)), the default solver package will be used (see [SYBIL_SETTINGS](#)). For the name of the class, the arguments `prefix` and `solver` are stick together separated by `sep` (default: a single underscore "_"): `prefix_solver`.

Value

An instance of a subclass of class [optObj](#).

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Class [optObj](#), [SYBIL_SETTINGS](#) and [checkDefaultMethod](#).

optObj-class

Class "optObj"

Description

Structure of the class "optObj". Objects extending `optObj` returned by the constructor function `optObj`. These objects are used as part of class [sysBiolAlg](#).

Details

The intention of class `optObj` is, to provide a flexible user interface to several optimization software products. The methods here working on the slot `oobj` are interface functions to low level functions invoking corresponding C functions. Basically, the user has not to care about the nature of the solver, or solver-specific functions. That is done by the class.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`oobj`: Object of class "pointerToProb" containing a pointer to a problem object (see section Note).

`solver`: Object of class "character" containing the name of the solver software (see [SYBIL_SETTINGS](#) for suitable values).

`method`: Object of class "character" containing the method (algorithm) used by the solver software (see [SYBIL_SETTINGS](#) for suitable values).

`probType`: Object of class "character" giving the problem type (see `optObj` argument `pType` for suitable values).

Methods

`dim` signature(`x = "optObj"`): returns a vector `d` of length two with `d[1]` and `d[2]` containing the number of rows and columns of the constraint matrix.

`method` signature(`object = "optObj"`): gets the method slot.

`probType` signature(`object = "optObj"`): gets the `probType` slot.

`solver` signature(`object = "optObj"`): gets the solver slot.

Further usefull Functions

`checkSolStat`: `checkSolStat(stat, solver = SYBIL_SETTINGS("SOLVER"))`

Returns the indices of problems with a non-optimal solution status, or NA if it is not possible to retrieve a solution status.

`stat` Vector of integer values containing the solution status.

`solver` Single character string specifying the used solver (see [SYBIL_SETTINGS](#)).

`getMeanReturn`: `getMeanReturn(code, solver = SYBIL_SETTINGS("SOLVER"))`

Translates the return value (`code`) of a solver in a human readable string. Returns NA if the translation is not possible.

`getMeanStatus`: `getMeanStatus(code, solver = SYBIL_SETTINGS("SOLVER"), env = NULL)`

Translates the solution status value (`code`) of a solver in a human readable string. Returns NA if the translation is not possible. Argument `env` is for use with IBM ILOG CPLEX holding an object of class `cplexPtr` pointing to a IBM ILOG CPLEX environment.

`wrong_type_msg`: `wrong_type_msg(lp)`

prints a warning message, if slot `oobj` from `lp` (an instance of class `optObj`) does not contain a pointer to a valid solver. See also [SYBIL_SETTINGS](#) for possible solvers.

wrong_solver_msg: wrong_solver_msg(lp, method, printOut = TRUE)
if printOut == TRUE, it will print a warning message, if method is not available for solver in lp.

Additional methods used by classes extending class optObj

addCols: add columns to the problem object.
addRows: add rows to the problem object.
addRowsCols: add rows and columns to the problem object.
addColsToProb: add new columns (variables) to the problem object.
addRowsToProb: add new rows (constraints) to the problem object.
backupProb: copies a problem object into a new problem object.
changeColsBnds: change column (variable) bounds in the problem object.
changeColsBndsObjCoefs: change column (variable) bounds and objective coefficients in the problem object.
changeMatrixRow: change a row in the constraint matrix of the problem object.
changeObjCoefs: change objective coefficients in the problem object.
changeRowsBnds: change row bounds in the problem object.
delProb: delete (free) memory associated to the pointer to the problem object.
getColPrim: get primal value of variables after optimization.
getColsLowBnds: get lower bounds of variables.
getColsUppBnds: get upper bounds of variables.
getFluxDist: get all primal values of variables after optimization (resulting flux distribution).
getNumCols: get number of columns in the problem object.
getNumNnz: get number of non zero elements in the constraint matrix of the problem object.
getNumRows: get number of rows in the problem object.
getObjCoefs: get objective coefficients in the problem object.
getObjDir: get direction of optimization.
getObjVal: get value of the objective function after optimization.
getRedCosts: get reduced costs of all variables after optimization.
getRowsLowBnds: get lower row bounds of the problem object.
getRowsUppBnds: get lower bounds of the rows (constraints) of the problem object.
getSolStat: get solution status after optimization.
getSolverParm: get current parameter settings of the used solver.
initProb: initialize problem object.
loadLPprob: load data to the problem object. Use this method to generate problem objects.
loadQobj: load quadratic part of the objective function to the problem object.
readProb: read problem object from file (e.g. lp formatted).
scaleProb: scaling of the constraint matrix.

sensitivityAnalysis: perform sensitivity analysis.

setObjDir: set direction of optimization.

setRhsZero: set right hand side of the problem object to zero: $Sv = 0$.

setSolverParm: set parameters for the used solver.

solveLp: run optimization with the solver mentioned in slot **solver** and with the method given by slot **method**.

writeProb: write problem object to file (e.g. in lp format).

Note

The class pointerToProb contains an external pointer to a problem object (usually a C/C++ pointer). This is for **glpkAPI** an object of class **glpkPtr**, for **clpAPI** an object of class **externalptr**, for **lpSolveAPI** an object of class **lpExtPtr** and for **cplexAPI** an object of class **cplexPointer**.

The class **cplexPointer** has two slots **env** and **lp**, each of class **cplexPtr**. To access for example the environment pointer from an object of class **optObj**, one can write `lp@oobj@env`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

The constructor function **sysBio1Alg** for objects extending class **sysBio1Alg**; The constructor function **optObj**; **SYBIL_SETTINGS** and **checkDefaultMethod**.

Examples

```
showClass("optObj")
```

```
optObj_clpAPI-class   Class "optObj_clpAPI"
```

Description

Structure of the class "optObj_clpAPI".

Objects from the Class

Objects can be created by calls of the constructor function **optObj**:

```
test <- optObj(solver = "clpAPI").
```

Slots

- oobj: Object of class "pointerToProb" containing a pointer to a **clpAPI** problem object.
- solver: Object of class "character" containing the name of the solver software (see [SYBIL_SETTINGS](#) for suitable values).
- method: Object of class "character" containing the method (algorithm) used by the solver software (see [SYBIL_SETTINGS](#) for suitable values).
- probType: Object of class "character" giving the problem type (see [optObj](#) for suitable values).

Extends

Class "[optObj](#)", directly.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#)

Examples

```
showClass("optObj_c1pAPI")
```

```
optObj_cplexAPI-class  Class "optObj_cplexAPI"
```

Description

Structure of the class "optObj_cplexAPI".

Objects from the Class

Objects can be created by calls of the constructor function `optObj`:

```
test <- optObj(solver = "cplexAPI").
```

Slots

- oobj: Object of class "pointerToProb" containing a pointer to a **cplexAPI** problem object.
- solver: Object of class "character" containing the name of the solver software (see [SYBIL_SETTINGS](#) for suitable values).
- method: Object of class "character" containing the method (algorithm) used by the solver software (see [SYBIL_SETTINGS](#) for suitable values).
- probType: Object of class "character" giving the problem type (see [optObj](#) for suitable values).

Extends

Class "[optObj](#)", directly.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#)

Examples

```
showClass("optObj_cplexAPI")
```

```
optObj_glpkAPI-class  Class "optObj_glpkAPI"
```

Description

Structure of the class "[optObj_glpkAPI](#)".

Objects from the Class

Objects can be created by calls of the constructor function [optObj](#):

```
test <- optObj(solver = "glpkAPI").
```

Slots

oobj: Object of class "[pointerToProb](#)" containing a pointer to a **glpkAPI** problem object.

solver: Object of class "[character](#)" containing the name of the solver software (see [SYBIL_SETTINGS](#) for suitable values).

method: Object of class "[character](#)" containing the method (algorithm) used by the solver software (see [SYBIL_SETTINGS](#) for suitable values).

probType: Object of class "[character](#)" giving the problem type (see [optObj](#) for suitable values).

Extends

Class "[optObj](#)", directly.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#)

Examples

```
showClass("optObj_glpkAPI")
```

```
optObj_lpSolveAPI-class  
      Class "optObj_lpSolveAPI"
```

Description

Structure of the class "optObj_lpSolveAPI".

Objects from the Class

Objects can be created by calls of the constructor function [optObj](#):

```
test <- optObj(solver = "lpSolveAPI").
```

Slots

oobj: Object of class "pointerToProb" containing a pointer to a **lpSolveAPI** problem object.

solver: Object of class "character" containing the name of the solver software (see [SYBIL_SETTINGS](#) for suitable values).

method: Object of class "character" containing the method (algorithm) used by the solver software (see [SYBIL_SETTINGS](#) for suitable values).

probType: Object of class "character" giving the problem type (see [optObj](#) for suitable values).

Extends

Class "[optObj](#)", directly.

Further usefull Functions

return_codeLPSOLVE: (code) prints a human readable translation of return codes of **lpSolveAPI**.

loadMatrixPerColumnLPSOLVE: (lpmat, constMat) load a constraint matrix (an object of class [Matrix](#)) to a **lpSolveAPI** problem object column by column.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#)

Examples

```
showClass("optObj_lpSolveAPI")
```

optsol-class

Class optsol

Description

The class `optsol` provides data structures to store and access the results of optimizations. This class is extended by other classes and will not be used as is. The representation of class `optsol` is used as superclass.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`mod_id`: Object of class "character" containing the model id of the used model.

`mod_key`: Object of class "character" containing the model key of the used model.

`solver`: Object of class "character" indicating the used solver.

`method`: Object of class "character" indicating the used method.

`algorithm`: Object of class "character" containing the name of the algorithm used for optimizations.

`num_of_prob`: Object of class "integer" indicating the number of optimization problems.

`lp_num_cols`: Object of class "integer" indicating the number of columns.

`lp_num_rows`: Object of class "integer" indicating the number of rows.

`lp_obj`: Object of class "numeric" containing the optimal values of the objective function after optimization. If no flux distribution is available, slot `lp_obj` contains the cross-product of the objective coefficients in slot `obj_coef` and the part of the flux distribution in slot `fluxdist` containing the values representing fluxes in the entire metabolic network (slot `fldind`).

`lp_ok`: Object of class "integer" containing the exit code of the optimization.

`lp_stat`: Object of class "integer" containing the solution status of the optimization.

`lp_dir`: Object of class "character" indicating the direction of optimization.

`obj_coef`: Object of class "numeric" containing the objective coefficients of the used model (slot `obj_coef` of an object of class [modelorg](#)). These are not necessarily the objective coefficients of the used algorithm.

`obj_func`: Object of class "character" containing the objective function of the used model. Usually, it contains the return value of [printObjFunc](#).

fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the problem object represents reaction i in the original network.

fluxdist: Object of class "fluxDistribution" containing the solutions flux distributions.

alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Methods

algorithm<-: signature(object = "optsol"): sets the algorithm slot.

algorithm: signature(object = "optsol"): gets the algorithm slot.

alg_par signature(object = "optsol"): gets the alg_par slot.

alg_par<- signature(object = "optsol"): sets the alg_par slot.

checkStat: signature(opt = "optsol"): returns the indices of problems with a non optimal solution status.

fldind<-: signature(object = "optsol"): sets the fldind slot.

fldind: signature(object = "optsol"): gets the fldind slot.

fluxdist<-: signature(object = "optsol"): sets the fluxdist slot.

fluxdist: signature(object = "optsol"): gets the fluxdist slot.

fluxes<-: signature(object = "optsol"): sets the fluxes slot of slot fluxdist.

fluxes: signature(object = "optsol"): gets the fluxes slot of slot fluxdist.

plot: signature(x = "optsol"): plots a [histogram](#) of the values of the objective function given in the model in optimal state. Additional arguments can be passed to [histogram](#) via the ... argument.

length: signature(x = "optsol"): returns the number of optimizations.

lp_dir<-: signature(object = "optsol", value = "character"): sets the lp_dir slot. Argument value can be "min" (minimization) or "max" (maximization).

lp_dir<-: signature(object = "optsol", value = "factor"): sets the lp_dir slot.

lp_dir<-: signature(object = "optsol", value = "numeric"): sets the lp_dir slot. Argument value can be 1 (minimization) or -1 (maximization).

lp_dir: signature(object = "optsol"): gets the lp_dir slot.

lp_num_cols<-: signature(object = "optsol"): sets the lp_num_cols slot.

lp_num_cols: signature(object = "optsol"): gets the lp_num_cols slot.

lp_num_rows<-: signature(object = "optsol"): sets the lp_num_rows slot.

lp_num_rows: signature(object = "optsol"): gets the lp_num_rows slot.

lp_obj<-: signature(object = "optsol"): sets the lp_obj slot.

lp_obj: signature(object = "optsol"): gets the lp_obj slot.

lp_ok<-: signature(object = "optsol"): sets the lp_ok slot.

lp_ok: signature(object = "optsol"): gets the lp_ok slot.

lp_stat<-: signature(object = "optsol"): sets the lp_stat slot.

lp_stat: signature(object = "optsol"): gets the lp_stat slot.

`method<-`: signature(object = "optsol"): sets the method slot.
`method`: signature(object = "optsol"): gets the method slot.
`mod_id<-`: signature(object = "optsol"): sets the mod_id slot.
`mod_id`: signature(object = "optsol"): gets the mod_id slot.
`mod_key<-`: signature(object = "optsol"): sets the mod_key slot.
`mod_key`: signature(object = "optsol"): gets the mod_key slot.
`mod_obj`: signature(object = "optsol_fluxdel"): returns always the cross-product of the objective coefficients in slot `obj_coef` and the part of the flux distribution in slot `fluxdist` containing the values representing fluxes in the entire metabolic network (slot `fldind`). If slot `obj_coef` is NA, the content of slot `lp_obj` is returned. In contrast, method `lp_obj` always returns the value of the objective function of the used algorithm after optimization.
`nfluxes`: signature(object = "optsol"): gets the number of elements in the flux distribution matrix.
`num_of_prob<-`: signature(object = "optsol"): sets the num_of_prob slot.
`num_of_prob`: signature(object = "optsol"): gets the num_of_prob slot.
`obj_coef<-`: signature(object = "optsol"): sets the obj_coef slot.
`obj_coef`: signature(object = "optsol"): gets the obj_coef slot.
`obj_func<-`: signature(object = "optsol"): sets the obj_func slot.
`obj_func`: signature(object = "optsol"): gets the obj_func slot.
`react_id<-`: signature(object = "optsol"): sets the react_id slot.
`react_id`: signature(object = "optsol"): gets the react_id slot.
`show`: signature(object = "optsol"): prints a summary of the content of instance of class `optsol`.
`solver<-`: signature(object = "optsol"): sets the solver slot.
`solver`: signature(object = "optsol"): gets the solver slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkOptSol](#), [optsol_optimizeProb](#), [optsol_fluxdel](#), [optsol_genedel](#), [optsol_robAna](#) and [optsol_fluxVar](#)

Examples

```
showClass("optsol")
```

```
optsol_blockedReact-class
      Class "optsol_blockedReact"
```

Description

Structure of the class "optsol_blockedReact". Objects of that class are returned by the function [blockedReact](#).

Objects from the Class

Objects can be created by calls of the form `new("optsol_blockedReact", ...)`.

Slots

blocked: Object of class "logical" indicating if a reaction is blocked, or not.

react: Object of class "reactId" containing the reaction id's of checked reactions.

mod_id: Object of class "character" containing the model id of the used model.

mod_key: Object of class "character" containing the model key of the used model.

solver: Object of class "character" indicating the used solver.

method: Object of class "character" indicating the used method.

algorithm: Object of class "character" containing the name of the algorithm used for optimizations.

num_of_prob: Object of class "integer" indicating the number of optimization problems.

lp_num_cols: Object of class "integer" indicating the number of columns.

lp_num_rows: Object of class "integer" indicating the number of rows.

lp_obj: Object of class "numeric" containing the optimal values of the objective function after optimization. If no flux distribution is available, slot `lp_obj` contains the cross-product of the objective coefficients in slot `obj_coef` and the part of the flux distribution in slot `fluxdist` containing the values representing fluxes in the entire metabolic network (slot `fldind`).

lp_ok: Object of class "integer" containing the exit code of the optimization.

lp_stat: Object of class "integer" containing the solution status of the optimization.

lp_dir: Object of class "character" indicating the direction of optimization.

obj_coef: Object of class "numeric" containing the objective coefficients of the used model (slot `obj_coef` of an object of class [modelorg](#)). These are not necessarily the objective coefficients of the used algorithm.

obj_func: Object of class "character" containing the objective function of the used model. Usually, it contains the return value of [printObjFunc](#).

fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.

fluxdist: Object of class "fluxDistribution" containing the solutions flux distributions.

alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "optsol", directly.

Methods

`blocked`: signature(object = "optsol_blockedReact"): gets the blocked slot.

`blocked<-`: signature(object = "optsol_blockedReact") sets the blocked slot.

`react`: signature(object = "optsol_blockedReact"): gets the react slot.

`react<-`: signature(object = "optsol_blockedReact") sets the react slot.

`maxSol`: signature(object = "optsol_blockedReact")(slot): returns the values in the slot given in slot for optimizations in "max" direction.

`minSol`: signature(object = "optsol_blockedReact")(slot): returns the values in the slot given in slot for optimizations in "min" direction.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkOptSol](#) and [optsol](#)

Examples

```
showClass("optsol_blockedReact")
```

optsol_fluxdel-class *Class "optsol_fluxdel"*

Description

Structure of the class "optsol_fluxdel". Objects of that class are returned by the function [oneFluxDel](#).

Objects from the Class

Objects can be created by calls of the form `new("optsol_fluxdel", ...)`.

Slots

- chlb:** Object of class "numeric" containing the new (changed) values for the columns lower bounds.
- chub:** Object of class "numeric" containing the new (changed) values for the columns upper bounds.
- dels:** Object of class "matrix" containing the reaction id's of constrained reactions. Each row of the matrix represents one set of simultaneously constrained reactions.
- preProc:** Object of class "ppProc" containing the results of pre-processing. See also [optimizeProb](#).
- postProc:** Object of class "ppProc" containing the results of post-processing. See also [optimizeProb](#).
- mod_id:** Object of class "character" containing the model id of the used model.
- mod_key:** Object of class "character" containing the model key of the used model.
- solver:** Object of class "character" indicating the used solver.
- method:** Object of class "character" indicating the used method.
- algorithm:** Object of class "character" containing the name of the algorithm used for optimizations.
- num_of_prob:** Object of class "integer" indicating the number of optimization problems.
- lp_num_cols:** Object of class "integer" indicating the number of columns.
- lp_num_rows:** Object of class "integer" indicating the number of rows.
- lp_obj:** Object of class "numeric" containing the optimal values of the objective function after optimization. If no flux distribution is available, slot `lp_obj` contains the cross-product of the objective coefficients in slot `obj_coef` and the part of the flux distribution in slot `fluxdist` containing the values representing fluxes in the entire metabolic network (slot `fldind`).
- lp_ok:** Object of class "integer" containing the exit code of the optimization.
- lp_stat:** Object of class "integer" containing the solution status of the optimization.
- lp_dir:** Object of class "character" indicating the direction of optimization.
- obj_coef:** Object of class "numeric" containing the objective coefficients of the used model (slot `obj_coef` of an object of class [modelorg](#)). These are not necessarily the objective coefficients of the used algorithm.
- obj_func:** Object of class "character" containing the objective function of the used model. Usually, it contains the return value of [printObjFunc](#).
- fldind:** Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.
- fluxdist:** Object of class "fluxDistribution" containing the solutions flux distributions.
- alg_par:** Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "[optsol_optimizeProb](#)", directly. Class "[optsol](#)", by class "[optsol_optimizeProb](#)", distance 2.

Methods

`react_id`: signature(object = "optsol_fluxdel"): gets the `react_id` slot.
`react_id<-`: signature(object = "optsol_fluxdel") sets the `react_id` slot.
`allGenes`: signature(object = "optsol_fluxdel"): gets the `allGenes` slot.
`allGenes<-`: signature(object = "optsol_fluxdel") sets the `allGenes` slot.
`chlb`: signature(object = "optsol_fluxdel"): gets the `chlb` slot.
`chlb<-`: signature(object = "optsol_fluxdel") sets the `chlb` slot.
`chub`: signature(object = "optsol_fluxdel"): gets the `chub` slot.
`chub<-`: signature(object = "optsol_fluxdel") sets the `chub` slot.
`dels`: signature(object = "optsol_fluxdel"): gets the `dels` slot.
`dels<-`: signature(object = "optsol_fluxdel") sets the `dels` slot.
`algorithm`: signature(object = "optsol_fluxdel"): gets the `algorithm` slot.
`algorithm<-`: signature(object = "optsol_fluxdel") sets the `algorithm` slot.
`lethal`: signature(object = "optsol_fluxdel")(wt, tol): returns a logical vector of length `num_of_prob(object)`. Argument `wt` is an optimal (wild type) growth rate, e.g. computed via FBA. If the absolute growth ratio (`mod_obj(object)/wt`) of knock-out `i` is less than `tol`, the deletion is considered as lethal. If `lethal(object)[i]` is TRUE, deletion `[i]` is lethal.
`deleted`: signature(object = "optsol_fluxdel")(i): gets the `i`th element of the `dels` slot.
`[`: signature(x = "optsol_fluxdel"): access like a vector. `x[i]` returns a new object of class `optsol_fluxdel` containing the `i`th deletion experiment.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

[checkOptSol](#), [optsol](#), [optsol_genedel](#) and [optsol_optimizeProb](#)

Examples

```
showClass("optsol_fluxdel")
```

optsol_fluxVar-class *Class "optsol_fluxVar"*

Description

Structure of the class "optsol_fluxVar". Objects of that class are returned by the function [fluxVar](#).

Objects from the Class

Objects can be created by calls of the form `new("optsol_fluxVar", ...)`.

Slots

react: Object of class "reactId" containing reaction id's for which ranges were calculated.

preProc: Object of class "ppProc" containing the results of pre-processing. See also [optimizeProb](#).

postProc: Object of class "ppProc" containing the results of post-processing. See also [optimizeProb](#).

mod_id: Object of class "character" containing the model id of the used model.

mod_key: Object of class "character" containing the model key of the used model.

solver: Object of class "character" indicating the used solver.

method: Object of class "character" indicating the used method.

algorithm: Object of class "character" containing the name of the algorithm used for optimizations.

num_of_prob: Object of class "integer" indicating the number of optimization problems.

lp_num_cols: Object of class "integer" indicating the number of columns.

lp_num_rows: Object of class "integer" indicating the number of rows.

lp_obj: Object of class "numeric" containing the optimal values of the objective function after optimization.

lp_ok: Object of class "integer" containing the exit code of the optimization.

lp_stat: Object of class "integer" containing the solution status of the optimization.

lp_dir: Object of class "character" indicating the direction of optimization.

obj_coef: Object of class "numeric" containing the objective coefficients of the used model (slot `obj_coef` of an object of class [modelorg](#)). These are not necessarily the objective coefficients of the used algorithm.

obj_func: Object of class "character" containing the objective function of the used model. Usually, it contains the return value of [printObjFunc](#).

fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.

fluxdist: Object of class "fluxDistribution" containing the solutions flux distributions.

alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "[optsol_optimizeProb](#)", directly. Class "[optsol](#)", by class "optsol_optimizeProb", distance 2.

Methods

`react`: signature(object = "optsol_fluxVar"): gets the react slot.

`react<-`: signature(object = "optsol_fluxVar"): sets the react slot.

`maxSol`: signature(object = "optsol_fluxVar")(slot): returns the values in the slot given in slot for optimizations in "max" direction.

`minSol`: signature(object = "optsol_fluxVar")(slot): returns the values in the slot given in slot for optimizations in "min" direction.

`plot` signature(x = "optsol_fluxVar", y = "missing")(ylim, xlab = "", ylab = "Value", pch = 20, col = "black"): plots the range of values each flux can have still giving an optimal objective function value.

`ylim` scaling of y-axis, if missing, the maximum and minimum value of all optimizations is used (rounded to the next smaller/larger integer value).

`xlab` label of x-axis, see also [par](#).

`ylab` label of y-axis, see also [par](#).

`pch` how to plot the points, see also [par](#).

`col` color of the plot, see also [par](#).

`collower` color of the minimum range value. Default `col`.

`colupper` color of the maximum range value. Default `col`.

`pchupper` how to plot the point for the maximum range value. Default `pch`.

`pchlower` how to plot the point for the minimum range value. Default `pch`.

`dottedline` if set to `FALSE`, from each minimum range value a dotted line to the corresponding x-axis label will be plotted. Default `FALSE`.

`baseline` plot a horizontal dashed line at the value of `baseline`. Default `0`. If set to `NA`, no baseline will be plotted.

`connect` if set to `TRUE`, a solid connecting line will be drawn between the minimum and maximum value of one reaction. Default `TRUE`.

`colconnect` color of the connecting line. Default "black".

`...` further arguments to the [plot](#) function.

`plotRangeVar` signature(object = "optsol_fluxVar") (...): plot a histogram of the span of the minimum and maximum range values for each flux.

`...` further arguments to the [hist](#) function.

`blReact` signature(object = "optsol_fluxVar") (tol = SYBIL_SETTINGS("TOLERANCE")): returns a logical vector of length equal to the number of reactions analyzed during flux variance analysis (number of optimizations divided by two). If `blReact(object)[j]` equals `TRUE`, reaction `j` is considered to be blocked (zero flux rate) given the used conditions. A reaction `j` is considered to be 'blocked', if its calculated range of reaction rates does not exceed $0 \pm \text{tol}$.

`tol` limit of tolerance.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkOptSol](#) and [optsol](#)

Examples

```
showClass("optsol_fluxVar")
```

optsol_genedel-class *Class "optsol_genedel"*

Description

Structure of the class "optsol_genedel". Objects of that class are returned by the function geneDel.

Objects from the Class

Objects can be created by calls of the form new("optsol_genedel", ...).

Slots

fluxdels: Object of class "list" containing the reaction id's of constrained reactions (fluxes).
 fluxdels(optsol_genedel)[[i]][j] = 1: The deletion of gene i requires the deletion of a set of fluxes 1..k ($j \leq k$), j being the j'th reaction of that set.

hasEffect: Object of class "logical" indicating whether deletion of gene i has an effect or not. This is determined on basis of the gprRules and not by optimizations.

chlb: Object of class "numeric" containing the new (changed) values for the columns lower bounds.

chub: Object of class "numeric" containing the new (changed) values for the columns upper bounds.

dels: Object of class "matrix" containing the gene id of constrained genes. Each row of the matrix represents one set of simultaneously constrained genes.

preProc: Object of class "ppProc" containing the results of pre-processing. See also [optimizeProb](#).

postProc: Object of class "ppProc" containing the results of post-processing. See also [optimizeProb](#).

mod_id: Object of class "character" containing the model id of the used model.

mod_key: Object of class "character" containing the model key of the used model.

solver: Object of class "character" indicating the used solver.

method: Object of class "character" indicating the used method.

algorithm: Object of class "character" containing the name of the algorithm used for optimizations.

num_of_prob: Object of class "integer" indicating the number of optimization problems.
lp_num_cols: Object of class "integer" indicating the number of columns.
lp_num_rows: Object of class "integer" indicating the number of rows.
lp_obj: Object of class "numeric" containing the optimal values of the objective function after optimization. If no flux distribution is available, slot lp_obj contains the cross-product of the objective coefficients in slot obj_coef and the part of the flux distribution in slot fluxdist containing the values representing fluxes in the entire metabolic network (slot fldind).
lp_ok: Object of class "integer" containing the exit code of the optimization.
lp_stat: Object of class "integer" containing the solution status of the optimization.
lp_dir: Object of class "character" indicating the direction of optimization.
obj_coef: Object of class "numeric" containing the objective coefficients of the used model (slot obj_coef of an object of class `modelorg`). These are not necessarily the objective coefficients of the used algorithm.
obj_func: Object of class "character" containing the objective function of the used model. Usually, it contains the return value of `printObjFunc`.
fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the problem object represents reaction i in the original network.
fluxdist: Object of class "fluxDistribution" containing the solutions flux distributions.
alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "`optsol_fluxdel`", directly. Class "`optsol_optimizeProb`", by class "`optsol_fluxdel`", distance 2. Class "`optsol`", by class "`optsol_fluxdel`", distance 3.

Methods

fluxdels: signature(object = "optsol_genedel"): gets the fluxdels slot.
fluxdels<-: signature(object = "optsol_genedel") sets the fluxdels slot.
hasEffect: signature(object = "optsol_genedel"): gets the hasEffect slot.
hasEffect<-: signature(object = "optsol_genedel"): sets the hasEffect slot.
deleted: signature(object = "optsol_genedel")(i): gets the ith element of the dels slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

`checkOptSol`, `optsol`, `optsol_fluxdel` and `optsol_optimizeProb`

Examples

```
showClass("optsol_genedel")
```

```
optsol_optimizeProb-class
      Class "optsol_optimizeProb"
```

Description

Structure of the class "optsol_optimizeProb". Objects of that class are returned by the function [optimizeProb](#) with the argument `retOptSol` set to TRUE.

Objects from the Class

Objects can be created by calls of the form `new("optsol_optimizeProb", ...)`, or via the constructor function [makeOptSolM0](#).

Slots

`preProc`: Object of class "ppProc" containing the results of pre-processing. See also [optimizeProb](#).
`postProc`: Object of class "ppProc" containing the results of post-processing. See also [optimizeProb](#).
`mod_id`: Object of class "character" containing the model id of the used model.
`mod_key`: Object of class "character" containing the model key of the used model.
`solver`: Object of class "character" indicating the used solver.
`method`: Object of class "character" indicating the used method.
`algorithm`: Object of class "character" containing the name of the algorithm used for optimizations.
`num_of_prob`: Object of class "integer" indicating the number of optimization problems.
`lp_num_cols`: Object of class "integer" indicating the number of columns.
`lp_num_rows`: Object of class "integer" indicating the number of rows.
`lp_obj`: Object of class "numeric" containing the optimal values of the objective function after optimization. If no flux distribution is available, slot `lp_obj` contains the cross-product of the objective coefficients in slot `obj_coef` and the part of the flux distribution in slot `fluxdist` containing the values representing fluxes in the entire metabolic network (slot `fldind`).
`lp_ok`: Object of class "integer" containing the exit code of the optimization.
`lp_stat`: Object of class "integer" containing the solution status of the optimization.
`lp_dir`: Object of class "character" indicating the direction of optimization.
`obj_coef`: Object of class "numeric" containing the objective coefficients of the used model (slot `obj_coef` of an object of class [modelorg](#)). These are not necessarily the objective coefficients of the used algorithm.
`obj_func`: Object of class "character" containing the objective function of the used model. Usually, it contains the return value of [printObjFunc](#).
`fldind`: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.
`fluxdist`: Object of class "fluxDistribution" containing the solutions flux distributions.
`alg_par`: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "[optsol](#)", directly.

Methods

preProc: signature(object = "optsol_optimizeProb"): gets the preProc slot.
 preProc<=: signature(object = "optsol_optimizeProb"): sets the preProc slot.
 postProc: signature(object = "optsol_optimizeProb"): gets the postProc slot.
 postProc<=: signature(object = "optsol_optimizeProb"): sets the postProc slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkOptSol](#), [optsol](#), [optsol_genedel](#) and [optsol_fluxdel](#)

Examples

```
showClass("optsol_optimizeProb")
```

optsol_phpp-class	Class "optsol_phpp"
-------------------	---------------------

Description

Structure of the class "optsol_robAna". Objects of that class are returned by the function [phpp](#).

Objects from the Class

Objects can be created by calls of the form `new("optsol_phpp", ...)`.

Slots

ctrlflm: Object of class "matrix" containing the control flux values.
 redCosts: Object of class "matrix" containing the reduced costs of the two control flux values.
 ctrlr: Object of class "reactId" containing the reaction id of the control reaction.
 ctrlfl: Object of class "numeric" unused, see ctrlflm.
 preProc: Object of class "ppProc" containing the results of pre-processing. See also [optimizeProb](#).
 postProc: Object of class "ppProc" containing the results of post-processing. See also [optimizeProb](#).
 mod_id: Object of class "character" containing the model id of the used model.
 mod_key: Object of class "character" containing the model key of the used model.

solver: Object of class "character" indicating the used solver.

method: Object of class "character" indicating the used method.

algorithm: Object of class "character" containing the name of the algorithm used for optimizations.

num_of_prob: Object of class "integer" indicating the number of optimization problems.

lp_num_cols: Object of class "integer" indicating the number of columns.

lp_num_rows: Object of class "integer" indicating the number of rows.

lp_obj: Object of class "numeric" containing the optimal values of the objective function after optimization. If no flux distribution is available, slot lp_obj contains the cross-product of the objective coefficients in slot obj_coef and the part of the flux distribution in slot fluxdist containing the values representing fluxes in the entire metabolic network (slot fldind).

lp_ok: Object of class "integer" containing the exit code of the optimization.

lp_stat: Object of class "integer" containing the solution status of the optimization.

lp_dir: Object of class "character" indicating the direction of optimization.

obj_coef: Object of class "numeric" containing the objective coefficients of the used model (slot obj_coef of an object of class `modelorg`). These are not necessarily the objective coefficients of the used algorithm.

obj_func: Object of class "character" containing the objective function of the used model. Usually, it contains the return value of `printObjFunc`.

fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the problem object represents reaction i in the original network.

fluxdist: Object of class "fluxDistribution" containing the solutions flux distributions.

alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "optsol_robAna", directly. Class "optsol_optimizeProb", by class "optsol_robAna", distance 2. Class "optsol", by class "optsol_robAna", distance 3.

Methods

ctrlfl signature(object = "optsol_phpp"): gets the ctrlflm slot.

ctrlfl<- signature(object = "optsol_phpp"): sets the ctrlflm slot.

getRedCosts signature(lp = "optsol_phpp"): gets the ctrlflm slot.

plot signature(x = "optsol_phpp", y = "character"): (main = paste("Reduced Costs:", y), xlab = y, ylab = y, shrink = 0.5, col.regions = "grey50") plots the reduced costs of the control fluxes as `levelplot`.

y reaction id of one control reaction.

main plot title, see also `levelplot`.

xlab label of x-axis, see also `levelplot`.

ylab label of y-axis, see also `levelplot`.

shrink scale of rectangles to plot, see `levelplot`.

col.regions a vector of colors (default greyscale) see `levelplot`.

... further graphical parameters to the [levelplot](#) function.

plot signature(x = "optsol_phpp", y = "missing"): (xlab = list(label = react_id(ctrlr(x)[1]), rot = 30, grey(w * irr + (1 - w) * (1-(1-ref)^0.75)) }, ...): plots the optimal values of the objective function vs. the control flux values in a [wireframe](#) plot.

xlab label of x-axis, see also [wireframe](#).

ylab label of y-axis, see also [wireframe](#).

zlab label of z-axis, see also [wireframe](#).

scales parameters describing scales, see [wireframe](#).

par.settings additional parameters, see [wireframe](#).

shade enable/disable shading, see [wireframe](#).

shade.colors a function for the shading color (default greyscale), see [wireframe](#).

... further graphical parameters to the [wireframe](#) function.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[phpp](#), [checkOptSol](#) and [optsol](#)

Examples

```
showClass("optsol_phpp")
```

```
optsol_robAna-class   Class "optsol_robAna"
```

Description

Structure of the class "optsol_robAna". Objects of that class are returned by the function [robAna](#).

Objects from the Class

Objects can be created by calls of the form `new("optsol_robAna", ...)`.

Slots

ctrlr: Object of class "reactId" containing the reaction id of the control reaction.

ctrlfl: Object of class "numeric" containing the control flux values.

preProc: Object of class "ppProc" containing the results of pre-processing. See also [optimizeProb](#).

postProc: Object of class "ppProc" containing the results of post-processing. See also [optimizeProb](#).

mod_id: Object of class "character" containing the model id of the used model.

mod_key: Object of class "character" containing the model key of the used model.
solver: Object of class "character" indicating the used solver.
method: Object of class "character" indicating the used method.
algorithm: Object of class "character" containing the name of the algorithm used for optimizations.
num_of_prob: Object of class "integer" indicating the number of optimization problems.
lp_num_cols: Object of class "integer" indicating the number of columns.
lp_num_rows: Object of class "integer" indicating the number of rows.
lp_obj: Object of class "numeric" containing the optimal values of the objective function after optimization. If no flux distribution is available, slot `lp_obj` contains the cross-product of the objective coefficients in slot `obj_coef` and the part of the flux distribution in slot `fluxdist` containing the values representing fluxes in the entire metabolic network (slot `fldind`).
lp_ok: Object of class "integer" containing the exit code of the optimization.
lp_stat: Object of class "integer" containing the solution status of the optimization.
lp_dir: Object of class "character" indicating the direction of optimization.
obj_coef: Object of class "numeric" containing the objective coefficients of the used model (slot `obj_coef` of an object of class `modelorg`). These are not necessarily the objective coefficients of the used algorithm.
obj_func: Object of class "character" containing the objective function of the used model. Usually, it contains the return value of `printObjFunc`.
fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.
fluxdist: Object of class "fluxDistribution" containing the solutions flux distributions.
alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "`optsol_optimizeProb`", directly. Class "`optsol`", by class "`optsol_optimizeProb`", distance 2.

Methods

ctrlfl: signature(object = "optsol_robAna"): gets the `ctrlfl` slot.
ctrlfl<-: signature(object = "optsol_robAna"): sets the `ctrlfl` slot.
ctrllr: signature(object = "optsol_robAna"): gets the `ctrllr` slot.
ctrllr<-: signature(object = "optsol_robAna"): sets the `ctrllr` slot.
plot signature(x = "optsol_robAna", y = "missing")(xlab = paste("Control Flux:", react_id(ctrllr(x))), plots the optimal values of the objective function vs. the control flux values.
xlab label of x-axis, see also `par`.
ylab label of y-axis, see also `par`.
type plot type, see also `par`.

pch how to plot the points, see also [par](#).
 fillColorBg color of the area below the curve.
 fillBg logical: color the area below the curve.
 absCtrl if set to TRUE, the control flux values (x axis) are plotted as absolute values.
 ... further graphical parameters to the [points](#) function.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[robAna](#), [checkOptSol](#) and [optsol](#)

Examples

```
showClass("optsol_robAna")
```

 phpp

Phenotypic Phase Plane Analysis

Description

Performs phenotypic phase plane analysis for a given metabolic model.

Usage

```
phpp(model, ctrlreact, rng = c(0, 0, 20, 20),
      numP = 50, setToZero = TRUE, redCosts = FALSE, ...)
```

Arguments

model	An object of class modelorg .
ctrlreact	An object of class reactId , character or integer. Specifies two control reactions.
rng	A numeric vector of length four, giving the lower and upper bounds of the control reactions. The first two values contain the lower bounds, the last two values the upper bounds. Default: <code>c(0, 0, 20, 20)</code>
numP	The number of points to analyse. Default: 50
setToZero	Logical: If the mathematical programming software returns a solution status which is not optimal, set the corresponding objective value to zero (see also optimizer). Default: TRUE.
redCosts	Logical: store reduced costs of the control variables. Default: FALSE.
...	Further arguments passed to optimizer .

Details

The two control reactions given in argument `ctrlreact` are treated as uptake reactions: reactions that transport metabolites into the metabolic network. That means, the optimizations are performed using `abs(rng) * -1`.

Value

An object of class `optsol_phpp`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

Edwards, J. S., Ibarra, R. U. and Palsson, B. Ø. (2001) In silico predictions of *Escherichia coli* metabolic capabilities are consistent with experimental data. *Nat Biotechnol* **19**, 125–130.

Edwards, J. S., Ramakrishna, R. and Palsson, B. Ø. (2002) Characterizing the metabolic phenotype: a phenotype phase plane analysis. *Biotechnol Bioeng* **77**, 27–36.

Bernhard Ø. Palsson (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press.

Examples

```
data(Ec_core)

# switch off glucose input
Ec_core_wo_glc <- changeUptake(Ec_core, off = "glc_D[e]")
opt <- phpp(Ec_core_wo_glc, ctrlreact = c("EX_succ(e)", "EX_o2(e)"))

# plot phenotypic phase plane
plot(opt)

# plot reduced costs of the two control reactions
plot(opt, "EX_succ(e)")
plot(opt, "EX_o2(e)")
```

ppProc-class	Class "ppProc"
--------------	----------------

Description

Structure of the class "ppProc". Objects of that class are returned as part of class `optsol` when performing pre- or post-processing of an optimization, e.g. in `optimizeProb`.

Objects from the Class

Objects can be created by calls of the function `ppProc`:

```
test <- ppProc(cmd).
```

cmd: Object of class "list".

Slots

cmd: Object of class "list" a character vector or a list of character strings containing pre- or postprocessing commands.

pa: Object of class "list" return values of the pre- or postprocessing commands. They can be numeric, integer, character, list or of class `sybilError`.

ind: Object of class "integer" giving the indices of the optimizations when pre- or postprocessing was performed.

Methods

cmd: signature(object = "ppProc"): gets the cmd slot.

cmd<-: signature(object = "ppProc"): sets the cmd slot.

pa: signature(object = "ppProc"): gets the pa slot.

pa<-: signature(object = "ppProc"): sets the pa slot.

ind: signature(object = "ppProc"): gets the ind slot.

ind<-: signature(object = "ppProc"): sets the ind slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

`optimizeProb` and `optimizer`

Examples

```
showClass("ppProc")
```

`printMetabolite-methods`*Print Rows of the Stoichiometric Matrix*

Description

Print the rows of the stoichiometric matrix or an FBA model in CPLEX LP file format.

Usage

```
## S4 method for signature 'modelorg'  
printMetabolite(object, met, FBA1p = FALSE, printOut = TRUE, ...)
```

Arguments

<code>object</code>	An object of class <code>modelorg</code> .
<code>met</code>	A numeric or character vector containing the metabolite id's of metabolites to print out. If missing, all metabolites given in the model are used.
<code>FBA1p</code>	A single logical value. If set to TRUE, the output will be in CPLEX LP file format, including the objective function given in the model and reaction bounds. Additionally, if set to TRUE, argument <code>met</code> will be ignored; all metabolites present in the model are used. See also Details. Default: FALSE.
<code>printOut</code>	A single Boolean value. If set to TRUE, the desired reactions will be printed via the <code>cat</code> function. Default: TRUE.
<code>...</code>	Further arguments passed to <code>cat</code> , e.g. argument file.

Details

Metabolite id's beginning with a digit or period will be prefixed by the letter "r", reaction id's beginning with a digit or period will be prefixed by the letter "x" and square brackets in reaction or metabolite id's will be replaced by round brackets.

Value

The `modelorg` method returns a character vector of length equal to the number of metabolites given in argument `met`, invisibly. Each string represents the reaction participation of one particular metabolite.

Methods

`signature(object = "modelorg")` method to use with objects of class `modelorg`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Class [modelorg](#)

printReaction-methods *Print Columns of the Stoichiometric Matrix*

Description

Print the columns of the stoichiometric matrix.

Usage

```
## S4 method for signature 'modelorg,ANY'  
printReaction(object, react, printOut = TRUE, ...)  
## S4 method for signature 'summaryOptsol,modelorg'  
printReaction(object, mod, j, ...)
```

Arguments

object	An object of class modelorg or of class summaryOptsol .
mod	An object of class modelorg .
react	A numeric of character vector or an object of class reactId containing the reaction id's of reactions to print out.
j	A numeric of character vector indicating the simulations to consider, see Details .
printOut	A single Boolean value. If set to TRUE, the desired reactions will be printed via the cat function. Default: TRUE.
...	Further arguments passed to cat , e.g. argument file.

Details

The output of the [modelorg](#) method is compatible to the file format produced by [modelorg2tsv](#). Two columns are used: "abbreviation" containing the reaction id's and "equation" containing the reaction equation.

The [summaryOptsol](#) method prints the limiting reactions generated in simulations and stored in objects of class [summaryOptsol](#). Slot `react_id` of class [summaryOptsol](#) contains a list of reaction id's: list element `j` gives the reaction id's limiting simulation number `j`.

Value

The `modelorg` method returns invisibly a character vector of length equal to the number of reactions given in argument `react`. Each string consists of two tab-delimited values: first, the reaction id, second, the reaction equation.

The `summaryOptsol` returns invisibly a list of length equal to the number of elements in argument `j`. Each list element is of the same type as the return value of the `modelorg` method.

Methods

`signature(object = "modelorg")` method to use with objects of class `modelorg`.

`signature(object = "summaryOptsol", mod = "modelorg")` method to use with objects of class `summaryOptsol`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritze-meier <clausjonathan.fritze-meier@uni-duesseldorf.de>

See Also

Class `modelorg` and class `summaryOptsol`.

promptSysBiolAlg *Generate A Skeletal Structure of Subclasses of sysBiolAlg*

Description

Generates a skeletal structure of new subclasses of class `sysBiolAlg`, in particular for the constructor method `initialize`.

Usage

```
promptSysBiolAlg(algorithm, prefix = "sysBiolAlg", sep = "_",
                 suffix = "R", fpath = ".", ...)
```

Arguments

<code>algorithm</code>	A single character string containing the name of the new algorithm.
<code>prefix</code>	A single character string containing a prefix for the new algorithm, see Details below. Default: "sysBiolAlg".
<code>sep</code>	A single character string containing a separator for prefix and algorithm. Default: "_".
<code>suffix</code>	A single character string containing a file name suffix. Default: "R".

fpath A single character string containing a file path.
 Default:

. . . Further arguments passed to [file](#).

Details

The arguments `prefix` `algorithm` are stick together separated by `sep` (default: a single underscore `"_"`) to get the new class name: `prefix_algorithm`. The filename will be: `prefix_algorithmClass.R`.

The class definition in the new file will extend class `sysBiolAlg` directly and will not add any slots. Additionally a skeletal structure for method `initialize` will be generated. In this method, the user should create all arguments to the `initialize` method described in the base class `sysBiolAlg` and put them all to `callNextMethod`. Or, alternatively, generate an instance of class `optObj` "by hand".

Value

Returns NULL invisible.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

[sysBiolAlg](#)

reactId-class	<i>Structure of Class "reactId"</i>
---------------	-------------------------------------

Description

Structure of the class `"reactId"`. Objects of that class are returned by the function [checkReactId](#).

Objects from the Class

Objects can be created by calls of the form `new("reactId", mod_id, pnt, id = NULL, mod_key = "")`.

`mod_id`: Object of class `"character"` containing the model id.

`pnt`: Object of class `"numeric"` containing the column indices in a stoichiometric matrix of the reactions given in `react`.

`id`: Object of class `"character"` containing the reaction id's corresponding to argument `pos`. If set to `NULL` (default), no reaction id's are used.

`mod_key`: Object of class `"character"` containing the model key.

Slots

- `mod_id`: Object of class "character" containing the model id.
- `mod_key`: Object of class "character" containing the model key of the used model.
- `react_pos`: Object of class "integer" containing the column indices of reaction id's in the stoichiometric matrix of the metabolic model with id `mod_id`.
- `react_id`: Object of class "character" containing the reaction id's corresponding to the indices given in slot `react_pos`.
- `react_num`: Object of class "integer" containing the number of reaction id's.

Methods

- `mod_id<-`: signature(object = "reactId"): sets the `mod_id` slot.
- `mod_id`: signature(object = "reactId"): gets the `mod_id` slot.
- `mod_key<-`: signature(object = "reactId"): sets the `mod_key` slot.
- `mod_key`: signature(object = "reactId"): gets the `mod_key` slot.
- `react_pos<-`: signature(object = "reactId"): sets the `react_pos` slot.
- `react_pos`: signature(object = "reactId"): gets the `react_pos` slot.
- `react_id<-`: signature(object = "reactId"): sets the `react_id` slot.
- `react_id`: signature(object = "reactId"): gets the `react_id` slot.
- `length` signature(object = "reactId"): returns the number of reaction id's.
- `[`: signature(x = "reactId"): access like a vector. `x[i]` returns a new object of class `reactId` containing the `i`th reaction id.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

[checkReactId](#)

Examples

```
showClass("reactId")
```

reactId_Exch-class	Class "reactId_Exch"
--------------------	----------------------

Description

Structure of the class "reactId_Exch". Objects of that class are returned by the function [findExchReact](#).

Objects from the Class

Objects can be created by calls of the form `new("reactId_Exch", mod_id, mod_key, rpnt, rid, upt, mpnt, mid, lb,`

`mod_id`: Object of class "character" containing the model id.

`mod_key`: Object of class "character" containing the model key.

`rpnt`: Object of class "numeric" containing the column indices in a stoichiometric matrix of the reactions given in `rid`.

`rid`: Object of class "character" containing the reaction id's corresponding to argument `rpnt`.

`upt`: Object of class "logical": `upt[j]` equals TRUE if reaction j in `rid` is an uptake reaction (an exchange reaction with a lower bound less than zero).

`mpnt`: Object of class "numeric" containing the row indices in a stoichiometric matrix of the metabolites given in `mid`. The reaction given in `rid[j]` transports metabolite `mid[j]` across the system boundary of the model.

`mid`: Object of class "character" containing the metabolite id's corresponding to argument `mpnt`.

`lb`: Object of class "numeric" containing the lower bounds of the reactions given in `rpnt`.

`ub`: Object of class "numeric" containing the upper bounds of the reactions given in `rpnt`.

Slots

`uptake`: Object of class "logical" indicating if a certain reaction is an uptake reaction or not.

`met_pos`: Object of class "integer" containing the row indices of metabolite id's in the stoichiometric matrix of the metabolic model with id `mod_id`.

`met_id`: Object of class "character" containing the metabolite id's corresponding to the indices given in slot `met_pos`.

`lowbnd`: Object of class "numeric" containing the lower bounds of the reactions given in slot `react_pos`.

`uppbnd`: Object of class "numeric" containing the upper bounds of the reactions given in slot `react_pos`.

`mod_id`: Object of class "character" containing the model id.

`mod_key`: Object of class "character" containing the model key of the used model.

`react_pos`: Object of class "integer" containing the column indices of reaction id's in the stoichiometric matrix of the metabolic model with id `mod_id`.

`react_id`: Object of class "character" containing the reaction id's corresponding to the indices given in slot `react_pos`.

`react_num`: Object of class "integer" containing the number of reaction id's.

Extends

Class "[reactId](#)", directly.

Methods

met_pos signature(object = "reactId_Exch"): gets the met_pos slot.
met_pos<- signature(object = "reactId_Exch"): sets the met_pos slot.
met_id signature(object = "reactId_Exch"): gets the met_id slot.
met_id<- signature(object = "reactId_Exch"): sets the met_id slot.
react_pos signature(object = "reactId_Exch"): gets the react_pos slot.
react_pos<- signature(object = "reactId_Exch"): sets the react_pos slot.
react_id<- signature(object = "reactId"): sets the react_id slot.
react_id: signature(object = "reactId"): gets the react_id slot.
lowbnd signature(object = "reactId_Exch"): gets the lowbnd slot.
lowbnd<- signature(object = "reactId_Exch"): sets the lowbnd slot.
uppbnd signature(object = "reactId_Exch"): gets the uppbnd slot.
uppbnd<- signature(object = "reactId_Exch"): sets the uppbnd slot.
uptake signature(object = "reactId_Exch"): gets the uptake slot.
uptake<- signature(object = "reactId_Exch"): sets the uptake slot.
uptReact signature(object = "reactId_Exch"): gets the id's of uptake reactions.
uptMet signature(object = "reactId_Exch"): gets the metabolite id's of metabolites used by uptake reactions.
[: signature(x = "reactId_Exch"): access like a vector. x[i] returns a new object of class reactId_Exch containing the ith exchange reaction id.
show: signature(x = "reactId_Exch"): prints a table of all exchange reactions. If an upper or lower bound is equal or greater than abs(SYBIL_SETTINGS("MAXIMUM")), it will be shown as Inf or -Inf.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkReactId](#)

Examples

```
showClass("reactId")
```

readProb-methods *Read Problem Object From File*

Description

Read problem object from file.

Usage

```
## S4 method for signature 'optObj_clpAPI,character'
readProb(lp, fname, ff = "mps", ...)
```

```
## S4 method for signature 'optObj_cplexAPI,character'
readProb(lp, fname, ff = "lp")
```

```
## S4 method for signature 'optObj_glpkAPI,character'
readProb(lp, fname, ff = "lp", ...)
```

```
## S4 method for signature 'optObj_lpSolveAPI,character'
readProb(lp, fname, ff = "lp", ...)
```

Arguments

lp	An object extending class <code>optObj</code> .
fname	A single character string giving the file name to read from.
ff	A single character string giving the file format to use, see Details. Default: "lp".
...	Further arguments passed to the corresponding API routine.

Details

Argument "ff" in conjunction with **clpAPI** can be `mps` for MPS file format or `c1p` for COIN-OR Clp file format. Valid values for **cplexAPI** and `lpSolveAPI` are available in their documentations. For **glpkAPI**, argument "ff" can be `lp` for LP file format, `mps` for MPS file format or `glpk` for GLPK file format.

Methods

`signature(lp = "optObj_clpAPI", fname = "character")` method to use with package **optObj_clpAPI**. Argument `ff` is not used here.

`signature(lp = "optObj_cplexAPI", fname = "character")` method to use with package **optObj_cplexAPI**.

`signature(lp = "optObj_glpkAPI", fname = "character")` method to use with package **optObj_glpkAPI**.

`signature(lp = "optObj_lpSolveAPI", fname = "character")` method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#). Method to write problem objects: [writeProb](#)

Examples

```
## Not run:
# In very rare cases it is handy to save a sysBiolAlg-object:

library(sybil)
data(Ec_core)

# create a sysBiolAlg object (we use here GLPK (!))
prob <- sysBiolAlg(Ec_core, algorithm = "fba", solver="glpkAPI")

# write the R-object to disc
save(file="prob.RData",prob)

# now write the linear program part (managed by the solver) to disc
writeProb(prob@problem, fname="prob.lp", ff="lp")

# start new R session

library(sybil)
library(glpkAPI)
load("prob.RData") # restore the R-object
prob@problem@oobj <- initProbGLPK() # initialize a new linear program
readProb(problem(prob), fname="prob.lp") # load the previously saved linear program

## End(Not run)
```

readTSVmod

Read a Metabolic Network in a TSV (CSV) Format

Description

The function `readTSVmod` reads metabolic networks in text files, following a character-separated value format. Each line should contain one entry; the default value separator is a tab. Output files from the BiGG database are compatible.

Usage

```
readTSVmod(prefix, suffix,
           reactList, metList = NA, modDesc = NA,
           fielddelim = "\t", entrydelim = ", ", extMetFlag = "b",
           excludeComments = TRUE,
           oneSubSystem = TRUE,
           mergeMet = TRUE,
           balanceReact = TRUE,
           remUnusedMetReact = TRUE,
           singletonMet = FALSE,
           deadEndMet = FALSE,
           remMet = FALSE,
           constrMet = FALSE,
           tol = SYBIL_SETTINGS("TOLERANCE"),
           fpath = SYBIL_SETTINGS("PATH_TO_MODEL"),
           def_bnd = SYBIL_SETTINGS("MAXIMUM"),
           arrowlength = NULL,
           quoteChar = "",
           commentChar, ...)
```

Arguments

prefix	A single character string giving the prefix for three possible input files (see Details below).
suffix	A single character string giving the file name extension. If missing, the value of suffix depends on the argument fielddelim, see Details below. Default: "tsv".
reactList	A single character vector giving a file name containing a reaction list. Only necessary, if argument suffix is empty.
metList	A single character vector giving a file name containing a metabolite list. Default: NA.
modDesc	A single character vector giving a file name containing a model description. Default: NA.
fielddelim	A single character string giving the value separator. Default: "\t".
entrydelim	A single character string giving the a separator for values containing more than one entry. Default: ", ".
extMetFlag	A single character string giving the identifier for metabolites which are outside the system boundary. Only necessary, if the model is a closed one. Default: "b".
excludeComments	A Boolean value. Sometimes, the reaction abbreviations and/or the metabolite abbreviations contain comments in square brackets. If set to TRUE, these comments will be removed. If set to FALSE, whitespaces included in comments

in metabolite abbreviations will be removed. Comments in reaction abbreviations stay unchanged. A reaction id with comment is, for example, the string: pfk [comment], with [comment] being the comment. There must be at least one whitespace between id and comment, otherwise it will be considered as compartment flag.
Default: TRUE.

oneSubSystem	A Boolean value. Ignore parameter entrydelim for the field 'subsystem', if every reaction belongs to exactly one sub system. Default: TRUE.
mergeMet	Boolean: if set to TRUE, metabolites used more than once as reactand or product in a particular reaction are added up, see details below. If set to FALSE, the last value is used without warning. Default: TRUE.
balanceReact	Boolean: if set to TRUE, metabolites used as reactand and product in a particular reaction at the same time are balanced, see details below. If set to FALSE the last value is used without warning (reactands before products). Default: TRUE.
remUnusedMetReact	Boolean: if set to TRUE, metabolites and reactions which are not used in the stoichiometric matrix will be removed. A metabolite or a reaction is considered as unused, if the corresponding element of rowSums (metabolites) or colSums (reactions) of the binary version of the stoichiometric matrix is zero, see details below. If set to FALSE, only a warning is given. Default: FALSE.
singletonMet	Boolean: if set to TRUE, metabolites appearing only once in the stoichiometric matrix are identified. Metabolites appear only once, if rowSums of the binary stoichiometric matrix is one in the corresponding row, see details below. Default: FALSE.
deadEndMet	Boolean: if set to TRUE, metabolites which are produced but not consumed, or vice versa are identified, see details below. If both arguments singletonMet and deadEndMet are set to TRUE, the function will first look for singleton metabolites, and exclude them (and the corresponding reactions) from the search list. Afterwards, dead end metabolites are searched only in the smaller model. Default: FALSE.
remMet	Boolean: if set to TRUE, metabolites identified as singleton or dead end metabolites will be removed from the model. Additionally, reactions containing such metabolites will be removed also. Default: FALSE.
constrMet	Boolean: if set to TRUE, reactions containing metabolites identified as singleton or dead end metabolites will be constrained to zero. Default: FALSE.
tol	A single numeric value, giving the smallest positive floating point number unequal to zero, see details below. Default: SYBIL_SETTINGS("TOLERANCE").
fpath	A single character string giving the path to a certain directory containing the model files. Default: SYBIL_SETTINGS("PATH_TO_MODEL").

def_bnd	A single numeric value. Absolute value for upper and lower bounds for reaction bounds. Default: SYBIL_SETTINGS("MAXIMUM").
arrowlength	A single numeric or character value or NULL. This argument controls the number of "-" and "=" used in reaction arrows in the equation strings. If set to NULL, one or more symbols are used. The regular expression used is "<?[=-]+>". If numeric, all reaction arrows must consist of exactly arrowlength signs. The regular expression used is "<?[=-]{arrowlength}>". If character, arrowlength must be a regular expression and will be used as "<?[=-]arrowlength>". For example, if arrowlength is "{1,2}" the regular expression is "<?[=-]{1,2}>", meaning the reaction arrow can consist of one or two signs. In any case, the completed regular expression will always used with argument perl = TRUE. Default: NULL.
quoteChar	Set of quoting characters used for the argument quote in read.table , see there for details. Default: "" (disable quoting).
commentChar	A single character used for the argument comment.char in read.table , see there for details. If a comment char is needed, e.g. "@" (at) seems to be a good one. Default: "".
...	Further arguments passed to read.table , e.g. argument quote, comment.char or argument fill, if some lines do not have enough elements. If all fields are in double quotes, for example, set quote to "\"".

Details

A metabolic model consists of three input files:

1. <prefix>_react.<suffix> containing all reactions.
2. <prefix>_met.<suffix> containing all metabolites.
3. <prefix>_desc.<suffix> containing a model description.

All of these files must be character separated value files (for a detailed format description and examples, see package vignette). The argument prefix is the part of the filenames, all three have in common (e.g. if they were produced by [modelorg2tsv](#)). Alternatively, the arguments reactList, metList and modDesc can be used. A file containing all reactions must be there, everything else is optional.

If suffix is missing, it is set according to the value of fielddelim:

"\t"	"tsv"
","	"csv"
","	"csv"
" "	"dsv"
anything else	"dsv"

The argument ... is passed to [read.table](#).

In some cases, it could be necessary, to turn off quoting `quoteChar = ""` (default), if e.g. metabolite names contain quoting characters `"'` like in 3',5'-bisphosphate nucleotidase. If all fields are in quotes (e.g. files generated by `modelorg2tsv`), use `quoteChar = "\"` for example.

The input files are read using the function `read.table`. The argument `header` is set to `TRUE` and the argument `sep` is set to the value of `fielddelim`. Everything else can be passed via the `...` argument.

The header for the reactions list may have the following columns:

<code>"abbreviation"</code>	a unique reaction id
<code>"name"</code>	a reaction name
<code>"equation"</code>	the reaction equation
<code>"reversible"</code>	TRUE, if the reaction is reversible
<code>"compartment"</code>	reaction compartment(s) (currently unused)
<code>"lowbnd"</code>	lower bound
<code>"uppbnd"</code>	upper bound
<code>"obj_coef"</code>	objective coefficient
<code>"rule"</code>	gene to reaction association
<code>"subsystem"</code>	subsystem of the reaction

Every entry except for `"equation"` is optional. If there are missing values in field `"lowbnd"`, they will be set to `-1 * def_bnd`; if there are missing values in field `"uppbnd"`, they will be set to `def_bnd`; if there are missing values in field `"obj_coef"`, they will be set to `0`.

The header for the metabolites list may have the following columns:

<code>"abbreviation"</code>	a unique metabolite id
<code>"name"</code>	a metabolite name
<code>"compartment"</code>	metabolite compartment (currently unused)

If a metabolite list is provided, it is supposed to contain at least the entries `"abbreviation"` and `"name"`.

The header for the model description file may have the following columns:

<code>"name"</code>	a name for the model
<code>"id"</code>	a shorter model id
<code>"description"</code>	a model description
<code>"compartment"</code>	the compartments
<code>"abbreviation"</code>	unique compartment abbreviations
<code>"Nmetabolites"</code>	number of metabolites
<code>"Nreactions"</code>	number of reactions
<code>"Ngenes"</code>	number of independent genes
<code>"Nnnz"</code>	number of non-zero elements in the stoichiometric matrix

If a file contains a certain column name, there must be no empty entries.

If a model description file is provided, it is supposed to contain at least the entries "name" and "id". Otherwise, the filename of the reactions list will be used (the filename extension and the string _react at the end of the filename will be removed).

The compartments in which a reaction takes place is determined by the compartment flags of the participating metabolites.

All fields in the output files of `modelorg2tsv` are in double quotes. In order to read them, set argument `quoteChar` to "\"".

Please read the package vignette for detailed information about input formats and examples.

If a metabolite is used more than once as product or reactand of a particular reaction, it is merged: $a + (2) a$ is converted to $(3) a$ and a warning will be given.

If a metabolite is used first as reactand and then as product of a particular reaction, the reaction is balanced: $(2) b + a \rightarrow b + c$ is converted to $b + a \rightarrow c$

A binary version of the stoichiometric matrix S is constructed via $|S| > tol$.

A binary version of the stoichiometric matrix S is scanned for reactions and metabolites which are not used in S . If there are some, a warning will be given and the corresponding reactions and metabolites will be removed from the model if `remUnusedMetReact` is set to TRUE.

The binary version of the stoichiometric matrix S is scanned for metabolites, which are used only once in S . If there are some, at least a warning will be given. If either `constrMet` or `remMet` is set to TRUE, the binary version of S is scanned for paths of singleton metabolites. If `constrMet` is set to TRUE, reactions containing those metabolites will be constrained to zero; if `remMet` is set to TRUE, the metabolites and the reactions containing those metabolites will be removed from the network.

In order to find path of singleton metabolites a binary version of the stoichiometric matrix S is used. Sums of rows gives the vector of metabolite usage, each element is the number of reactions a metabolite participates. A single metabolite (singleton) is a metabolite with a row sum of one. All columns in S (reactions) containing singleton metabolites will be set to zero. And again, singleton metabolites will be searched until none are found.

The algorithm to find dead end metabolites works in a quite similar way, but not in the binary version of the stoichiometric matrix. Here, metabolite i is considered as dead end, if it is for example produced by reaction j but not used by any other reaction k .

Value

An instance of class `modelorg`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

References

The BiGG database <http://bigg.ucsd.edu/>.

Schellenberger, J., Park, J. O., Conrad, T. C., and Palsson, B. Ø., (2010) BiGG: a Biochemical Genetic and Genomic knowledgebase of large scale metabolic reconstructions. *BMC Bioinformatics* **11**, 213.

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

See Also

[read.table](#), [modelorg2tsv](#), [modelorg](#)

Examples

```
## read example dataset
mp <- system.file(package = "sybil", "extdata")
mod <- readTSVmod(prefix = "Ec_core", fpath = mp, quoteChar = "\\")

## redirect warnings to a log file
sink(file = "warn.log")
mod <- readTSVmod(prefix = "Ec_core", fpath = mp, quoteChar = "\\")
warnings()
sink()
unlink("warn.log")

## print no warnings
suppressWarnings(
  mod <- readTSVmod(prefix = "Ec_core", fpath = mp, quoteChar = "\\")
)

## print no messages
suppressMessages(
  mod <- readTSVmod(prefix = "Ec_core", fpath = mp, quoteChar = "\\")
)

## Not run:
## set number of warnings to keep
options(nwarnings = 1000)

## redirect every output to a file
zz <- file("log.Rout", open = "wt")
sink(zz)
sink(zz, type = "message")
mod <- readTSVmod(prefix = "Ec_core", fpath = mp, quoteChar = "\\")
warnings()
sink(type = "message")
sink()
close(zz)

## End(Not run)
```

resetChanges-methods *Generic Function to Reset Temporary Changes in Objects of Class sysBiolAlg*

Description

Use method `resetChanges` to undo changes in objects of class `sysBiolAlg` made by `applyChanges`.

Usage

```
## S4 method for signature 'sysBiolAlg'  
resetChanges(object, old_val)  
  
## S4 method for signature 'sysBiolAlg_room'  
resetChanges(object, old_val)
```

Arguments

`object` An object of class `sysBiolAlg`.
`old_val` A list containing the original values of the model. This list is returned by `applyChanges`.

Value

Invisibly TRUE will be returned.

Methods

`signature(object = "sysBiolAlg")` Method used with objects extending class `sysBiolAlg`
`signature(object = "sysBiolAlg_room")` Method used with objects of class `sysBiolAlg_room`

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Class `sysBiolAlg` and `applyChanges`

rmReact	<i>Remove Reactions From a Model</i>
---------	--------------------------------------

Description

The function `rmReact` removes reactions from a model.

Usage

```
rmReact(model, react, rm_met = TRUE)
```

Arguments

<code>model</code>	An object of class <code>modelorg</code>
<code>react</code>	An object of class <code>reactId</code> , a numeric vector, or a character vector containing reaction id's.
<code>rm_met</code>	Logical: also remove unused metabolites (default: TRUE).

Details

The argument `react` is evaluated by the function `checkReactId`.

Value

An object of class `modelorg`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

See Also

`modelorg`, `reactId` and `checkReactId`

Examples

```
data(Ec_core)
Ec_r <- rmReact(Ec_core, c("ATPM", "Biomass"))
```

robAna

*Robustness Analysis***Description**

Performs robustness analysis for a given metabolic model.

Usage

```
robAna(model, ctrlreact, rng = NULL,
        numP = 20, verboseMode = 1, ...)
```

Arguments

model	An object of class <code>modelorg</code> .
ctrlreact	An object of class <code>reactId</code> , character or integer. Specifies the control reaction – the parameter to vary.
rng	A numeric vector of length two, giving the lower and upper bound of the control reaction. If set to NULL (the default), the range will be computed by flux variability analysis for the reaction given in <code>ctrlreact</code> . Default: NULL
numP	The number of points to analyse. Default: 20
verboseMode	An integer value indicating the amount of output to stdout, see <code>optimizer</code> for details. Default: 1.
...	Further arguments passed to <code>optimizer</code> .

Details

The function `robAna` performs a robustness analysis with a given model. The flux of `ctrlreact` will be varied in `numP` steps between the maximum and minimum value the flux of `ctrlreact` can reach. For each of the `numP` datapoints the following lp problem is solved

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{v} \\ \text{s. t.} \quad & \mathbf{S} \mathbf{v} = 0 \\ & v_j = c_k \\ & \alpha_i \leq v_i \leq \beta_i \quad \forall i \in \{1, \dots, n\}, i \neq j \end{aligned}$$

with \mathbf{S} being the stoichiometric matrix, α_i and β_i being the lower and upper bounds for flux (variable) i . The total number of variables of the optimization problem is denoted by n . The parameter

c_k is varied `numP` times in the range of $v_{j,\min}$ to $v_{j,\max}$. The result of the optimization is returned as object of class `optsol_robAna` containing the objective value for each datapoint.

The extreme points of the range for `ctrlreact` are calculated via flux balance analysis (see also `sysBiolAlg_fba`) with the objective function being minimization and maximization of the flux through `ctrlreact`.

Value

An object of class `optsol_robAna`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

Bernhard Ø. Palsson (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press.

Examples

```
data(Ec_core)
rb <- robAna(Ec_core, ctrlreact = "EX_o2(e)")
plot(rb)
```

Description

Scaling of the constraint matrix of an optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI'  
scaleProb(lp, opt)  
  
## S4 method for signature 'optObj_cplexAPI'  
scaleProb(lp, opt)  
  
## S4 method for signature 'optObj_glpkAPI'  
scaleProb(lp, opt)  
  
## S4 method for signature 'optObj_lpSolveAPI'  
scaleProb(lp, opt)
```

Arguments

lp An object extending class `optObj`.
opt Scaling option depending on the used solver software.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

sensitivityAnalysis-methods

Sensitivity Analysis

Description

Perform sensitivity analysis.

Usage

```
## S4 method for signature 'optObj_cplexAPI'
sensitivityAnalysis(lp, ...)

## S4 method for signature 'optObj_glpkAPI'
sensitivityAnalysis(lp, ...)
```

Arguments

lp An object extending class `optObj`.
 ... Further arguments passed to the initialization function of the solver package.

Value

The `glpkAPI` method generates a file "sar.txt" and the `cplexAPI` method returns a list.

Methods

signature(lp = "optObj_cplexAPI") method to use with package `optObj_cplexAPI`.
 signature(lp = "optObj_glpkAPI") method to use with package `optObj_glpkAPI`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>
 Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

setColsNames-methods *Set/Change Variable Names*

Description

Set or change names of variables (columns) used in a optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric,character'
setColsNames(lp, j, names)

## S4 method for signature 'optObj_cplexAPI,numeric,character'
setColsNames(lp, j, names)

## S4 method for signature 'optObj_glpkAPI,numeric,character'
setColsNames(lp, j, names)
```

```
## S4 method for signature 'optObj_lpSolveAPI,numeric,character'  
setColsNames(lp, j, names)
```

Arguments

`lp` An object extending class `optObj`.
`j` A numeric vector of column indices.
`names` A character vector of the same length as `j` containing the column names.

Value

NULL is invisibly returned.

Methods

`signature(lp = "optObj_clpAPI", j = "numeric", names = "character")` method to use with package **optObj_clpAPI**.
`signature(lp = "optObj_cplexAPI", j = "numeric", names = "character")` method to use with package **optObj_cplexAPI**.
`signature(lp = "optObj_glpkAPI", j = "numeric", names = "character")` method to use with package **optObj_glpkAPI**.
`signature(lp = "optObj_lpSolveAPI", j = "numeric", names = "character")` method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

setObjDir-methods *Set Direction of Optimization*

Description

Set direction of optimization.

Usage

```
## S4 method for signature 'optObj_clpAPI,character'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_clpAPI,numeric'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_cplexAPI,character'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_cplexAPI,integer'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_cplexAPI,numeric'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_glpkAPI,character'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_glpkAPI,integer'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_glpkAPI,numeric'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_lpSolveAPI,character'
setObjDir(lp, lpdire)

## S4 method for signature 'optObj_lpSolveAPI,numeric'
setObjDir(lp, lpdire)
```

Arguments

`lp` An object extending class `optObj`.

`lpdire` A single character string, numeric or integer value. Can be set to "max" or -1 for maximization, or "min" or 1 for minimization. For packages `cplexAPI` and `glpkAPI` it is also possible to use the corresponding constant given by the package.

Methods

`signature(lp = "optObj_clpAPI", lpdire = "character")` method to use with package **optObj_clpAPI**. Set `lpdire` to "max" for maximization or "min" for minimization.

`signature(lp = "optObj_clpAPI", lpdire = "numeric")` method to use with package **optObj_clpAPI**. Set `lpdire` to -1 for maximization or 1 for minimization.

`signature(lp = "optObj_cplexAPI", lpdire = "character")` method to use with package **optObj_cplexAPI**. Set `lpdire` to "max" for maximization or "min" for minimization.

signature(lp = "optObj_cplexAPI", lpdire = "integer") method to use with package **optObj_cplexAPI**. Set lpdire to CPX_MAX for maximization or CPX_MIN for minimization.

signature(lp = "optObj_cplexAPI", lpdire = "numeric") method to use with package **optObj_cplexAPI**. Set lpdire to -1 for maximization or 1 for minimization.

signature(lp = "optObj_glpkAPI", lpdire = "character") method to use with package **optObj_glpkAPI**. Set lpdire to "max" for maximization or "min" for minimization.

signature(lp = "optObj_glpkAPI", lpdire = "integer") method to use with package **optObj_glpkAPI**. Set lpdire to GLP_MAX for maximization or GLP_MIN for minimization.

signature(lp = "optObj_glpkAPI", lpdire = "numeric") method to use with package **optObj_glpkAPI**. Set lpdire to -1 for maximization or 1 for minimization.

signature(lp = "optObj_lpSolveAPI", lpdire = "character") method to use with package **optObj_lpSolveAPI**. Set lpdire to "max" for maximization or "min" for minimization.

signature(lp = "optObj_lpSolveAPI", lpdire = "numeric") method to use with package **optObj_lpSolveAPI**. Set lpdire to -1 for maximization or 1 for minimization.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

setRhsZero-methods *Set Right Hand Side of the Optimization Problem To Zero*

Description

Set right hand side of the optimization problem to zero: $Sv = 0$.

Usage

```
## S4 method for signature 'optObj_clpAPI'
setRhsZero(lp)

## S4 method for signature 'optObj_cplexAPI'
setRhsZero(lp)

## S4 method for signature 'optObj_glpkAPI'
setRhsZero(lp)

## S4 method for signature 'optObj_lpSolveAPI'
setRhsZero(lp)
```


Arguments

lp An object extending class `optObj`.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

setRowsNames-methods *Set/Change Constraint Names*

Description

Set or change names of constraints (rows) used in a optimization problem.

Usage

```
## S4 method for signature 'optObj_clpAPI,numeric,character'
setRowsNames(lp, i, names)
```

```
## S4 method for signature 'optObj_cplexAPI,numeric,character'
setRowsNames(lp, i, names)
```

```
## S4 method for signature 'optObj_glpkAPI,numeric,character'
setRowsNames(lp, i, names)
```

```
## S4 method for signature 'optObj_lpSolveAPI,numeric,character'
setRowsNames(lp, i, names)
```

Arguments

lp An object extending class `optObj`.

i A numeric vector of row indices.

names A character vector of the same length as i containing the row names.

Value

NULL is invisibly returned.

Methods

signature(lp = "optObj_clpAPI", i = "numeric", names = "character") method to use with package **optObj_clpAPI**.

signature(lp = "optObj_cplexAPI", i = "numeric", names = "character") method to use with package **optObj_cplexAPI**.

signature(lp = "optObj_glpkAPI", i = "numeric", names = "character") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI", i = "numeric", names = "character") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

setSolverParm-methods *Set Parameters Used By The Optimization Software*

Description

Set parameters used by the optimization software. Parameters are set on a key-value basis. Sets of parameters can be set via a named list or a named data frame. The names of the parameters itself and possible values differ from solver to solver. Please consult the documentation of your solver software to get information about available parameters.

Usage

```
## S4 method for signature 'optObj_clpAPI'
setSolverParm(lp, solverParm)
```

```
## S4 method for signature 'optObj_cplexAPI'
setSolverParm(lp, solverParm)
```

```
## S4 method for signature 'optObj_glpkAPI'
setSolverParm(lp, solverParm)
```

```
## S4 method for signature 'optObj_lpSolveAPI'
setSolverParm(lp, solverParm)
```

Arguments

lp	An object extending class optObj .
solverParm	A named list or data frame containing sets of parameters. They must not contain NA values and every list or data frame element must have length one.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**. This method is currently unused. It is not possible to provide parameters for package **clpAPI**. Always FALSE will be returned.

signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**. In order to set integer parameters (parameters of type CPXINT), the value must be of type integer. For example, like `as.integer(42)` or `23L`.

signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.

signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass [optObj](#) and constructor function [optObj](#).

shrinkMatrix-methods *Get a Subset of Matrix Like Objects*

Description

Generate subsets of matrix-like objects.

Usage

```
## S4 method for signature 'modelorg'
shrinkMatrix(X, i = NULL, j = NULL,
             tol = SYBIL_SETTINGS("TOLERANCE"))
```

Arguments

X	An object treated to be matrix-like.
i	A numeric or character vector containing row indices of the matrix given in argument X. For the modelorg method, this can be an object of class reactId_Exch . Default: NULL.

j	A numeric or character vector containing column indices of the matrix given in argument X. For the <code>modelorg</code> method, this can be an object of class <code>reactId</code> . Default: NULL.
tol	A tolerance value. An element X_{ij} of the matrix given in argument X is considered to be zero, if $ X_{ij} > tol$ is true. Default: <code>SYBIL_SETTINGS("TOLERANCE")</code> .

Value

The `modelorg` method will return an object of class `Matrix`, with columns named by their reaction id's and rows named by their metabolite id's.

Methods

`signature(X = "modelorg")` method to use with objects of class `modelorg` for subsets of the stoichiometric matrix. Either argument `i` or argument `j` can be used, not both at the same time. If they are of type character, they must contain metabolite or reaction id's existing in the `modelorg` object. Use `i` to get the reactions in which the metabolites given in `i` participate (the metabolites given in `i` will be located in the first rows of the result). Use `j` to get all reactions given in `j`. The method will remove all non-zero rows and columns from the result.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Class `modelorg`.

Examples

```
# get the part of the stoichiometric containing
# the exchange reactions
data(Ec_core)
ex <- findExchReact(Ec_core)
shrinkMatrix(Ec_core, j = ex)
```

singletonMetabolites-methods

Identify Singleton Metabolites

Description

Search a metabolic network for metabolites, which appear only once in the stoichiometric matrix.

Usage

```
## S4 method for signature 'modelorg'  
singletonMetabolites(object, tol, retIds)
```

Arguments

object	An object of class <code>modelorg</code> .
tol	A numeric tolerance value: an entry of the stoichiometric matrix s_{ij} is considered to be non-zero if $abs(s_{ij}) > tol$ is TRUE. Default: SYBIL_SETTINGS("TOLERANCE").
retIds	Boolean. If set to TRUE, a list containing metabolite id's will be returned, otherwise a list of logical vectors. Default: TRUE.

Value

A list will be returned:

smet	singleton metabolites
sreact	reactions containing singleton metabolites

Methods

`signature(object = "modelorg")` method to use with class `modelorg`.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Class `modelorg` and `readTSVmod`.

Description

Optimize problem object.

Usage

```
## S4 method for signature 'optObj_clpAPI'
solveLp(lp)

## S4 method for signature 'optObj_cplexAPI'
solveLp(lp)

## S4 method for signature 'optObj_glpkAPI'
solveLp(lp)

## S4 method for signature 'optObj_lpSolveAPI'
solveLp(lp)
```

Arguments

lp An object extending class `optObj`.

Methods

signature(lp = "optObj_clpAPI") method to use with package **optObj_clpAPI**.
signature(lp = "optObj_cplexAPI") method to use with package **optObj_cplexAPI**.
signature(lp = "optObj_glpkAPI") method to use with package **optObj_glpkAPI**.
signature(lp = "optObj_lpSolveAPI") method to use with package **optObj_lpSolveAPI**.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`.

summaryOptsol

Summarize Objects of Class Optsol

Description

Generates a quick overview of results of simulations stored in objects of class `optsol`.

Usage

```
summaryOptsol(opt, mod, perc = 1, tol = SYBIL_SETTINGS("TOLERANCE"))
```

Arguments

opt	An object of class optsol .
mod	An object of class modelorg .
perc	A single numeric value in between zero and one indicating how close a flux value has to reach a flux boundary in order to be called “limiting”, see Details below. Default: 1.
tol	A tolerance value, see Details below. Default: SYBIL_SETTINGS("TOLERANCE").

Details

The function `summaryOptsol` generates a summary of the simulations resulting in the object given in argument `opt`. Both model id's, of the [optsol](#) object and of the [modelorg](#) object must be identical. The resulting object of class [summaryOptsol](#) contains information about the number of zeros and non-zeros in the flux distribution, the substrates and products and about the limiting reactions.

A reaction i is called “limiting”, if its flux value v_i is non-zero: $|v_i| > tol$ and if its flux value hits the flux boundaries: $v_i \leq v_{i,\min} \cdot perc \vee v_i \geq v_{i,\max} \cdot perc$.

Value

An object of class [summaryOptsol](#) if a flux distribution exists in argument `opt`, otherwise a [summary](#) of the objective values (`mod_obj`) is returned.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Class [optsol](#), class [modelorg](#) and class [summaryOptsol](#).

```
summaryOptsol-class  Class "summaryOptsol"
```

Description

Class `summaryOptsol` stores a summary of instances of class [optsol](#).

Objects from the Class

Objects can be created by calls of the form `summaryOptsol(opt, mod)`.

Slots

mod_id: Object of class "character" containing the model id of the analyzed model.

mod_key: Object of class "character" containing the model key of the used model.

nzeros: Object of class "integer" giving the number of zeros in the flux distribution.

nnonzero: Object of class "integer" giving the number of non-zeros in the flux distribution.

mod_obj: Object of class "numeric" containing the objective coefficients of the model.

ex_met: Object of class "character" containing the id's of exchange metabolites. These are metabolites which are transported across the system boundary.

ex_val: Object of class "Matrix" with each column being the flux distribution of the exchange metabolites of one optimization.

react_id: Object of class "list" with each list element containing a set of reaction id's limiting one optimization. A reactions is considered as "limiting", if it has a non-zero flux value and if its flux value hits an upper or lower bound.

chkcsol: Object of class "checksol" describing return values of the mathematical programming software and solution status.

Methods

ex_met signature(object = "summaryOptsol"): gets the ex_met slot.

ex_val signature(object = "summaryOptsol"): gets the ex_val slot.

plot: signature(x = "summaryOptsol"): plots a [histogram](#) of the values of the objective function in optimal state. Additional arguments can be passed to [histogram](#) via the ... argument.

image signature(x = "summaryOptsol"): plots a grey-scale representation of the exchange fluxes of the flux distribution. Black: metabolite is produced, grey: metabolite is imported. Further arguments are:

printOut A single logical value. If set to FALSE, a [trellis.object](#) is returned invisibly. Otherwise, a plot is drawn additionally.
Default: TRUE.

... Further arguments to [image-methods](#).

mod_id signature(object = "summaryOptsol"): gets the mod_id slot.

mod_id<- signature(object = "summaryOptsol"): sets the mod_id slot.

mod_key signature(object = "summaryOptsol"): gets the mod_key slot.

mod_key<- signature(object = "summaryOptsol"): sets the mod_key slot.

mod_obj signature(object = "summaryOptsol"): gets the mod_obj slot.

mod_obj<- signature(object = "summaryOptsol"): sets the mod_obj slot.

nnzero signature(object = "summaryOptsol"): gets the nnonzero slot.

nzeros signature(object = "summaryOptsol"): gets the nzeros slot.

printExchange signature(object = "summaryOptsol"): prints a matrix indicating wether a particular metabolite is taken up or produced by the metabolic network given certain conditions. Each line corresponds to one metabolite and each column to one optimization. A "-" indicates uptake and "+" indicates excretion. A whitespace character " " is used, if the metabolite is unused. Further arguments are:

- i A numeric vector indicating the metabolites (rows) to print: `i[x]` points to metabolite `ec_met(object)[x]`.
- j A numeric vector indicating the optimizations (columns) to print.
- dense A single Boolean value. If set to TRUE, each column has a column with of one letter.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

Constructor function [summaryOptsol](#), class [optsol](#) and class [modelorg](#).

Examples

```
showClass("summaryOptsol")
```

sybil-deprecated

Deprecated Functions and Methods in Package sybil

Description

These functions and methods will be defunct in the next release.

Details

- Function [blockedReact](#)

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[Deprecated](#)

sybilError-class *Class "sybilError"*

Description

Structure of the class "sybilError".

Objects from the Class

Objects can be created by calls of the function `sybilError`:

```
test <- sybilError(errmsg = "", number = NA).
```

`errmsg`: Object of class "character" containing an error message.

`number`: Object of class "integer" containing an error number.

Slots

`errmsg`: Object of class "character" error message.

`enum`: Object of class "integer" error number.

Methods

`errmsg`: `signature(object = "sybilError")`: gets the `errmsg` slot.

`errmsg<-`: `signature(object = "sybilError")`: sets the `errmsg` slot.

`enum`: `signature(object = "sybilError")`: gets the `enum` slot.

`enum<-`: `signature(object = "sybilError")`: sets the `enum` slot.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[optimizeProb](#)

Examples

```
showClass("sybilError")
```

sybilLog-class	Class "sybilLog"
----------------	------------------

Description

Handles log files, messages warnings and errors.

Objects from the Class

Objects can be created by calls of the function `sybilLog`:

```
logObj <- sybilLog(filename).
```

Slots

fh: Object of class `file` which is a connection to a file to print to.

fname: Object of class "character" being the name of the file to print to. If set to NA, no logfile is used. Default: NA.

fpath: Object of class "character" giving the path to the file mentioned in `fname`. Default: ".".

fenc: Object of class "character" encoding of the log file. Default: "".

loglevel: Object of class "integer" controlling the amount of details to log: If set to 0, nothing will be written to the logfile. If set to > 0, all warnings are logged; if set do > 1, also messages are logged. If `loglevel` is > 2, the used function call will be printed. Default: 0.

verblevel: Object of class "integer" controlling the amount of details to log: If set to 0, nothing will be written to the standard output connection. If set to > 0, all warnings are logged; if set do > 1, also messages are logged. Default: 0.

lastStep: Object of class "list" which is a stack, containing character strings describing performed steps. See also `sybilStack`.

lstname: Object of class "list" giving the name of the stack in `lastStep`.

didFoot: Object of class "logical" which is FALSE, if the footer of the log file is not yet printed, otherwise TRUE. This is useful if the function which is logged, stops unexpected.

Methods

`didFoot` signature(object = "sybilLog"): gets the `didFoot` slot.

`didFoot<-` signature(object = "sybilLog"): sets the `didFoot` slot.

`fenc` signature(object = "sybilLog"): gets the `fenc` slot.

`fenc<-` signature(object = "sybilLog"): sets the `fenc` slot.

`fh` signature(object = "sybilLog"): gets the `fh` slot.

`fh<-` signature(object = "sybilLog"): sets the `fh` slot.

`fname` signature(object = "sybilLog"): gets the `fname` slot.

`fname<-` signature(object = "sybilLog"): sets the `fname` slot.

`fpath` signature(object = "sybilLog"): gets the `fpath` slot.

`fpath<- signature(object = "sybilLog")`: sets the `fpath` slot.
`loglevel signature(object = "sybilLog")`: gets the `loglevel` slot.
`loglevel<- signature(object = "sybilLog")`: sets the `loglevel` slot.
`lstname signature(object = "sybilLog")`: gets the `lstname` slot.
`verblevel signature(object = "sybilLog")`: gets the `verblevel` slot.
`verblevel<- signature(object = "sybilLog")`: sets the `verblevel` slot.
`logCall signature(object = "sybilLog") (nog)`: writes all arguments and values of the function call to be logged to the log file. Nothing is printed to the standard output; `verblevel` has no meaning here; `verblevel` must be > 2 .

`nog` number of generations to go back

`logClose<- signature(object = "sybilLog")`: close the connection in slot `fh` and set it to NA. If slot `didFoot` is not TRUE, it prints a log comment to the connection in `fh` mentioning, that the logging ended unexpectedly.
`logComment signature(object = "sybilLog") (cmt, commentChar)`: add a comment to the log file if `loglevel` is > 2 and to `stdout` if `verblevel` is > 2 .

`cmt` the comment text
`cmtChar` a string to prefix `cmt`, default: #

`logError signature(object = "sybilLog") (msg, num)`: add an error message to the log file. Returns an object of class `sybilError`.

`msg` the error message
`num` an error number

`logFH signature(object = "sybilLog")`: Returns TRUE, if slot `fh` is of class `file`, otherwise FALSE.

`logFoot<- signature(object = "sybilLog")`: Print a head for your log file.

`logHead signature(object = "sybilLog")`: Print a foot for your log file.

`logMessage signature(object = "sybilLog")`: add a message to the log file if `loglevel` is > 1 .

... strings pasted to the log file

`logOptimization signature(object = "sybilLog") (ok, stat, obj, del, i)`: add a row containing results of an optimization to the log file if `loglevel` is > 2 and to `stdout` if `verblevel` is > 2 .

`opt no.`

```

ret
stat
obj value (numeric) val
dir if not given, it is a global value of the algorithm (here empty), otherwise the
obj c if not given, it is a global value of the model (here empty), otherwise the current setting of the objective coefficient
flux no. fluxes (variables) wh

```

`logOptimizationTH` signature(object = "sybilLog"): add a row containing a table header for results of an optimization to the log file if `loglevel` is > 2 and to stdout if `verblevel` is > 2. This should be used prior `logOptimization`.

`logStep<-` signature(object = "sybilLog"): (value): add a status message to the log file if `loglevel` is > 1, like "performing step x".

value strings giving the status

If `is.na(value)` evaluates to TRUE, the current process is assumed to have finished as expected. If `verblevel` is > 1, "OK" will be printed on the command line end if `loglevel` is > 1, "# done step x" will be printed to the log file.

`logWarning` signature(object = "sybilLog"): (...): add a warning to the log file if `loglevel` is > 0.

... strings pastes to the log file

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

Examples

```
showClass("sybilLog")
```

sybilStack

A Data Type Providing Stack (LIFO) And Queue (FIFO) Functionality

Description

These functions implement simple stack or queue functionality.

Usage

```
stinit(stname)
stclear(stname)
stpush(stname, value)
stpop(stname)
stunshift(stname, value)
stshift(stname)
stseek(stname)
stfirst(stname)
stlist(stname)
stlength(stname)
stexists(stname)
```

Arguments

stname	A single character string, giving the name of the stack or queue.
value	Value to add to the stack or queue.

Details

The funtion `stinit` creates an empty stack named `stname`.

The funtion `stclear` removes the stack named `stname`.

The funtion `stpush` appends element `value` at the end of the stack named `stname`.

The funtion `stpop` removes the last element of the stack named `stname` and returns it invisibly.

The funtion `stunshift` appends element `value` at the beginning of the stack `stname`.

The funtion `stshift` removes the first element of the stack named `stname` and returns it invisibly.

The funtion `stseek` returns the last element of the stack named `stname` but does not remove it.

The funtion `stfirst` returns the first element of the stack named `stname` but does not remove it.

The funtion `stlist` returns the stack named `stname` as list.

The funtion `stlength` returns the number of elements stored in the stack named `stname`.

The funtion `stexists` returns `TRUE` if a stack named `stname` exists, otherwise `FALSE`.

Value

The functions `stpop` and `stshift` return the last/first element of the stack invisibly. The functions `stseek` and `stfirst` just return the last/first element.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

Examples

```

## initialize empty stack named test
stinit("test")

## add a few elemets
stpush("test", 9)
stpush("test", 3)
stpush("test", 7)

## get last element
stpop("test")

## remove stack
stclear("test")

```

SYBIL_SETTINGS

Set and Get sybil Parameters

Description

Manage a set of default parameter settings for sybil.

Usage

```
SYBIL_SETTINGS(parm, value, ...)
```

Arguments

parm	A character string giving the name of the parameter to set.
value	The corresponding value.
...	Further arguments passed to checkDefaultMethod . Only used if parameters "SOLVER" or "METHOD" are set.

Details

Typical usages are

```

SYBIL_SETTINGS(parm, value)
SYBIL_SETTINGS(parm)
SYBIL_SETTINGS()

```

Possible parameters are:

"SOLVER" The default solver for lp problems. Possible values are depend on your installed API package.

glpkAPI: "glpkAPI",
cplexAPI: "cplexAPI",
clpAPI: "clpAPI",
lpSolveAPI: "lpSolveAPI".

Default: "glpkAPI".

"METHOD" The default method to solve lp problems. Possible values are

glpkAPI: "simplex", "interior", "exact" or mip.
cplexAPI: "lpopt", "primopt" "dualopt", "baropt", "hybbaropt", "hybnetopt", "siftopt",
 mipopt or qpopt.
clpAPI: "general_solve", "inidual" "iniprimal", "inibarrier", "inibarriernoc",
 "idiot", "dual" or "primal".
lpSolveAPI: "lp_solve".

Default: "simplex".

If the parameter "SOLVER" is changed, the corresponding default "METHOD" is the first one mentioned, e.g. for "cplexAPI", it will be "lpopt". This change is done automatically when changing the solver. It is not possible, to set a not existing "METHOD" for a particular "SOLVER", the corresponding default value will be used in such a case.

"MAXIMUM" Absolute maximum value.

Default: 1000.

"MODELORG_VERSION" Current version of modelorg-Class.

Value: "2.0".

This value must not be changed.

"ALGORITHM" Algorithm to use in order to analyze metabolic networks. Possible values are:

"fba" flux-balance analysis,
 "fv" flux-variance analysis,
 "mtf" minimize total flux,
 "moma" minimization of metabolic adjustment (MOMA),
 "lmoma" linear version of MOMA,
 "room" regulatory on/off minimization (ROOM).

Default: "fba".

"OPT_DIRECTION" Direction of optimization. Can be "max" or "min".

Default: "max".

"USE_NAMES" A logical value indicating if reaction id's and metabolite id's (or other names) should be used as names for variables and constraints in objects of class [sysBiolAlg](#).

Default: FALSE.

"PATH_TO_MODEL" Path to a directory to read or write files.

Default: ".".

"SOLVER_CTRL_PARM" A data.frame giving parameters to the optimizer software (e.g. GLPK).

Default: as.data.frame(NA).

"TOLERANCE" Tolerance value.

Default: 1E-6.

Value

If successful, a set of parameters to sybil will be returned.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzemeier <clausjonathan.fritzemeier@uni-duesseldorf.de>

See Also

[checkDefaultMethod](#)

Examples

```
## show all current parameters
SYBIL_SETTINGS()

## show current setting for "SOLVER"
SYBIL_SETTINGS("SOLVER")

## change current solver to glpkAPI
SYBIL_SETTINGS("SOLVER", "glpkAPI")
## Not run:
## this needs cplexAPI installed
## change current solver to cplexAPI
SYBIL_SETTINGS("SOLVER", "cplexAPI")

## End(Not run)
```

sysBiolAlg

General Constructor Function For Objects of Class sysBiolAlg

Description

This function serves as a user constructor function for objects of class [sysBiolAlg](#).

Usage

```
sysBiolAlg(model,
            algorithm = SYBIL_SETTINGS("ALGORITHM"),
            prefix = "sysBiolAlg", sep = "_",
            ...)
```

Arguments

model	An object of class modelorg .
algorithm	A single character string giving the name of the algorithm to use. See parameter "ALGORITHM" in SYBIL_SETTINGS for possible values. Default: <code>SYBIL_SETTINGS("ALGORITHM")</code> .
prefix	A single character string containing a prefix for the new class name. Default: <code>"sysBiolAlg"</code> .
sep	A single character string containing a separator for prefix and algorithm. Default: <code>"_"</code> .
...	Further arguments passed to the initialize method depending on the desired algorithm (see Details below).

Details

If argument `algorithm` is set to `"foo"` and `prefix` is set to `"sysBiolAlg"` (default), `sysBiolAlg` will try to build an instance of class `sysBiolAlg_foo`. If no such class definition exists, an error will be returned. For the name of the class, the values of arguments `prefix` and `algorithm` are stick together separated by the value of argument `sep`: `prefix_algorithm`.

Additional arguments required by the initialize method are for example `solver`, `method` and `solverParm`.

Value

An instance of a subclass of class [sysBiolAlg](#).

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Class [sysBiolAlg](#)

Examples

```
## Not run:
## The examples here require the package glpkAPI to be
## installed. If that package is not available, you have to set
## the argument 'solver' (the default is: solver = SYBIL_SETTINGS("SOLVER")).

data(Ec_core)

## algorithm: fba (flux balance analysis)
fb <- sysBiolAlg(Ec_core, algorithm = "fba")

## algorithm: lmoma (linearized version of MOMA)
fb <- sysBiolAlg(Ec_core, algorithm = "lmoma")

## End(Not run)
```

sysBioAlg-class	Class "sysBioAlg"
-----------------	-------------------

Description

The class sysBioAlg holds an object of class `optObj` which is generated concerning a particular algorithm, e.g. FBA or ROOM. This class is extended by other classes and will not be used as is. The representation of class sysBioAlg is used as superclass.

Details

The initialize method has the following arguments:

solver Single character string giving the solver package to use. See `SYBIL_SETTINGS` for possible values.

Default: `SYBIL_SETTINGS("SOLVER")`.

method Single character string giving the method the desired solver has to use. `SYBIL_SETTINGS` for possible values.

Default: `SYBIL_SETTINGS("METHOD")`.

solverParm A named data frame or list containing parameters for the specified solver. Parameters can be set as data frame or list: `solverParm = list(parm1 = val1, parm2 = val2)` with `parm1` and `parm2` being the names of two different parameters and `val1` and `val2` the corresponding values. For possible parameters and values see the documentation of the used solver package (e.g. `glpkAPI`).

Default: `SYBIL_SETTINGS("SOLVER_CTRL_PARM")`.

termOut A single boolean, numeric or character value, controlling the amount of terminal output of the solver software. See also `initProb` (argument `to`) for more details.

Default: `NULL`.

sbalg Single character string containing the name of the algorithm to use.

pType Single character string containing the type of the problem object. Can be "lp": linear program, mip: mixed integer program or "qp": quadratic program.

Default: "lp".

scaling Scaling options used to scale the constraint matrix. If set to `NULL`, no scaling will be performed (see `scaleProb`).

Default: `NULL`.

fi Pointers to columns (variables) representing a flux (reaction) in the original network. The variable `f1dind[i]` in the problem object represents reaction `i` in the original network.

nCols Number of columns (variables) of the problem object.

nRows Number of rows (constraints) of the problem object.

mat An object of class `Matrix`. The constraint matrix of the problem object. The number of columns in `mat` must be `nCols` and the number of rows in `mat` must be `nRows`.

ub A numeric vector of length `nCols` giving the upper bounds of the variables of the problem object.

- lb** A numeric vector of length nCols giving the lower bounds of the variables of the problem object.
- obj** A numeric vector of length nCols giving the objective coefficients of the variables of the problem object.
- rlb** A numeric vector of length nRows giving the right hand side of the problem object. If argument rub is not NULL, rlb contains the lower bounds of the constraints of the problem object.
- rtype** A character vector of length nRows giving the constraint type. See [loadLPprob](#) for details.
- lpdir** Single character string containing the direction of optimization. Can be set to "min" or "max".
Default: "max".
- rub** A numeric vector of length nRows giving the right hand side of the problem object. If not NULL, it contains the upper bounds of the constraints of the problem object.
Default: NULL.
- ctype** A character vector of length nCols giving the variable type. If set to NULL, no specific variable type is set, which usually means, all variables are treated as continuous variables. See [loadLPprob](#) for details.
Default: NULL.
- cnames** A character vector of length nCols giving the variable names. If set to NULL, no specific variable names are set.
Default: NULL.
- rnames** A character vector of length nRows giving the constraint names. If set to NULL, no specific constraint names are set.
Default: NULL.
- pname** A single character string containing a name for the problem object.
Default: NULL.
- retAlgPar** A single boolean flag, if algorithm specific parameters should be saved in the object extending class sysBiolAlg.
Default: TRUE.
- algPar** A named list containing algorithm specific parameters.
Default: NULL.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

- problem:** Object of class "optObj" containing the problem object.
- algorithm:** Object of class "character" containing the name of the algorithm.
- nr:** Object of class "integer" containing the number of rows of the problem object.
- nc:** Object of class "integer" containing the number of columns of the problem object
- fldind:** Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the problem object represents reaction i in the original network.
- alg_par:** Object of class "list" containing a named list of algorithm specific parameters.

Methods

algorithm signature(object = "sysBiolAlg"): gets the algorithm slot.

algorithm<- signature(object = "sysBiolAlg"): sets the algorithm slot.

alg_par signature(object = "sysBiolAlg"): gets the alg_par slot.

alg_par<- signature(object = "sysBiolAlg"): sets the alg_par slot.

fldind signature(object = "sysBiolAlg"): gets the fldind slot.

fldind<- signature(object = "sysBiolAlg"): sets the fldind slot.

nc signature(object = "sysBiolAlg"): gets the nc slot.

nc<- signature(object = "sysBiolAlg"): sets the nc slot.

nr signature(object = "sysBiolAlg"): gets the nr slot.

nr<- signature(object = "sysBiolAlg"): sets the nr slot.

optimizeProb signature(object = "sysBiolAlg"): runs optimization on the given problem object (see [optimizeProb](#) for details).

problem signature(object = "sysBiolAlg"): gets the problem slot.

initialize signature(object = "sysBiolAlg"): default constructor method for objects inheriting from class sysBiolAlg. It gets all data structures necessary to build a problem object (object of class [optObj](#)) representing a particular algorithm. This method can be used in constructor methods for subclasses of sysBiolAlg via [callNextMethod](#). In this case, the constructor has to generate all the data structures, pass them to [callNextMethod](#) and let the constructor of the superclass do all the work in generating the problem object and interacting with the solver software. See also the Details section.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

See Also

The general constructor function [sysBiolAlg](#), and classes [sysBiolAlg_fba](#), [sysBiolAlg_fv](#), [sysBiolAlg_mtf](#), [sysBiolAlg_lmoma](#), [sysBiolAlg_moma](#) and [sysBiolAlg_room](#).

Examples

```
showClass("sysBiolAlg")
```

sysBiolAlg_fba-class *Class "sysBiolAlg_fba"*

Description

The class `sysBiolAlg_fba` holds an object of class `optObj` which is generated to meet the requirements of the FBA algorithm.

Details

The `initialize` method has the following arguments:

model An object of class `modelorg`.

lpdir Single character string containing the direction of optimization. Can be set to "min" or "max".
Default: "max".

useNames A single boolean value. If set to TRUE, variables and constraints will be named according to `cnames` and `rnames`. If set to NULL, no specific variable or constraint names are set.
Default: `SYBIL_SETTINGS("USE_NAMES")`.

cnames A character vector giving the variable names. If set to NULL, the reaction id's of `model` are used.
Default: NULL.

rnames A character vector giving the constraint names. If set to NULL, the metabolite id's of `model` are used.
Default: NULL.

pname A single character string containing a name for the problem object.
Default: NULL.

scaling Scaling options used to scale the constraint matrix. If set to NULL, no scaling will be performed (see `scaleProb`).
Default: NULL.

writeProbToFileName A single character string containing a file name to which the problem object will be written in LP file format.
Default: NULL.

... Further arguments passed to the `initialize` method of `sysBiolAlg`. They are `solver`, `method` and `solverParm`.

The problem object is built to be capable to perform flux balance analysis (FBA) with a given model, which is basically the solution of a linear programming problem

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{v} \\ \text{s. t.} \quad & \mathbf{S} \mathbf{v} = 0 \\ & \alpha_i \leq v_i \leq \beta_i \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

with \mathbf{S} being the stoichiometric matrix, α_i and β_i being the lower and upper bounds for flux (variable) i respectively. The total number of variables of the optimization problem is denoted by n . The

solution of the optimization is a flux distribution maximizing the objective function $c^T v$ under the a given environment and the assumption of steady state. The optimization can be executed by using [optimizeProb](#).

Objects from the Class

Objects can be created by calls of the form

```
sysBiolAlg(model, algorithm = "fba", ...).
```

Arguments to ... which are passed to method `initialize` of class `sysBiolAlg_fba` are described in the Details section.

Slots

`problem`: Object of class "optObj" containing the problem object.

`algorithm`: Object of class "character" containing the name of the algorithm.

`nr`: Object of class "integer" containing the number of rows of the problem object.

`nc`: Object of class "integer" containing the number of columns of the problem object

`fldind`: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.

`alg_par`: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "[sysBiolAlg](#)", directly.

Methods

No methods defined with class "sysBiolAlg_fba" in the signature.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

References

Edwards, J. S., Covert, M and Palsson, B. Ø. (2002) Metabolic modelling of microbes: the flux-balance approach. *Environ Microbiol* **4**, 133–140.

Edwards, J. S., Ibarra, R. U. and Palsson, B. Ø. (2001) In silico predictions of *Escherichia coli* metabolic capabilities are consistent with experimental data. *Nat Biotechnol* **19**, 125–130.

See Also

Constructor function [sysBiolAlg](#) and superclass [sysBiolAlg](#).

Examples

```
showClass("sysBiolAlg_fba")
```

```
sysBiolAlg_fbaEasyConstraint-class
      Class "sysBiolAlg_fbaEasyConstraint" and
      Class "sysBiolAlg_mtfEasyConstraint"
```

Description

The classes `sysBiolAlg_fbaEasyConstraint` `sysBiolAlg_mtfEasyConstraint` hold an object of class `optObj` which is generated to meet the requirements of the FBA/MTF algorithm.

In Addition to this, it is very easy to add additional linear constraints to that linear problem. Each constraints is defined by the affected reaction, the coefficient, lower and upper bounds, and the constraint type.

Details

The problem object is built to be capable to perform flux balance analysis (FBA) with a given model, which is basically the solution of a linear programming problem

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{v} \\ \text{s. t.} \quad & \mathbf{S} \mathbf{v} = 0 \\ & \alpha_i \leq v_i \leq \beta_i \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

with \mathbf{S} being the stoichiometric matrix, α_i and β_i being the lower and upper bounds for flux (variable) i respectively. The total number of variables of the optimization problem is denoted by n . The solution of the optimization is a flux distribution maximizing the objective function $\mathbf{c}^T \mathbf{v}$ under the a given environment and the assumption of steady state. The optimization can be executed by using `optimizeProb`.

The additional i -th EasyConstraint will be added as follows to the problem: *to be checked*.

$$\gamma_i \leq v_{r_i} * (x_i)^T \leq \delta_i$$

Here r_i (= `easyConstraint$react[[i]]`) is a set of reaction indices and x_i (= `easyConstraint$x[[i]]`) is the corresponding set of coefficients. γ and δ are the vectors of lower and upper bounds for the constraints, respectively. For the type of (in)equality (\leq, \dots) see the text above for parameter `rtype`.

Objects from the Class

Objects can be created by calls of the form

```
sysBiolAlg(model, algorithm = "fbaEasyConstraint", ...).
```

Arguments to `...` which are passed to method `initialize` of class `sysBiolAlg_fba` are described in the Details section.

Slots

Slots are the same as in the original MTF/FBA classes. In addition, this slot is implemented:

Named list holding the information for the constraints (see details):

`easyConstraint` react List of numeric vectors. Values indicate, to which reaction the constraint applies.

- `x` List of numeric vectors. Values indicate coefficients of the constraint. Lengths have to be equal to `react`-field.
- `lb` Numeric vector of lower bounds for constraints. If not given, a default bound of 0 will be used.
- `ub` Numeric vector of upper bounds for constraints. If not given, a default bound of 0 will be used. Only needed for constraints, that need two bounds.
- `rtype` Character vector defining the type of constraint.

"F":	free constraint (GLPK only)	$-\infty < x < \infty$
"L":	constraint with lower bound	$lb \leq x < \infty$
"U":	constraint with upper bound	$-\infty < x \leq ub$
"D":	double-bounded (ranged) constraint	$lb \leq x \leq ub$
"E":	fixed (equality) constraint	$lb = x = ub$

If `rtype[i]` is not one of "F", "L", "U", "D" or "E", the value of `rtype[i]` will be set to "E". See Details of [loadLPprob](#).

Extends

Class "[sysBiolAlg](#)", directly.

Methods

No methods defined with class "`sysBiolAlg_fbaEasyConstraint`" in the signature.

Author(s)

Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Edwards, J. S., Covert, M and Palsson, B. Ø. (2002) Metabolic modelling of microbes: the flux-balance approach. *Environ Microbiol* **4**, 133–140.

Edwards, J. S., Ibarra, R. U. and Palsson, B. Ø. (2001) In silico predictions of *Escherichia coli* metabolic capabilities are consistent with experimental data. *Nat Biotechnol* **19**, 125–130.

See Also

Constructor function [sysBiolAlg](#) and superclass [sysBiolAlg](#).

Examples

```

showClass("sysBiolAlg_fbaEasyConstraint")

# see package vignette for second example with more comments:
#vignette("sybil")

#load model
data(Ec_core)

# allow influx of Fumarate and restrict outflux of Fumarate and Glucose
lowbnd(Ec_core)[react_id(Ec_core) %in% c("EX_fum(e)")] <- -1000
uppbnd(Ec_core)[react_id(Ec_core) %in% c("EX_glc(e)", "EX_fum(e)")] <- 0

# see result
findExchReact(Ec_core)
optimizeProb(Ec_core)

# define easyConstraint to have the same influx for Glucose and Fumarate:
# EX_glc(e) = EX_fum(e)
# here we omit the upper and lower bound, hence they are set to zero.
ec <- list(
  react=list(which(react_id(Ec_core) %in% c("EX_glc(e)", "EX_fum(e)"))),
  x=list(c(1, -1)),
  rtype="E")

# optimize
opt <- optimizeProb(Ec_core, algorithm="fbaEasyConstraint", easyConstraint=ec)

# check if fluxes are really the same:
fluxes(opt)[react_id(Ec_core) %in% c("EX_glc(e)", "EX_fum(e)")]

```

```
sysBiolAlg_fv-class    Class "sysBiolAlg_fv"
```

Description

The class `sysBiolAlg_fv` holds an object of class `optObj` which is generated to meet the requirements of the flux variance algorithm.

Details

The `initialize` method has the following arguments:

- model** An object of class `modelorg`.
- percentage** Consider solutions with x percent of the optimal solution.
Default: 100.
- Zopt** A single numeric value giving the optimal value to be fixed during all other optimizations (see argument `fixObjVal`). If `Zopt` is set to NULL and `model` has an objective function, a default value is computed based on FBA. If given, arguments `solver`, `method` and `solverParm` are used during FBA.
Default: NULL.
- fixObjVal** A single Boolean value. If set to TRUE and if the model contains an objective function, an optimal value of this objective function will be fixed during all other optimizations. The optimal value can be controlled by argument `Zopt`.
Default: TRUE.
- tol** Single numeric value giving the tolerance value.
Default: SYBIL_SETTINGS("TOLERANCE").
- lpdir** Single character string containing the direction of optimization. Can be set to "min" or "max".
Default: SYBIL_SETTINGS("OPT_DIRECTION").
- useNames** A single boolean value. If set to TRUE, variables and constraints will be named according to `cnames` and `rnames`. If set to NULL, no specific variable or constraint names are set.
Default: SYBIL_SETTINGS("USE_NAMES").
- cnames** A character vector giving the variable names. If set to NULL, the reaction id's of `model` are used.
Default: NULL.
- rnames** A character vector giving the constraint names. If set to NULL, the metabolite id's of `model` are used. If an objective value has to be fixed (see argument `fixObjVal`), the corresponding constrained is named "Z".
Default: NULL.
- pname** A single character string containing a name for the problem object.
Default: NULL.
- scaling** Scaling options used to scale the constraint matrix. If set to NULL, no scaling will be performed (see `scaleProb`).
Default: NULL.
- writeProbToFileName** A single character string containing a file name to which the problem object will be written in LP file format.
Default: NULL.
- ... Further arguments passed to the initialize method of `sysBiolAlg`. They are `solver`, `method` and `solverParm`.

The problem object is built to be capable to perform the flux variance algorithm with a given model, which is basically the solution of a linear program

$$\begin{aligned}
 & \max \text{ or } \min && v_i \\
 & \text{s. t.} && Z = Z_{\text{opt}} \\
 & && \mathbf{S}\mathbf{v} = 0 \\
 & && \alpha_i \leq v_i \leq \beta_i \quad \forall i \in \{1, \dots, n\}
 \end{aligned}$$

with S being the stoichiometric matrix, α_i and β_i being the lower and upper bounds for flux (variable) i . The total number of variables of the optimization problem is denoted by n . The optimization can be executed by using `optimizeProb`.

Objects from the Class

Objects can be created by calls of the form

```
sysBiolAlg(model, algorithm = "fv", ...).
```

Arguments to `...` which are passed to method `initialize` of class `sysBiolAlg_fv` are described in the Details section.

Slots

`problem`: Object of class "optObj" containing the problem object.

`algorithm`: Object of class "character" containing the name of the algorithm.

`nr`: Object of class "integer" containing the number of rows of the problem object.

`nc`: Object of class "integer" containing the number of columns of the problem object

`fldind`: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction i in the original network.

`alg_par`: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "`sysBiolAlg`", directly.

Methods

No methods defined with class "`sysBiolAlg_fv`" in the signature.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

Bernhard Ø. Palsson (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press.

See Also

Constructor function [sysBiolAlg](#) and superclass [sysBiolAlg](#).

Examples

```
showClass("sysBiolAlg_fv")
```

```
sysBiolAlg_lmoma-class
```

```
Class "sysBiolAlg_lmoma"
```

Description

The class `sysBiolAlg_lmoma` holds an object of class `optObj` which is generated to meet the requirements of a linearized version of the MOMA algorithm.

Details

The `initialize` method has the following arguments:

model An object of class `modelorg`.

wflux A numeric vector holding an optimal wild type flux distribution for the given model. If missing, a default value is computed based on FBA. If given, arguments `solver` and `method` are used, but `solverParm` is not.

COBRAflag Boolean, prepare problem object in order to perform minimization of metabolic adjustment as in COBRA Toolbox.
Default: FALSE.

wtobj Only used if argument `COBRAflag` is set to TRUE: A single numeric value giving the optimized value of the objective function of the wild type problem. If missing, a default value is computed based on FBA. If given, arguments `solver` and `method` are used, but `solverParm` is not.

wtobjLB Only used if argument `COBRAflag` is set to TRUE: Boolean. If set to TRUE, the value of argument `wtobj` is treated as lower bound. If set to FALSE, `wtobj` serves as an upper bound.
Default: TRUE.

obj_coefD A numeric vector of length two times the number of reactions in the model containing the non-zero part of the objective function. If set to NULL, the vector is filled with ones.
Default: NULL.

absMAX A single numerical value used as a maximum value for upper variable and constraint bounds.
Default: `SYBIL_SETTINGS("MAXIMUM")`.

useNames A single boolean value. If set to TRUE, variables and constraints will be named according to `cnames` and `rnames`. If set to NULL, no specific variable or constraint names are set.
Default: `SYBIL_SETTINGS("USE_NAMES")`.

cnames A character vector giving the variable names. If set to NULL, the reaction id's of model are used.

Default: NULL.

rnames A character vector giving the constraint names. If set to NULL, the metabolite id's of model are used.

Default: NULL.

pname A single character string containing a name for the problem object.

Default: NULL.

scaling Scaling options used to scale the constraint matrix. If set to NULL, no scaling will be performed (see [scaleProb](#)).

Default: NULL.

writeProbToFileName A single character string containing a file name to which the problem object will be written in LP file format.

Default: NULL.

... Further arguments passed to the initialize method of [sysBiolAlg](#). They are solver, method and solverParm.

The problem object is built to be capable to perform a linearized version of the MOMA algorithm with a given model, which is basically the solution of a linear programming problem

$$\begin{aligned} \min \quad & \sum_{i,j=1}^n |v_{j,\text{del}} - v_{i,\text{wt}}| \\ \text{s. t.} \quad & \mathbf{S}\mathbf{v}_{\text{del}} = 0 \\ & v_i = v_{i,\text{wt}} \quad \forall i \in \{1, \dots, n\} \\ & \alpha_j \leq v_{j,\text{del}} \leq \beta_j \quad \forall j \in \{1, \dots, n\} \end{aligned}$$

Here, \mathbf{v}_{wt} is the optimal wild type flux distribution. This can be set via the argument `wtflux`. If `wtflux` is NULL (the default), the wild type flux distribution will be calculated by a standard FBA.

If argument `COBRAflag` is set to TRUE, the linear program is formulated differently. Wild type and knock-out strain will be computed simultaneously.

$$\begin{aligned} \min \quad & \sum_{i,j=1}^n |v_{j,\text{del}} - v_{i,\text{wt}}| \\ \text{s. t.} \quad & \mathbf{S}\mathbf{v}_{\text{wt}} = 0 \\ & \alpha_i \leq v_{i,\text{wt}} \leq \beta_i \quad \forall i \in \{1, \dots, n\} \\ & \mathbf{S}\mathbf{v}_{\text{del}} = 0 \\ & \alpha_j \leq v_{j,\text{del}} \leq \beta_j \quad \forall j \in \{1, \dots, n\} \\ & \mu_{\text{wt}} = \mathbf{c}^T \mathbf{v}_{\text{wt}} \end{aligned}$$

with \mathbf{S} being the stoichiometric matrix, α_i and β_i being the lower and upper bounds for flux (variable) i (j for the deletion strain). The total number of variables of the optimization problem is

denoted by n . Here, μ_{wt} is the optimal wild type growth rate. This can be set via the argument `wtbody`. If `wtbody` is NULL (the default), the wild type growth rate will be calculated by a standard FBA. The optimization can be executed by using `optimizeProb`.

Objects from the Class

Objects can be created by calls of the form

```
sysBiolAlg(model, algorithm = "lmoma", ...).
```

Arguments to `...` which are passed to method `initialize` of class `sysBiolAlg_lmoma` are described in the Details section.

Slots

`problem`: Object of class "optObj" containing the problem object.

`algorithm`: Object of class "character" containing the name of the algorithm.

`nr`: Object of class "integer" containing the number of rows of the problem object.

`nc`: Object of class "integer" containing the number of columns of the problem object

`fldind`: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.

`alg_par`: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "`sysBiolAlg`", directly.

Methods

No methods defined with class "`sysBiolAlg_lmoma`" in the signature.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø. and Herrgard, M. J. (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat Protoc* **2**, 727–738.

Edwards, J. S., Covert, M and Palsson, B. Ø. (2002) Metabolic modelling of microbes: the flux-balance approach. *Environ Microbiol* **4**, 133–140.

Edwards, J. S., Ibarra, R. U. and Palsson, B. Ø. (2001) In silico predictions of *Escherichia coli* metabolic capabilities are consistent with experimental data. *Nat Biotechnol* **19**, 125–130.

Schellenberger, J., Que, R., Fleming, R. M. T., Thiele, I., Orth, J. D., Feist, A. M., Zielinski, D. C., Bordbar, A., Lewis, N. E., Rahmanian, S., Kang, J., Hyduke, D. R. and Palsson, B. Ø. (2011)

Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat Protoc* **6**, 1290–1307.

Segrè, D., Vitkup, D. and Church, G. M. (2002) Analysis of optimality in natural and perturbed metabolic networks. *PNAS* **99**, 15112–15117.

See Also

Constructor function [sysBiolAlg](#) and superclass [sysBiolAlg](#).

Examples

```
showClass("sysBiolAlg_lmoma")
```

```
sysBiolAlg_moma-class  Class "sysBiolAlg_moma"
```

Description

The class `sysBiolAlg_moma` holds an object of class `optObj` which is generated to meet the requirements of the MOMA algorithm.

Details

The initialize method has the following arguments:

model An object of class `modelorg`.

wflux A numeric vector holding an optimal wild type flux distribution for the given model. If set to NULL, a default value is computed based on flux-balance analysis. If given, arguments `solver` and `method` are used, but `solverParm` is not. Default: NULL.

Qmat A numeric vector or matrix (of class `Matrix`) holding the quadratic part of the objective function. If set to NULL, a quadratic unity matrix with number of columns and rows equal to the number of reactions given in the model is used. Default: NULL.

scaleDist A numeric vector containing scaling factors for each reaction in the objective function. If `scaleDist[j]` is set to 0, reaction `j` will be ignored. The quadratic and the linear part of the objective function are multiplied by this factor. If set to NULL, the reactions are not scaled. Default: NULL.

useNames A single boolean value. If set to TRUE, variables and constraints will be named according to `cnames` and `rnames`. If set to NULL, no specific variable or constraint names are set. Default: `SYBIL_SETTINGS("USE_NAMES")`.

cnames A character vector giving the variable names. If set to NULL, the reaction id's of `model` are used. Default: NULL.

rnames A character vector giving the constraint names. If set to NULL, the metabolite id's of `model` are used. Default: NULL.

- pname** A single character string containing a name for the problem object.
Default: NULL.
- scaling** Scaling options used to scale the constraint matrix. If set to NULL, no scaling will be performed (see [scaleProb](#)).
Default: NULL.
- writeProbToFileName** A single character string containing a file name to which the problem object will be written in LP file format.
Default: NULL.
- ... Further arguments passed to the initialize method of [sysBiolAlg](#). They are solver, method and solverParm.

The problem object is built to be capable to perform the MOMA algorithm with a given model, which is basically the solution of a quadratic programming problem

$$\begin{aligned} \min \quad & \sum_{j=1}^n ((v_{j,\text{del}} - v_{j,\text{wt}}) \cdot sd_j)^2 \\ \text{s. t.} \quad & \mathbf{S}\mathbf{v} = 0 \\ & \alpha_j \leq v_j \leq \beta_j \quad \forall j \in \{1, \dots, n\} \end{aligned}$$

with \mathbf{S} being the stoichiometric matrix, α_j and β_j being the lower and upper bounds for flux (variable) j and sd_j being the scaling factor for reaction j (default: $sd_j = 1, \forall j$). The total number of variables of the optimization problem is denoted by n . Here, \mathbf{v}_{wt} is the optimal wild type flux distribution. This can be set via the argument `wtflux`. If `wtflux` is NULL (the default), the wild type flux distribution will be calculated by a standard FBA. The optimization can be executed by using [optimizeProb](#).

Objects from the Class

Objects can be created by calls of the form
`sysBiolAlg(model, algorithm = "moma", ...)`.

Arguments to `...` which are passed to method `initialize` of class `sysBiolAlg_moma` are described in the Details section.

Slots

- problem**: Object of class "optObj" containing the problem object.
- algorithm**: Object of class "character" containing the name of the algorithm.
- nr**: Object of class "integer" containing the number of rows of the problem object.
- nc**: Object of class "integer" containing the number of columns of the problem object
- fldind**: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable `fldind[i]` in the problem object represents reaction `i` in the original network.
- alg_par**: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "[sysBiolAlg](#)", directly.

Methods

No methods defined with class "sysBiolAlg_moma" in the signature.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Segrè, D., Vitkup, D. and Church, G. M. (2002) Analysis of optimality in natural and perturbed metabolic networks. *PNAS* **99**, 15112–15117.

See Also

Constructor function [sysBiolAlg](#) and superclass [sysBiolAlg](#).

Examples

```
showClass("sysBiolAlg_moma")
```

```
sysBiolAlg_mtf-class  Class "sysBiolAlg_mtf"
```

Description

The class `sysBiolAlg_mtf` holds an object of class `optObj` which is generated to meet the requirements of the minimize total flux algorithm: minimize the absolute sum of all fluxes given a previously calculated objective value.

Details

The `initialize` method has the following arguments:

model An object of class `modelorg`.

wtojb A single numeric value giving the optimal value. If missing, a default value is computed based on FBA. If given, arguments `solver` and `method` are used, but `solverParm` is not.
Default: NULL.

react Arguments `react`, `lb` and `ub` are used, if argument `wtojb` is NULL, meaning: no previous objective value is given. Objective values will be calculated via `fba` using the parameters given in `react`, `lb` and `ub`.
Default: NULL.

- lb** See argument react.
Default: NULL.
- ub** See argument react.
Default: NULL.
- costcoeffw** A numeric vector containing cost coefficients for all variables (forward direction). If set to NULL, all cost coefficients are set to 1, so that all variables have the same impact on the objective function.
Default: NULL.
- costcoeffb** A numeric vector containing cost coefficients for all variables (backward direction). If set to NULL, all cost coefficients are set to the values given in costcoeffw.
Default: NULL.
- absMAX** A single numerical value used as a maximum value for upper variable and constraint bounds.
Default: SYBIL_SETTINGS("MAXIMUM").
- useNames** A single boolean value. If set to TRUE, variables and constraints will be named according to cnames and rnames. If set to NULL, no specific variable or constraint names are set.
Default: SYBIL_SETTINGS("USE_NAMES").
- cnames** A character vector giving the variable names. If set to NULL, the reaction id's of model are used.
Default: NULL.
- rnames** A character vector giving the constraint names. If set to NULL, the metabolite id's of model are used.
Default: NULL.
- pname** A single character string containing a name for the problem object.
Default: NULL.
- scaling** Scaling options used to scale the constraint matrix. If set to NULL, no scaling will be performed (see [scaleProb](#)).
Default: NULL.
- writeProbToFileName** A single character string containing a file name to which the problem object will be written in LP file format.
Default: NULL.
- ... Further arguments passed to the initialize method of [sysBiolAlg](#). They are solver, method and solverParm.

The problem object is built to be capable to perform minimize total flux with a given model, which is basically the solution of a linear programming problem

$$\begin{aligned}
 \min \quad & \sum_{i=1}^n cost_i |v_i| \\
 \text{s. t.} \quad & \mathbf{S}\mathbf{v} = 0 \\
 & \alpha_i \leq v_i \leq \beta_i \quad \forall i \in \{1, \dots, n\} \\
 & \mathbf{c}_{wt} \geq \mathbf{c}^T \mathbf{v}_{wt}
 \end{aligned}$$

with $c^T v_{wt}$ being the previously computed optimized value of the objective function (argument wtobj). The variable S denotes the stoichiometric matrix, α_i and β_i being the lower and upper bounds for flux (variable) i . The total number of variables of the optimization problem is denoted by n . The optimization can be executed by using `optimizeProb`.

Objects from the Class

Objects can be created by calls of the form

```
sysBiolAlg(model, algorithm = "mtf", ...).
```

Arguments to ... which are passed to method initialize of class sysBiolAlg_mtf are described in the Details section.

Slots

maxobj: Object of class "numeric" containing optimized objective values.

problem: Object of class "optObj" containing the problem object.

algorithm: Object of class "character" containing the name of the algorithm.

nr: Object of class "integer" containing the number of rows of the problem object.

nc: Object of class "integer" containing the number of columns of the problem object

fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the problem object represents reaction i in the original network.

alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "`sysBiolAlg`", directly.

Methods

changeMaxObj signature(object = "sysBiolAlg_mtf"): change current objective value to the j th value given in slot maxobj. Argument j must be in $[1:\text{length}(\text{maxobj})]$.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Edwards, J. S., Covert, M and Palsson, B. Ø. (2002) Metabolic modelling of microbes: the flux-balance approach. *Environ Microbiol* **4**, 133–140.

Edwards, J. S., Ibarra, R. U. and Palsson, B. Ø. (2001) In silico predictions of *Escherichia coli* metabolic capabilities are consistent with experimental data. *Nat Biotechnol* **19**, 125–130.

See Also

Constructor function [sysBiolAlg](#) and superclass [sysBiolAlg](#).

Examples

```
showClass("sysBiolAlg_mtf")
```

```
sysBiolAlg_room-class  Class "sysBiolAlg_room"
```

Description

The class `sysBiolAlg_room` holds an object of class `optObj` which is generated to meet the requirements of the ROOM algorithm.

Details

The initialize method has the following arguments:

model An object of class `modelorg`.

wtflux A numeric vector holding an optimal wild type flux distribution for the given model. If missing, a default value is computed based on FBA. If given, arguments `solver` and `method` are used to calculate the default, but `solverParm` is not.

delta A single numeric value giving the relative range of tolerance, see Details below.
Default: 0.03.

epsilon A single numeric value giving the absolute range of tolerance, see Details below.
Default: 0.001.

LPvariant Boolean. If TRUE, the problem object is formulated as linear program. See Details below.
Default: FALSE.

LPvariant Boolean. If TRUE, the problem object is formulated as linear program. See Details below.
Default: FALSE.

absMAX A single numerical value used as a maximum value for upper variable and constraint bounds.
Default: `SYBIL_SETTINGS("MAXIMUM")`.

cnames A character vector giving the variable names. If set to NULL, the reaction id's of `model` are used.
Default: NULL.

rnames A character vector giving the constraint names. If set to NULL, the metabolite id's of `model` are used.
Default: NULL.

pname A single character string containing a name for the problem object.
Default: NULL.

scaling Scaling options used to scale the constraint matrix. If set to NULL, no scaling will be performed (see [scaleProb](#)).

Default: NULL.

writeProbToFileName A single character string containing a file name to which the problem object will be written in LP file format.

Default: NULL.

... Further arguments passed to the initialize method of [sysBiolAlg](#). They are solver, method and solverParm.

The problem object is built to be capable to perform the ROOM algorithm with a given model, which is basically the solution of a mixed integer programming problem

$$\begin{aligned} \min \quad & \sum_{i=1}^n y_i \\ \text{s. t.} \quad & \mathbf{S}\mathbf{v} = 0 \\ & \alpha_i \leq v_i \leq \beta_i \quad \forall i \in \{1, \dots, n\} \\ & v_i - y(\beta_i - w_i^u) \leq w_i^u \\ & v_i - y(\alpha_i - w_i^l) \geq w_i^l \\ & y_i \in \{0, 1\} \\ & w_i^u = w_i + \delta|w_i| + \epsilon \\ & w_i^l = w_i - \delta|w_i| - \epsilon \end{aligned}$$

with \mathbf{S} being the stoichiometric matrix, α_i and β_i being the lower and upper bounds for flux (variable) i . The total number of fluxes of the optimization problem is denoted by n . Here, w is the optimal wild type flux distribution. This can be set via the argument `wtflux`. If `wtflux` is NULL (the default), the wild type flux distribution will be calculated by a standard FBA. All variables y_i are binary, with $y_i = 1$ for a significant flux change in v_i and $y_i = 0$ otherwise. Thresholds determining the significance of a flux change are given in w^u and w^l , with δ and ϵ specifying absolute and relative ranges in tolerance [Shlomi et al. 2005].

The Boolean argument `LPvariant` relax the binary constraints to $0 \leq y_i \leq 1$ so that the problem becomes a linear program. The optimization can be executed by using [optimizeProb](#).

Objects from the Class

Objects can be created by calls of the form

```
sysBiolAlg(model, algorithm = "room", ...).
```

Arguments to ... which are passed to method `initialize` of class `sysBiolAlg_room` are described in the Details section.

Slots

wu: Object of class "numeric" containing the upper threshold for a significant flux change, see Details below.

wl: Object of class "numeric" containing the lower threshold for a significant flux change, see Details below.

fnc: Object of class "integer" containing the number of reactions in the entire metabolic network (argument model to the constructor function [sysBiolAlg](#)).

fnr: Object of class "integer" containing the number of metabolites in the entire metabolic network (argument model to the constructor function [sysBiolAlg](#)).

problem: Object of class "optObj" containing the problem object.

algorithm: Object of class "character" containing the name of the algorithm.

nr: Object of class "integer" containing the number of rows of the problem object.

nc: Object of class "integer" containing the number of columns of the problem object

fldind: Object of class "integer" pointers to columns (variables) representing a flux (reaction) in the original network. The variable fldind[i] in the problem object represents reaction i in the original network.

alg_par: Object of class "list" containing a named list containing algorithm specific parameters.

Extends

Class "[sysBiolAlg](#)", directly.

Methods

optimizeProb signature(object = "sysBiolAlg_room"): runs optimization on the given problem object (see [optimizeProb](#) for details).

Note

If using **glpkAPI** as MIP solver, consider to set parameter PRESOLVE to GLP_ON.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzscheier <clausjonathan.fritzscheier@uni-duesseldorf.de>

References

Shlomi, T., Berkman, O. and Ruppin, E. (2005) Regulatory on/off minimization of metabolic flux changes after genetic perturbations. *PNAS* **102**, 7695–7700.

See Also

Constructor function [sysBiolAlg](#) and superclass [sysBiolAlg](#).

Examples

```
showClass("sysBiolAlg_room")
```

upgradeModelorg *Upgrade modelorg to newer version.*

Description

Performs necessary changes to the object to promote it to a newer version.

Usage

```
upgradeModelorg(object)
```

Arguments

object An object of class `modelorg`.

Details

This method performs the necessary changes on a `modelorg` object to promote it to a newer version.

Changes from previous `modelorg` version (no version slot set) to version 2.0: Representation in the `gprRules` slot is now incompatible to the earlier versions.

Value

An object of class `modelorg`, matching the current version requirements used by `sybil`.

Author(s)

Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

Examples

```
data(Ec_core)
upgradeModelorg(Ec_core)
```

writeProb-methods *Write Problem Object to File*

Description

Write problem object to file (e.g. in lp format).

Usage

```
## S4 method for signature 'optObj_clpAPI,character'
writeProb(lp, fname, ff = "lp")

## S4 method for signature 'optObj_cplexAPI,character'
writeProb(lp, fname, ff = "lp")

## S4 method for signature 'optObj_glpkAPI,character'
writeProb(lp, fname, ff = "lp", ...)

## S4 method for signature 'optObj_lpSolveAPI,character'
writeProb(lp, fname, ff = "lp", ...)
```

Arguments

lp	An object extending class <code>optObj</code> .
fname	A single character string giving the file name to write to.
ff	A single character string giving the file format to use, see Details. Default: "lp".
...	Further arguments passed to the corresponding API routine.

Details

Argument "ff" is unused with **clpAPI**. Valid values for **cplexAPI** and **lpSolveAPI** are available in their documentations. For **glpkAPI**, argument "ff" can be "lp" for LP file format, "mps" for MPS file format or "glpk" for GLPK file format.

Methods

```
signature(lp = "optObj_clpAPI", fname = "character") method to use with package optObj_clpAPI. Argument ff is not used here.
signature(lp = "optObj_cplexAPI", fname = "character") method to use with package optObj_cplexAPI.
signature(lp = "optObj_glpkAPI", fname = "character") method to use with package optObj_glpkAPI.
signature(lp = "optObj_lpSolveAPI", fname = "character") method to use with package optObj_lpSolveAPI.
```

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

See Also

Superclass `optObj` and constructor function `optObj`. Method to read problem objects: `readProb`

Examples

```
## Not run:
# In very rare cases it is handy to save a sysBiolAlg-object:

library(sybil)
data(Ec_core)
# create a sysBiolAlg object (we use here GLPK (!))
prob <- sysBiolAlg(Ec_core, algorithm = "fba", solver="glpkAPI")

# write the R-object to disc
save(file="prob.RData",prob)

# now write the linear program part (managed by the solver) to disc
writeProb(prob@problem, fname="prob.lp", ff="lp")

# start new R session

library(sybil)
library(glpkAPI)
load("prob.RData") # restore the R-object
prob@problem@oobj <- initProbGLPK() # initialize a new linear program
readProb(prob@problem, fname="prob.lp") # load the previously saved linear program

## End(Not run)
```

y_{pd}
In Sillico YPD Medium

Description

Apply in sillico medium to bakers yeast metabolic network model iND750 by Duarte et al. 2004.

Usage

```
ypd(model, def_bnd = SYBIL_SETTINGS("MAXIMUM"), ver = "harrison2007")
```

Arguments

model	An object of class modelorg .
def_bnd	A single numeric value. Absolute value for upper and lower bounds for reaction bounds. Default: SYBIL_SETTINGS("MAXIMUM").
ver	A single character string giving the version of the YPD medium. Can be set to harrison2007 or bilu2006 (see Details below). Default: harrison2007.

Details

The function `ypd` identifies exchange reactions via the function `findExchReact`. The lower bounds of all exchange fluxes is set to zero (not allowing any flux into the network) and the upper bounds are set to the value of `def_bnd` (default: output is unbounded). The lower bound input of the input fluxes is set like in the table below.

Two different versions of YPD medium are available: Harrison et al. 2007 and Bilu et al. 2006.

Harrison et al 2007:

EX_ala_L(e)	-0.5
EX_arg_L(e)	-0.5
EX_asn_L(e)	-0.5
EX_asp_L(e)	-0.5
EX_chol(e)	-0.5
EX_cys_L(e)	-0.5
EX_dcyt(e)	-0.5
EX_ergst(e)	-0.5
EX_glc(e)	-20
EX_glu_L(e)	-0.5
EX_gly(e)	-0.5
EX_gua(e)	-0.5
EX_h(e)	def_bnd * -1
EX_hdca(e)	-0.5
EX_his_L(e)	-0.5
EX_leu_L(e)	-0.5
EX_lys_L(e)	-0.5
EX_met_L(e)	-0.5
EX_nh4(e)	def_bnd * -1
EX_o2(e)	-2
EX_ocdca(e)	-0.5
EX_pi(e)	def_bnd * -1
EX_pro_L(e)	-0.5
EX_ser_L(e)	-0.5
EX_so4(e)	def_bnd * -1
EX_thr_L(e)	-0.5
EX_thymd(e)	-0.5
EX_trp_L(e)	-0.5
EX_ttdca(e)	-0.5
EX_tyr_L(e)	-0.5
EX_ura(e)	-0.5

Bilu et al 2006:

EX_nh4(e)	def_bnd * -1
EX_pi(e)	def_bnd * -1
EX_so4(e)	def_bnd * -1
EX_glc(e)	-20
EX_o2(e)	-2

EX_ala_L(e)	-0.5
EX_arg_L(e)	-0.5
EX_asn_L(e)	-0.5
EX_asp_L(e)	-0.5
EX_cys_L(e)	-0.5
EX_his_L(e)	-0.5
EX_leu_L(e)	-0.5
EX_lys_L(e)	-0.5
EX_met_L(e)	-0.5
EX_pro_L(e)	-0.5
EX_ser_L(e)	-0.5
EX_thr_L(e)	-0.5
EX_trp_L(e)	-0.5
EX_tyr_L(e)	-0.5
EX_dcyt(e)	-0.5
EX_gly(e)	-0.5
EX_gua(e)	-0.5
EX_thymd(e)	-0.5
EX_h2o(e)	def_bnd * -1
EX_na1(e)	def_bnd * -1
EX_k(e)	def_bnd * -1
EX_co2(e)	def_bnd * -1
EX_ade(e)	-0.5
EX_gln_L(e)	-0.5
EX_ile_L(e)	-0.5
EX_phe_L(e)	-0.5
EX_val_L(e)	-0.5

Value

An instance of class [modelorg](#) with input fluxes set corresponding to the desired YPD medium.

Author(s)

Gabriel Gelius-Dietrich <geliudie@uni-duesseldorf.de>

Maintainer: Claus Jonathan Fritzeimer <clausjonathan.fritzeimer@uni-duesseldorf.de>

References

Harrison, R., Papp, B., Pal, C., Oliver, S. G. and Delnert, D. (2007) Plasticity of genetic interactions in metabolic networks of yeast. *PNAS* **104**, 2307–2312.

Bilu, Y., Shlomi, T., Barkai, N. and Ruppin, E. (2006) Conservation of expression and sequence of metabolic genes is reflected by activity across metabolic states. *PLoS Comput Biol* **2**, 932–938.

See Also

[modelorg](#), [findExchReact](#) and [SYBIL_SETTINGS](#)

Index

*Topic **IO**

modelorg2ExPA, [81](#)
modelorg2tsv, [82](#)
promptSysBiolAlg, [131](#)
readTSVmod, [137](#)

*Topic **change**

onlyChangeGPR, [92](#)

*Topic **character**

checkReactId, [34](#)

*Topic **check**

onlyCheckGPR, [93](#)

*Topic **classes**

checksol-class, [35](#)
fluxDistribution-class, [46](#)
modelorg-class, [77](#)
modelorg_irrev-class, [86](#)
netFlux-class, [88](#)
optObj, [102](#)
optObj-class, [103](#)
optObj_clpAPI-class, [106](#)
optObj_cplexAPI-class, [107](#)
optObj_glpkAPI-class, [108](#)
optObj_lpSolveAPI-class, [109](#)
optsol-class, [110](#)
optsol_blockedReact-class, [113](#)
optsol_fluxdel-class, [114](#)
optsol_fluxVar-class, [117](#)
optsol_genedel-class, [119](#)
optsol_optimizeProb-class, [121](#)
optsol_php-class, [122](#)
optsol_robAna-class, [124](#)
ppProc-class, [128](#)
reactId-class, [132](#)
reactId_Exch-class, [134](#)
summaryOptsol, [158](#)
summaryOptsol-class, [159](#)
sybilError-class, [162](#)
sybilLog-class, [163](#)
sysBiolAlg, [169](#)

sysBiolAlg-class, [171](#)

sysBiolAlg_fba-class, [174](#)

sysBiolAlg_fbaEasyConstraint-class,
[176](#)

sysBiolAlg_fv-class, [178](#)

sysBiolAlg_lmoma-class, [181](#)

sysBiolAlg_moma-class, [184](#)

sysBiolAlg_mtf-class, [186](#)

sysBiolAlg_room-class, [189](#)

*Topic **datasets**

Ec_core, [43](#)

*Topic **manip**

addExchReact, [10](#)

addReact, [11](#)

doubleReact, [42](#)

mod2irrev, [76](#)

rmReact, [145](#)

*Topic **methods**

addCols-methods, [7](#)

addColsToProb-methods, [8](#)

addRows-methods, [13](#)

addRowsCols-methods, [14](#)

addRowsToProb-methods, [15](#)

applyChanges-methods, [18](#)

backupProb-methods, [20](#)

changeColsBnds-methods, [23](#)

changeColsBndsObjCoefs-methods, [24](#)

changeMatrixRow-methods, [26](#)

changeObjCoefs-methods, [27](#)

changeRowsBnds-methods, [29](#)

changeUptake-methods, [30](#)

checkVersion-methods, [36](#)

deadEndMetabolites-methods, [37](#)

delProb-methods, [38](#)

getColPrim-methods, [51](#)

getColsLowBnds-methods, [52](#)

getColsNames-methods, [53](#)

getColsUpBnds-methods, [54](#)

getFluxDist-methods, [55](#)

- getNumCols-methods, 56
- getNumNnz-methods, 57
- getNumRows-methods, 58
- getObjCoefs-methods, 59
- getObjDir-methods, 60
- getObjVal-methods, 61
- getRedCosts-methods, 62
- getRowsLowBnds-methods, 63
- getRowsNames-methods, 64
- getRowsUpBnds-methods, 65
- getSolStat-methods, 66
- getSolverParm-methods, 67
- initProb-methods, 69
- loadLPprob-methods, 70
- loadQobj-methods, 75
- optimizeProb-methods, 94
- printMetabolite-methods, 129
- printReaction-methods, 130
- readProb-methods, 136
- resetChanges-methods, 144
- scaleProb-methods, 147
- sensitivityAnalysis-methods, 148
- setColsNames-methods, 149
- setObjDir-methods, 150
- setRhsZero-methods, 152
- setRowsNames-methods, 153
- setSolverParm-methods, 154
- shrinkMatrix-methods, 155
- singletonMetabolites-methods, 156
- solveLp-methods, 157
- writeProb-methods, 192
- *Topic **optimize**
 - addCols-methods, 7
 - addColsToProb-methods, 8
 - addRows-methods, 13
 - addRowsCols-methods, 14
 - addRowsToProb-methods, 15
 - applyChanges-methods, 18
 - backupProb-methods, 20
 - blockedReact, 21
 - changeBounds, 22
 - changeColsBnds-methods, 23
 - changeColsBndsObjCoefs-methods, 24
 - changeMatrixRow-methods, 26
 - changeObjCoefs-methods, 27
 - changeObjFunc, 28
 - changeRowsBnds-methods, 29
 - checkDefaultMethod, 31
 - checkOptSol-methods, 33
 - delProb-methods, 38
 - doubleFluxDel, 39
 - doubleGeneDel, 40
 - fluxVar, 47
 - geneDeletion, 49
 - getColPrim-methods, 51
 - getColsLowBnds-methods, 52
 - getColsNames-methods, 53
 - getColsUpBnds-methods, 54
 - getFluxDist-methods, 55
 - getNumCols-methods, 56
 - getNumNnz-methods, 57
 - getNumRows-methods, 58
 - getObjCoefs-methods, 59
 - getObjDir-methods, 60
 - getObjVal-methods, 61
 - getRedCosts-methods, 62
 - getRowsLowBnds-methods, 63
 - getRowsNames-methods, 64
 - getRowsUpBnds-methods, 65
 - getSolStat-methods, 66
 - getSolverParm-methods, 67
 - initProb-methods, 69
 - loadLPprob-methods, 70
 - loadQobj-methods, 75
 - oneFluxDel, 89
 - oneGeneDel, 91
 - optimizeProb-methods, 94
 - optimizer, 99
 - phppp, 126
 - readProb-methods, 136
 - resetChanges-methods, 144
 - robAna, 146
 - scaleProb-methods, 147
 - sensitivityAnalysis-methods, 148
 - setColsNames-methods, 149
 - setObjDir-methods, 150
 - setRhsZero-methods, 152
 - setRowsNames-methods, 153
 - setSolverParm-methods, 154
 - solveLp-methods, 157
 - SYBIL_SETTINGS, 167
 - sysBioAlg-class, 171
 - writeProb-methods, 192
- *Topic **package**
 - sybil-package, 5
- *Topic **upgrade, version**

- upgradeModelorg, [192](#)
- [, [56](#)
- [, fluxDistribution, ANY, ANY, ANY-method (fluxDistribution-class), [46](#)
- [, optsol_fluxdel, ANY, ANY, ANY-method (optsol_fluxdel-class), [114](#)
- [, reactId, ANY, ANY, ANY-method (reactId-class), [132](#)
- [, reactId_Exch, ANY, ANY, ANY-method (reactId_Exch-class), [134](#)

- addAlgorithm, [7, 31](#)
- addCols, [105](#)
- addCols (addCols-methods), [7](#)
- addCols, optObj_clpAPI, numeric-method (addCols-methods), [7](#)
- addCols, optObj_cplexAPI, numeric-method (addCols-methods), [7](#)
- addCols, optObj_glpkAPI, numeric-method (addCols-methods), [7](#)
- addCols, optObj_lpSolveAPI, numeric-method (addCols-methods), [7](#)
- addCols-methods, [7](#)
- addColsToProb, [105](#)
- addColsToProb (addColsToProb-methods), [8](#)
- addColsToProb, optObj_clpAPI-method (addColsToProb-methods), [8](#)
- addColsToProb, optObj_cplexAPI-method (addColsToProb-methods), [8](#)
- addColsToProb, optObj_glpkAPI-method (addColsToProb-methods), [8](#)
- addColsToProb, optObj_lpSolveAPI-method (addColsToProb-methods), [8](#)
- addColsToProb-methods, [8](#)
- addExchReact, [10](#)
- addReact, [11, 11](#)
- addRows, [105](#)
- addRows (addRows-methods), [13](#)
- addRows, optObj_clpAPI, numeric-method (addRows-methods), [13](#)
- addRows, optObj_cplexAPI, numeric-method (addRows-methods), [13](#)
- addRows, optObj_glpkAPI, numeric-method (addRows-methods), [13](#)
- addRows, optObj_lpSolveAPI, numeric-method (addRows-methods), [13](#)
- addRows-methods, [13](#)
- addRowsCols, [105](#)
- addRowsCols (addRowsCols-methods), [14](#)
- addRowsCols, optObj_clpAPI, numeric, numeric-method (addRowsCols-methods), [14](#)
- addRowsCols, optObj_cplexAPI, numeric, numeric-method (addRowsCols-methods), [14](#)
- addRowsCols, optObj_glpkAPI, numeric, numeric-method (addRowsCols-methods), [14](#)
- addRowsCols, optObj_lpSolveAPI, numeric, numeric-method (addRowsCols-methods), [14](#)
- addRowsCols-methods, [14](#)
- addRowsToProb, [105](#)
- addRowsToProb (addRowsToProb-methods), [15](#)
- addRowsToProb, optObj_clpAPI-method (addRowsToProb-methods), [15](#)
- addRowsToProb, optObj_cplexAPI-method (addRowsToProb-methods), [15](#)
- addRowsToProb, optObj_glpkAPI-method (addRowsToProb-methods), [15](#)
- addRowsToProb, optObj_lpSolveAPI-method (addRowsToProb-methods), [15](#)
- addRowsToProb-methods, [15](#)
- addSolver, [17, 69](#)
- alg_par (sysBiolAlg-class), [171](#)
- alg_par, optsol-method (optsol-class), [110](#)
- alg_par, sysBiolAlg-method (sysBiolAlg-class), [171](#)
- alg_par<- (sysBiolAlg-class), [171](#)
- alg_par<-, optsol-method (optsol-class), [110](#)
- alg_par<-, sysBiolAlg-method (sysBiolAlg-class), [171](#)
- ALGORITHM (SYBIL_SETTINGS), [167](#)
- algorithm (optsol-class), [110](#)
- algorithm, optsol-method (optsol-class), [110](#)
- algorithm, sysBiolAlg-method (sysBiolAlg-class), [171](#)
- algorithm<- (optsol-class), [110](#)
- algorithm<-, optsol-method (optsol-class), [110](#)
- algorithm<-, sysBiolAlg-method (sysBiolAlg-class), [171](#)
- allGenes, [48, 95](#)
- allGenes (modelorg-class), [77](#)
- allGenes, modelorg-method (modelorg-class), [77](#)
- allGenes<- (modelorg-class), [77](#)

- allGenes<- , modelorg-method
(modelorg-class), 77
- applyChanges, 98, 144
- applyChanges (applyChanges-methods), 18
- applyChanges, sysBioAlg-method
(applyChanges-methods), 18
- applyChanges, sysBioAlg_room-method
(applyChanges-methods), 18
- applyChanges-methods, 18
- backupProb, 105
- backupProb (backupProb-methods), 20
- backupProb, optObj_clpAPI-method
(backupProb-methods), 20
- backupProb, optObj_cplexAPI-method
(backupProb-methods), 20
- backupProb, optObj_glpkAPI-method
(backupProb-methods), 20
- backupProb, optObj_lpSolveAPI-method
(backupProb-methods), 20
- backupProb-methods, 20
- blocked (optsol_blockedReact-class), 113
- blocked, optsol_blockedReact-method
(optsol_blockedReact-class),
113
- blocked<- (optsol_blockedReact-class),
113
- blocked<- , optsol_blockedReact-method
(optsol_blockedReact-class),
113
- blockedReact, 21, 113, 161
- blReact (optsol_fluxVar-class), 117
- blReact, optsol_fluxVar-method
(optsol_fluxVar-class), 117
- callNextMethod, 132, 173
- cat, 129, 130
- changeBounds, 22
- changeColsBnds, 105
- changeColsBnds
(changeColsBnds-methods), 23
- changeColsBnds, optObj_clpAPI-method
(changeColsBnds-methods), 23
- changeColsBnds, optObj_cplexAPI-method
(changeColsBnds-methods), 23
- changeColsBnds, optObj_glpkAPI-method
(changeColsBnds-methods), 23
- changeColsBnds, optObj_lpSolveAPI-method
(changeColsBnds-methods), 23
- changeColsBnds-methods, 23
- changeColsBndsObjCoefs, 105
- changeColsBndsObjCoefs
(changeColsBndsObjCoefs-methods),
24
- changeColsBndsObjCoefs, optObj_clpAPI-method
(changeColsBndsObjCoefs-methods),
24
- changeColsBndsObjCoefs, optObj_cplexAPI-method
(changeColsBndsObjCoefs-methods),
24
- changeColsBndsObjCoefs, optObj_glpkAPI-method
(changeColsBndsObjCoefs-methods),
24
- changeColsBndsObjCoefs, optObj_lpSolveAPI-method
(changeColsBndsObjCoefs-methods),
24
- changeColsBndsObjCoefs-methods, 24
- changeGPR, 25
- changeMatrixRow, 105
- changeMatrixRow
(changeMatrixRow-methods), 26
- changeMatrixRow, optObj_cplexAPI-method
(changeMatrixRow-methods), 26
- changeMatrixRow, optObj_glpkAPI-method
(changeMatrixRow-methods), 26
- changeMatrixRow, optObj_lpSolveAPI-method
(changeMatrixRow-methods), 26
- changeMatrixRow-methods, 26
- changeMaxObj (sysBioAlg_mtf-class), 186
- changeMaxObj, sysBioAlg_mtf-method
(sysBioAlg_mtf-class), 186
- changeObjCoefs, 105
- changeObjCoefs
(changeObjCoefs-methods), 27
- changeObjCoefs, optObj_clpAPI-method
(changeObjCoefs-methods), 27
- changeObjCoefs, optObj_cplexAPI-method
(changeObjCoefs-methods), 27
- changeObjCoefs, optObj_glpkAPI-method
(changeObjCoefs-methods), 27
- changeObjCoefs, optObj_lpSolveAPI-method
(changeObjCoefs-methods), 27
- changeObjCoefs-methods, 27
- changeObjFunc, 28
- changeRowsBnds, 105
- changeRowsBnds
(changeRowsBnds-methods), 29

- changeRowsBnds, optObj_clpAPI-method
(changeRowsBnds-methods), 29
- changeRowsBnds, optObj_cplexAPI-method
(changeRowsBnds-methods), 29
- changeRowsBnds, optObj_glpkAPI-method
(changeRowsBnds-methods), 29
- changeRowsBnds, optObj_lpSolveAPI-method
(changeRowsBnds-methods), 29
- changeRowsBnds-methods, 29
- changeUptake (changeUptake-methods), 30
- changeUptake, modelorg-method
(changeUptake-methods), 30
- changeUptake-methods, 30
- checkAlgorithm, 7, 31
- checkDefaultMethod, 31, 69, 103, 106, 167, 169
- checkOptSol, 35, 36, 40, 42, 50, 90, 92, 112, 114, 116, 119, 120, 122, 124, 126
- checkOptSol (checkOptSol-methods), 33
- checkOptSol, optsol-method
(checkOptSol-methods), 33
- checkOptSol-methods, 33
- checkReactId, 22, 23, 28, 34, 44, 132, 133, 135, 145
- checksol, 33
- checksol (checksol-class), 35
- checksol-class, 35
- checkSolStat (optObj-class), 103
- checkStat (optsol-class), 110
- checkStat, optsol-method (optsol-class), 110
- checkVersion (checkVersion-methods), 36
- checkVersion, modelorg
(checkVersion-methods), 36
- checkVersion, modelorg-method
(checkVersion-methods), 36
- checkVersion-methods, 36
- chlb (optsol_fluxdel-class), 114
- chlb, optsol_fluxdel-method
(optsol_fluxdel-class), 114
- chlb<- (optsol_fluxdel-class), 114
- chlb<-, optsol_fluxdel-method
(optsol_fluxdel-class), 114
- chub (optsol_fluxdel-class), 114
- chub, optsol_fluxdel-method
(optsol_fluxdel-class), 114
- chub<- (optsol_fluxdel-class), 114
- chub<-, optsol_fluxdel-method
(optsol_fluxdel-class), 114
- clpPtr-class (optObj-class), 103
- cmd (ppProc-class), 128
- cmd, ppProc-method (ppProc-class), 128
- cmd<- (ppProc-class), 128
- cmd<-, ppProc-method (ppProc-class), 128
- combn, 50
- comp_attr (modelorg-class), 77
- comp_attr, modelorg-method
(modelorg-class), 77
- comp_attr<- (modelorg-class), 77
- comp_attr<-, modelorg-method
(modelorg-class), 77
- cplexPointer-class (optObj-class), 103
- cplexPtr-class (optObj-class), 103
- ctrlfl (optsol_robAna-class), 124
- ctrlfl, optsol_phpp-method
(optsol_phpp-class), 122
- ctrlfl, optsol_robAna-method
(optsol_robAna-class), 124
- ctrlfl<- (optsol_robAna-class), 124
- ctrlfl<-, optsol_phpp-method
(optsol_phpp-class), 122
- ctrlfl<-, optsol_robAna-method
(optsol_robAna-class), 124
- ctrlr (optsol_robAna-class), 124
- ctrlr, optsol_robAna-method
(optsol_robAna-class), 124
- ctrlr<- (optsol_robAna-class), 124
- ctrlr<-, optsol_robAna-method
(optsol_robAna-class), 124
- deadEndMetabolites
(deadEndMetabolites-methods), 37
- deadEndMetabolites, modelorg-method
(deadEndMetabolites-methods), 37
- deadEndMetabolites-methods, 37
- deleted (optsol_fluxdel-class), 114
- deleted, optsol_fluxdel-method
(optsol_fluxdel-class), 114
- deleted, optsol_genedel-method
(optsol_genedel-class), 119
- delProb, 105
- delProb (delProb-methods), 38
- delProb, optObj_clpAPI-method
(delProb-methods), 38

- delProb, optObj_cplexAPI-method
(delProb-methods), 38
- delProb, optObj_glpkAPI-method
(delProb-methods), 38
- delProb, optObj_lpSolveAPI-method
(delProb-methods), 38
- delProb-methods, 38
- dels (optsol_fluxdel-class), 114
- dels, optsol_fluxdel-method
(optsol_fluxdel-class), 114
- dels<- (optsol_fluxdel-class), 114
- dels<-, optsol_fluxdel-method
(optsol_fluxdel-class), 114
- Deprecated, 161
- didFoot (sybilLog-class), 163
- didFoot, sybilLog-method
(sybilLog-class), 163
- didFoot<- (sybilLog-class), 163
- didFoot<-, sybilLog-method
(sybilLog-class), 163
- dim, modelorg-method (modelorg-class), 77
- dim, optObj-method (optObj-class), 103
- doubleFluxDel, 39, 87, 88
- doubleGeneDel, 40, 50, 87, 88
- doubleReact, 42
- EasyConstraint
(sysBiolAlg_fbaEasyConstraint-class),
176
- Ec_core, 43
- edit, 44
- editEnvir, 44
- emsg (sybilError-class), 162
- emsg, sybilError-method
(sybilError-class), 162
- emsg<- (sybilError-class), 162
- emsg<-, sybilError-method
(sybilError-class), 162
- enum (sybilError-class), 162
- enum, sybilError-method
(sybilError-class), 162
- enum<- (sybilError-class), 162
- enum<-, sybilError-method
(sybilError-class), 162
- ex_met (summaryOptsol-class), 159
- ex_met, summaryOptsol-method
(summaryOptsol-class), 159
- ex_val (summaryOptsol-class), 159
- ex_val, summaryOptsol-method
(summaryOptsol-class), 159
- exit_code (checksol-class), 35
- exit_code, checksol-method
(checksol-class), 35
- exit_code<- (checksol-class), 35
- exit_code<-, checksol-method
(checksol-class), 35
- exit_meaning (checksol-class), 35
- exit_meaning, checksol-method
(checksol-class), 35
- exit_meaning<- (checksol-class), 35
- exit_meaning<-, checksol-method
(checksol-class), 35
- exit_num (checksol-class), 35
- exit_num, checksol-method
(checksol-class), 35
- exit_num<- (checksol-class), 35
- exit_num<-, checksol-method
(checksol-class), 35
- externalptr, 106
- fba, 95, 100, 168, 186
- fba (sysBiolAlg_fba-class), 174
- fbaEasyConstraint
(sysBiolAlg_fbaEasyConstraint-class),
176
- fenc (sybilLog-class), 163
- fenc, sybilLog-method (sybilLog-class),
163
- fenc<- (sybilLog-class), 163
- fenc<-, sybilLog-method
(sybilLog-class), 163
- fh (sybilLog-class), 163
- fh, sybilLog-method (sybilLog-class), 163
- fh<- (sybilLog-class), 163
- fh<-, sybilLog-method (sybilLog-class),
163
- file, 132, 163
- file-class (sybilLog-class), 163
- findExchReact, 21, 39, 45, 134, 195, 196
- fldind (optsol-class), 110
- fldind, optsol-method (optsol-class), 110
- fldind, sysBiolAlg-method
(sysBiolAlg-class), 171
- fldind<- (optsol-class), 110
- fldind<-, optsol-method (optsol-class),
110

- fldind<- ,sysBiolAlg-method
(sysBiolAlg-class), 171
- fluxdels (optsol_genedel-class), 119
- fluxdels, optsol_genedel-method
(optsol_genedel-class), 119
- fluxdels<- (optsol_genedel-class), 119
- fluxdels<- ,optsol_genedel-method
(optsol_genedel-class), 119
- fluxdist (optsol-class), 110
- fluxdist, optsol-method (optsol-class),
110
- fluxdist<- (optsol-class), 110
- fluxdist<- ,optsol-method
(optsol-class), 110
- fluxDistribution
(fluxDistribution-class), 46
- fluxDistribution-class, 46
- fluxes (optsol-class), 110
- fluxes, fluxDistribution-method
(fluxDistribution-class), 46
- fluxes, optsol-method (optsol-class), 110
- fluxes<- (optsol-class), 110
- fluxes<- ,fluxDistribution-method
(fluxDistribution-class), 46
- fluxes<- ,optsol-method (optsol-class),
110
- fluxVar, 47, 87, 88, 117
- fname (sybilLog-class), 163
- fname, sybilLog-method (sybilLog-class),
163
- fname<- (sybilLog-class), 163
- fname<- ,sybilLog-method
(sybilLog-class), 163
- fpath (sybilLog-class), 163
- fpath, sybilLog-method (sybilLog-class),
163
- fpath<- (sybilLog-class), 163
- fpath<- ,sybilLog-method
(sybilLog-class), 163
- fv, 47, 100, 168
- fv (sysBiolAlg_fv-class), 178

- geneDel, 41, 48, 91
- geneDeletion, 49
- genes (modelorg-class), 77
- genes, modelorg-method (modelorg-class),
77
- genes<- (modelorg-class), 77

- genes<- ,modelorg-method
(modelorg-class), 77
- getColPrim, 105
- getColPrim (getColPrim-methods), 51
- getColPrim, optObj_clpAPI, numeric-method
(getColPrim-methods), 51
- getColPrim, optObj_cplexAPI, numeric-method
(getColPrim-methods), 51
- getColPrim, optObj_glpkAPI, numeric-method
(getColPrim-methods), 51
- getColPrim, optObj_lpSolveAPI, numeric-method
(getColPrim-methods), 51
- getColPrim-methods, 51
- getColsLowBnds, 105
- getColsLowBnds
(getColsLowBnds-methods), 52
- getColsLowBnds, optObj_clpAPI, numeric-method
(getColsLowBnds-methods), 52
- getColsLowBnds, optObj_cplexAPI, numeric-method
(getColsLowBnds-methods), 52
- getColsLowBnds, optObj_glpkAPI, numeric-method
(getColsLowBnds-methods), 52
- getColsLowBnds, optObj_lpSolveAPI, numeric-method
(getColsLowBnds-methods), 52
- getColsLowBnds-methods, 52
- getColsNames (getColsNames-methods), 53
- getColsNames, optObj_cplexAPI, numeric-method
(getColsNames-methods), 53
- getColsNames, optObj_glpkAPI, numeric-method
(getColsNames-methods), 53
- getColsNames, optObj_lpSolveAPI, numeric-method
(getColsNames-methods), 53
- getColsNames-methods, 53
- getColsUppBnds, 105
- getColsUppBnds
(getColsUppBnds-methods), 54
- getColsUppBnds, optObj_clpAPI, numeric-method
(getColsUppBnds-methods), 54
- getColsUppBnds, optObj_cplexAPI, numeric-method
(getColsUppBnds-methods), 54
- getColsUppBnds, optObj_glpkAPI, numeric-method
(getColsUppBnds-methods), 54
- getColsUppBnds, optObj_lpSolveAPI, numeric-method
(getColsUppBnds-methods), 54
- getColsUppBnds-methods, 54
- getFluxDist, 89, 105
- getFluxDist (getFluxDist-methods), 55
- getFluxDist, optObj_clpAPI-method

- (getFluxDist-methods), 55
- getFluxDist, optObj_cplexAPI-method
(getFluxDist-methods), 55
- getFluxDist, optObj_glpkAPI-method
(getFluxDist-methods), 55
- getFluxDist, optObj_lpSolveAPI-method
(getFluxDist-methods), 55
- getFluxDist, optsol-method
(getFluxDist-methods), 55
- getFluxDist-methods, 55
- getMeanReturn (optObj-class), 103
- getMeanStatus, 67
- getMeanStatus (optObj-class), 103
- getNetFlux (netFlux-class), 88
- getNumCols, 105
- getNumCols (getNumCols-methods), 56
- getNumCols, optObj_clpAPI-method
(getNumCols-methods), 56
- getNumCols, optObj_cplexAPI-method
(getNumCols-methods), 56
- getNumCols, optObj_glpkAPI-method
(getNumCols-methods), 56
- getNumCols, optObj_lpSolveAPI-method
(getNumCols-methods), 56
- getNumCols-methods, 56
- getNumNnz, 105
- getNumNnz (getNumNnz-methods), 57
- getNumNnz, optObj_clpAPI-method
(getNumNnz-methods), 57
- getNumNnz, optObj_cplexAPI-method
(getNumNnz-methods), 57
- getNumNnz, optObj_glpkAPI-method
(getNumNnz-methods), 57
- getNumNnz-methods, 57
- getNumRows, 105
- getNumRows (getNumRows-methods), 58
- getNumRows, optObj_clpAPI-method
(getNumRows-methods), 58
- getNumRows, optObj_cplexAPI-method
(getNumRows-methods), 58
- getNumRows, optObj_glpkAPI-method
(getNumRows-methods), 58
- getNumRows, optObj_lpSolveAPI-method
(getNumRows-methods), 58
- getNumRows-methods, 58
- getObjCoefs, 105
- getObjCoefs (getObjCoefs-methods), 59
- getObjCoefs, optObj_clpAPI, numeric-method
(getObjCoefs-methods), 59
- getObjCoefs, optObj_cplexAPI, numeric-method
(getObjCoefs-methods), 59
- getObjCoefs, optObj_glpkAPI, numeric-method
(getObjCoefs-methods), 59
- getObjCoefs, optObj_lpSolveAPI, numeric-method
(getObjCoefs-methods), 59
- getObjCoefs-methods, 59
- getObjDir, 105
- getObjDir (getObjDir-methods), 60
- getObjDir, optObj_clpAPI-method
(getObjDir-methods), 60
- getObjDir, optObj_cplexAPI-method
(getObjDir-methods), 60
- getObjDir, optObj_glpkAPI-method
(getObjDir-methods), 60
- getObjDir, optObj_lpSolveAPI-method
(getObjDir-methods), 60
- getObjDir-methods, 60
- getObjVal, 105
- getObjVal (getObjVal-methods), 61
- getObjVal, optObj_clpAPI-method
(getObjVal-methods), 61
- getObjVal, optObj_cplexAPI-method
(getObjVal-methods), 61
- getObjVal, optObj_glpkAPI-method
(getObjVal-methods), 61
- getObjVal, optObj_lpSolveAPI-method
(getObjVal-methods), 61
- getObjVal-methods, 61
- getRedCosts, 105
- getRedCosts (getRedCosts-methods), 62
- getRedCosts, optObj_clpAPI-method
(getRedCosts-methods), 62
- getRedCosts, optObj_cplexAPI-method
(getRedCosts-methods), 62
- getRedCosts, optObj_glpkAPI-method
(getRedCosts-methods), 62
- getRedCosts, optObj_lpSolveAPI-method
(getRedCosts-methods), 62
- getRedCosts, optsol_php-method
(optsol_php-class), 122
- getRedCosts-methods, 62
- getRowsLowBnds, 105
- getRowsLowBnds
(getRowsLowBnds-methods), 63
- getRowsLowBnds, optObj_clpAPI, numeric-method
(getRowsLowBnds-methods), 63

- getRowsLowBnds, optObj_cplexAPI, numeric-method (getSolverParm-methods), 67
(getRowsLowBnds-methods), 63
- getRowsLowBnds, optObj_glpkAPI, numeric-method
(getRowsLowBnds-methods), 63
- getRowsLowBnds, optObj_lpSolveAPI, numeric-method
(getRowsLowBnds-methods), 63
- getRowsLowBnds-methods, 63
- getRowsNames (getRowsNames-methods), 64
- getRowsNames, optObj_cplexAPI, numeric-method
(getRowsNames-methods), 64
- getRowsNames, optObj_glpkAPI, numeric-method
(getRowsNames-methods), 64
- getRowsNames, optObj_lpSolveAPI, numeric-method
(getRowsNames-methods), 64
- getRowsNames-methods, 64
- getRowsUppBnds, 105
- getRowsUppBnds
(getRowsUppBnds-methods), 65
- getRowsUppBnds, optObj_clpAPI, numeric-method
(getRowsUppBnds-methods), 65
- getRowsUppBnds, optObj_cplexAPI, numeric-method
(getRowsUppBnds-methods), 65
- getRowsUppBnds, optObj_glpkAPI, numeric-method
(getRowsUppBnds-methods), 65
- getRowsUppBnds, optObj_lpSolveAPI, numeric-method
(getRowsUppBnds-methods), 65
- getRowsUppBnds-methods, 65
- getSolStat, 105
- getSolStat (getSolStat-methods), 66
- getSolStat, optObj_clpAPI-method
(getSolStat-methods), 66
- getSolStat, optObj_cplexAPI-method
(getSolStat-methods), 66
- getSolStat, optObj_glpkAPI-method
(getSolStat-methods), 66
- getSolStat, optObj_lpSolveAPI-method
(getSolStat-methods), 66
- getSolStat-methods, 66
- getSolverParm, 105
- getSolverParm (getSolverParm-methods),
67
- getSolverParm, optObj_clpAPI-method
(getSolverParm-methods), 67
- getSolverParm, optObj_cplexAPI-method
(getSolverParm-methods), 67
- getSolverParm, optObj_glpkAPI-method
(getSolverParm-methods), 67
- getSolverParm, optObj_lpSolveAPI-method
(getSolverParm-methods), 67
- getSolverParm-methods, 67
- getSolverParm (getSolverParm-methods), 67
- getsybilenv, 7, 31, 32, 68
- glpkPtr, 106
- glpkPtr-class (optObj-class), 103
- gpr (modelorg-class), 77
- gpr, modelorg-method (modelorg-class), 77
- gpr<- (modelorg-class), 77
- gpr<- , modelorg-method (modelorg-class),
77
- gprRules (modelorg-class), 77
- gprRules, modelorg-method
(modelorg-class), 77
- gprRules<- (modelorg-class), 77
- gprRules<- , modelorg-method
(modelorg-class), 77
- hasEffect (optsol_genedel-class), 119
- hasEffect, optsol_genedel-method
(optsol_genedel-class), 119
- hasEffect<- (optsol_genedel-class), 119
- hasEffect<- , optsol_genedel-method
(optsol_genedel-class), 119
- hist, 118
- histogram, 111, 160
- image, summaryOptsol-method
(summaryOptsol-class), 159
- ind (ppProc-class), 128
- ind, ppProc-method (ppProc-class), 128
- ind<- (ppProc-class), 128
- ind<- , ppProc-method (ppProc-class), 128
- initialize, 131, 132
- initProb, 74, 105, 171
- initProb (initProb-methods), 69
- initProb, optObj_clpAPI-method
(initProb-methods), 69
- initProb, optObj_cplexAPI-method
(initProb-methods), 69
- initProb, optObj_glpkAPI-method
(initProb-methods), 69
- initProb, optObj_lpSolveAPI-method
(initProb-methods), 69
- initProb-methods, 69
- irrev (modelorg_irrev-class), 86
- irrev, modelorg_irrev-method
(modelorg_irrev-class), 86
- irrev2rev (modelorg_irrev-class), 86

- irrev2rev,modelorg_irrev-method
(modelorg_irrev-class), 86
- irrev2rev<- (modelorg_irrev-class), 86
- irrev2rev<-,modelorg_irrev-method
(modelorg_irrev-class), 86
- irrev<- (modelorg_irrev-class), 86
- irrev<-,modelorg_irrev-method
(modelorg_irrev-class), 86

- length,netFlux-method (netFlux-class),
88
- length,optsol-method (optsol-class), 110
- length,reactId-method (reactId-class),
132
- lethal (optsol_fluxdel-class), 114
- lethal,optsol_fluxdel-method
(optsol_fluxdel-class), 114
- levelplot, 123, 124
- lmoma, 100, 168
- lmoma (sysBiolAlg_lmoma-class), 181
- loadLPprob, 105, 172, 177
- loadLPprob (loadLPprob-methods), 70
- loadLPprob,optObj_clpAPI-method
(loadLPprob-methods), 70
- loadLPprob,optObj_cplexAPI-method
(loadLPprob-methods), 70
- loadLPprob,optObj_glpkAPI-method
(loadLPprob-methods), 70
- loadLPprob,optObj_lpSolveAPI-method
(loadLPprob-methods), 70
- loadLPprob-methods, 70
- loadMatrixPerColumnLPSOLVE
(optObj_lpSolveAPI-class), 109
- loadQobj, 105
- loadQobj (loadQobj-methods), 75
- loadQobj,optObj_cplexAPI,Matrix-method
(loadQobj-methods), 75
- loadQobj,optObj_cplexAPI,numeric-method
(loadQobj-methods), 75
- loadQobj-methods, 75
- logCall (sybilLog-class), 163
- logCall,sybilLog-method
(sybilLog-class), 163
- logClose<- (sybilLog-class), 163
- logClose<-,sybilLog-method
(sybilLog-class), 163
- logComment (sybilLog-class), 163
- logComment,sybilLog-method
(sybilLog-class), 163

- logError (sybilLog-class), 163
- logError,sybilLog,ANY,ANY-method
(sybilLog-class), 163
- logError,sybilLog,ANY,numeric-method
(sybilLog-class), 163
- logError,sybilLog-method
(sybilLog-class), 163
- logFH (sybilLog-class), 163
- logFH,sybilLog-method (sybilLog-class),
163
- logFileFH (sybilLog-class), 163
- logFoot<- (sybilLog-class), 163
- logFoot<-,sybilLog-method
(sybilLog-class), 163
- logHead (sybilLog-class), 163
- logHead,sybilLog-method
(sybilLog-class), 163
- loglevel (sybilLog-class), 163
- loglevel,sybilLog-method
(sybilLog-class), 163
- loglevel<- (sybilLog-class), 163
- loglevel<-,sybilLog-method
(sybilLog-class), 163
- logMessage (sybilLog-class), 163
- logMessage,sybilLog-method
(sybilLog-class), 163
- logOptimization (sybilLog-class), 163
- logOptimization,sybilLog-method
(sybilLog-class), 163
- logOptimizationTH (sybilLog-class), 163
- logOptimizationTH,sybilLog-method
(sybilLog-class), 163
- logStep<- (sybilLog-class), 163
- logStep<-,sybilLog-method
(sybilLog-class), 163
- logWarning (sybilLog-class), 163
- logWarning,sybilLog-method
(sybilLog-class), 163

- lowbnd (modelorg-class), 77
- lowbnd,modelorg-method
(modelorg-class), 77
- lowbnd,reactId_Exch-method
(reactId_Exch-class), 134
- lowbnd<- (modelorg-class), 77
- lowbnd<-,modelorg-method
(modelorg-class), 77
- lowbnd<-,reactId_Exch-method
(reactId_Exch-class), 134

- lp_dir (optsol-class), 110
- lp_dir, optsol-method (optsol-class), 110
- lp_dir<- (optsol-class), 110
- lp_dir<-, optsol, character-method (optsol-class), 110
- lp_dir<-, optsol, factor-method (optsol-class), 110
- lp_dir<-, optsol, numeric-method (optsol-class), 110
- lp_num_cols (optsol-class), 110
- lp_num_cols, optsol-method (optsol-class), 110
- lp_num_cols<- (optsol-class), 110
- lp_num_cols<-, optsol-method (optsol-class), 110
- lp_num_rows (optsol-class), 110
- lp_num_rows, optsol-method (optsol-class), 110
- lp_num_rows<- (optsol-class), 110
- lp_num_rows<-, optsol-method (optsol-class), 110
- lp_obj (optsol-class), 110
- lp_obj, optsol-method (optsol-class), 110
- lp_obj<- (optsol-class), 110
- lp_obj<-, optsol-method (optsol-class), 110
- lp_ok (optsol-class), 110
- lp_ok, optsol-method (optsol-class), 110
- lp_ok<- (optsol-class), 110
- lp_ok<-, optsol-method (optsol-class), 110
- lp_stat (optsol-class), 110
- lp_stat, optsol-method (optsol-class), 110
- lp_stat<- (optsol-class), 110
- lp_stat<-, optsol-method (optsol-class), 110
- lpExtPtr-class (optObj-class), 103
- lstname (sybilLog-class), 163
- lstname, sybilLog-method (sybilLog-class), 163

- makeOptsolMO, 76, 121
- matchrev (modelorg_irrev-class), 86
- matchrev, modelorg_irrev-method (modelorg_irrev-class), 86
- matchrev<- (modelorg_irrev-class), 86
- matchrev<-, modelorg_irrev-method (modelorg_irrev-class), 86

- Matrix, 45, 56, 71, 75, 109, 156, 171, 184
- matrix, 45
- MAXIMUM (SYBIL_SETTINGS), 167
- maxSol (optsol_blockedReact-class), 113
- maxSol, optsol_blockedReact-method (optsol_blockedReact-class), 113
- maxSol, optsol_fluxVar-method (optsol_fluxVar-class), 117
- mclapply, 87, 88
- met_attr (modelorg-class), 77
- met_attr, modelorg-method (modelorg-class), 77
- met_attr<- (modelorg-class), 77
- met_attr<-, modelorg-method (modelorg-class), 77
- met_comp (modelorg-class), 77
- met_comp, modelorg-method (modelorg-class), 77
- met_comp<- (modelorg-class), 77
- met_comp<-, modelorg-method (modelorg-class), 77
- met_de (modelorg-class), 77
- met_de, modelorg-method (modelorg-class), 77
- met_de<- (modelorg-class), 77
- met_de<-, modelorg-method (modelorg-class), 77
- met_id (modelorg-class), 77
- met_id, modelorg-method (modelorg-class), 77
- met_id, reactId_Exch-method (reactId_Exch-class), 134
- met_id<- (modelorg-class), 77
- met_id<-, modelorg-method (modelorg-class), 77
- met_id<-, reactId_Exch-method (reactId_Exch-class), 134
- met_name (modelorg-class), 77
- met_name, modelorg-method (modelorg-class), 77
- met_name<- (modelorg-class), 77
- met_name<-, modelorg-method (modelorg-class), 77
- met_num (modelorg-class), 77
- met_num, modelorg-method (modelorg-class), 77
- met_num<- (modelorg-class), 77

- met_num<- , modelorg-method
(modelorg-class), 77
- met_pos (reactId_Exch-class), 134
- met_pos, reactId_Exch-method
(reactId_Exch-class), 134
- met_pos<- (reactId_Exch-class), 134
- met_pos<- , reactId_Exch-method
(reactId_Exch-class), 134
- met_single (modelorg-class), 77
- met_single, modelorg-method
(modelorg-class), 77
- met_single<- (modelorg-class), 77
- met_single<- , modelorg-method
(modelorg-class), 77
- METHOD (SYBIL_SETTINGS), 167
- method, 106
- method (optsol-class), 110
- method, optObj-method (optObj-class), 103
- method, optsol-method (optsol-class), 110
- method<- (optsol-class), 110
- method<- , optsol-method (optsol-class),
110
- minSol (optsol_blockedReact-class), 113
- minSol, optsol_blockedReact-method
(optsol_blockedReact-class),
113
- minSol, optsol_fluxVar-method
(optsol_fluxVar-class), 117
- mod2irrev, 76, 86
- mod_attr (modelorg-class), 77
- mod_attr, modelorg-method
(modelorg-class), 77
- mod_attr<- (modelorg-class), 77
- mod_attr<- , modelorg-method
(modelorg-class), 77
- mod_compart, 12
- mod_compart (modelorg-class), 77
- mod_compart, modelorg-method
(modelorg-class), 77
- mod_compart<- (modelorg-class), 77
- mod_compart<- , modelorg-method
(modelorg-class), 77
- mod_desc (modelorg-class), 77
- mod_desc, modelorg-method
(modelorg-class), 77
- mod_desc<- (modelorg-class), 77
- mod_desc<- , modelorg-method
(modelorg-class), 77
- mod_id (modelorg-class), 77
- mod_id, modelorg-method
(modelorg-class), 77
- mod_id, optsol-method (optsol-class), 110
- mod_id, reactId-method (reactId-class),
132
- mod_id, summaryOptsol-method
(summaryOptsol-class), 159
- mod_id<- (modelorg-class), 77
- mod_id<- , modelorg-method
(modelorg-class), 77
- mod_id<- , optsol-method (optsol-class),
110
- mod_id<- , reactId-method
(reactId-class), 132
- mod_id<- , summaryOptsol-method
(summaryOptsol-class), 159
- mod_key (modelorg-class), 77
- mod_key, modelorg-method
(modelorg-class), 77
- mod_key, optsol-method (optsol-class),
110
- mod_key, reactId-method (reactId-class),
132
- mod_key, summaryOptsol-method
(summaryOptsol-class), 159
- mod_key<- (modelorg-class), 77
- mod_key<- , modelorg-method
(modelorg-class), 77
- mod_key<- , optsol-method (optsol-class),
110
- mod_key<- , reactId-method
(reactId-class), 132
- mod_key<- , summaryOptsol-method
(summaryOptsol-class), 159
- mod_name (modelorg-class), 77
- mod_name, modelorg-method
(modelorg-class), 77
- mod_name<- (modelorg-class), 77
- mod_name<- , modelorg-method
(modelorg-class), 77
- mod_obj, 159
- mod_obj (optsol-class), 110
- mod_obj, optsol-method (optsol-class),
110
- mod_obj, summaryOptsol-method
(summaryOptsol-class), 159
- mod_obj<- (summaryOptsol-class), 159

- mod_obj<- ,summaryOptsol-method
(summaryOptsol-class), 159
- modelorg, 5, 10–13, 21, 22, 25, 28, 30, 34,
36–40, 42, 44, 45, 47, 49, 50, 76,
81–83, 86, 87, 90–95, 97, 98, 110,
113, 115, 117, 120, 121, 123, 125,
126, 129–131, 142, 143, 145, 146,
155–157, 159, 161, 170, 174, 179,
181, 184, 186, 189, 192, 194, 196
- modelorg (modelorg-class), 77
- modelorg-class, 77
- modelorg2ExPA, 81
- modelorg2tsv, 82, 86, 130, 140–143
- modelorg_irrev, 12, 77, 81
- modelorg_irrev (modelorg_irrev-class),
86
- modelorg_irrev-class, 86
- moma, 100, 168
- moma (sysBiolAlg_moma-class), 184
- mtf, 19, 95, 100, 168
- mtf (sysBiolAlg_mtf-class), 186
- mtfEasyConstraint
(sysBiolAlg_fbaEasyConstraint-class),
176
- multiDel, 87

- nc (sysBiolAlg-class), 171
- nc, sysBiolAlg-method
(sysBiolAlg-class), 171
- nc<- (sysBiolAlg-class), 171
- nc<- , sysBiolAlg-method
(sysBiolAlg-class), 171
- netFlux (netFlux-class), 88
- netFlux-class, 88
- nfluxes (optsol-class), 110
- nfluxes, optsol-method (optsol-class),
110
- nnzero, fluxDistribution-method
(fluxDistribution-class), 46
- nnzero, summaryOptsol-method
(summaryOptsol-class), 159
- nr (sysBiolAlg-class), 171
- nr, sysBiolAlg-method
(sysBiolAlg-class), 171
- nr<- (sysBiolAlg-class), 171
- nr<- , sysBiolAlg-method
(sysBiolAlg-class), 171
- NULL, 45

- num_of_fluxes (fluxDistribution-class),
46
- num_of_fluxes, fluxDistribution-method
(fluxDistribution-class), 46
- num_of_prob (optsol-class), 110
- num_of_prob, checksol-method
(checksol-class), 35
- num_of_prob, optsol-method
(optsol-class), 110
- num_of_prob<- (optsol-class), 110
- num_of_prob<- , checksol-method
(checksol-class), 35
- num_of_prob<- , optsol-method
(optsol-class), 110
- nvar (fluxDistribution-class), 46
- nvar, fluxDistribution-method
(fluxDistribution-class), 46
- nzeros (summaryOptsol-class), 159
- nzeros, summaryOptsol-method
(summaryOptsol-class), 159

- obj_coef (modelorg-class), 77
- obj_coef, modelorg-method
(modelorg-class), 77
- obj_coef, optsol-method (optsol-class),
110
- obj_coef<- (modelorg-class), 77
- obj_coef<- , modelorg-method
(modelorg-class), 77
- obj_coef<- , optsol-method
(optsol-class), 110
- obj_func (optsol-class), 110
- obj_func, optsol-method (optsol-class),
110
- obj_func<- (optsol-class), 110
- obj_func<- , optsol-method
(optsol-class), 110
- oneFluxDel, 87, 88, 89, 114
- oneGeneDel, 33, 50, 87, 88, 91
- onlyChangeGPR, 92
- onlyCheckGPR, 93
- OPT_DIRECTION (SYBIL_SETTINGS), 167
- optimizeProb, 33, 50, 88, 89, 99, 101, 102,
115, 117, 119, 121, 122, 124, 128,
162, 173, 175, 176, 180, 183, 185,
188, 190, 191
- optimizeProb (optimizeProb-methods), 94
- optimizeProb, modelorg-method
(optimizeProb-methods), 94

- optimizeProb, sysBioAlg-method
 (optimizeProb-methods), 94
 optimizeProb-methods, 94
 optimizer, 39–42, 47–50, 76, 90–92, 96, 99,
 126, 128, 146
 optObj, 8, 9, 14–16, 20, 23–27, 29, 30, 32, 38,
 51–68, 70, 74, 75, 96, 97, 102, 102,
 103, 104, 106–110, 132, 136, 137,
 148–155, 158, 171, 173, 174, 176,
 178, 181, 184, 186, 189, 193
 optObj-class, 103
 optObj_clpAPI, 71, 73, 74
 optObj_clpAPI-class, 106
 optObj_cplexAPI, 71, 74
 optObj_cplexAPI-class, 107
 optObj_glpkAPI, 54, 65, 71, 74
 optObj_glpkAPI-class, 108
 optObj_lpSolveAPI, 71, 74
 optObj_lpSolveAPI-class, 109
 optsol, 33, 40, 42, 46, 50, 55, 56, 88, 90, 92,
 114–116, 118–120, 122–126, 128,
 158, 159, 161
 optsol (optsol-class), 110
 optsol-class, 110
 optsol_blockedReact, 21, 22
 optsol_blockedReact-class, 113
 optsol_fluxdel, 40, 90, 112, 120, 122
 optsol_fluxdel-class, 114
 optsol_fluxVar, 47, 112
 optsol_fluxVar-class, 117
 optsol_genedel, 41, 42, 50, 92, 112, 116, 122
 optsol_genedel-class, 119
 optsol_optimizeProb, 76, 96, 97, 112, 115,
 116, 118, 120, 123, 125
 optsol_optimizeProb-class, 121
 optsol_phpp, 127
 optsol_phpp-class, 122
 optsol_robAna, 112, 123, 147
 optsol_robAna-class, 124

 pa (ppProc-class), 128
 pa, ppProc-method (ppProc-class), 128
 pa<- (ppProc-class), 128
 pa<- , ppProc-method (ppProc-class), 128
 par, 118, 125, 126
 PATH_TO_MODEL (SYBIL_SETTINGS), 167
 phpp, 122, 124, 126
 plot, 118
 plot, fluxDistribution, missing-method
 (fluxDistribution-class), 46
 plot, optsol, missing-method
 (optsol-class), 110
 plot, optsol_fluxVar, missing-method
 (optsol_fluxVar-class), 117
 plot, optsol_phpp, character-method
 (optsol_phpp-class), 122
 plot, optsol_phpp, missing-method
 (optsol_phpp-class), 122
 plot, optsol_robAna, missing-method
 (optsol_robAna-class), 124
 plot, summaryOptsol, missing-method
 (summaryOptsol-class), 159
 plotRangeVar (optsol_fluxVar-class), 117
 plotRangeVar, optsol_fluxVar-method
 (optsol_fluxVar-class), 117
 pointerToProb (optObj-class), 103
 pointerToProb-class (optObj-class), 103
 points, 126
 postProc (optsol_optimizeProb-class),
 121
 postProc, optsol_optimizeProb-method
 (optsol_optimizeProb-class),
 121
 postProc<- (optsol_optimizeProb-class),
 121
 postProc<- , optsol_optimizeProb-method
 (optsol_optimizeProb-class),
 121
 ppProc, 96, 97, 102
 ppProc (ppProc-class), 128
 ppProc-class, 128
 preProc (optsol_optimizeProb-class), 121
 preProc, optsol_optimizeProb-method
 (optsol_optimizeProb-class),
 121
 preProc<- (optsol_optimizeProb-class),
 121
 preProc<- , optsol_optimizeProb-method
 (optsol_optimizeProb-class),
 121
 printExchange (summaryOptsol-class), 159
 printExchange, summaryOptsol-method
 (summaryOptsol-class), 159
 printMetabolite
 (printMetabolite-methods), 129
 printMetabolite, modelorg-method

- (printMetabolite-methods), 129
- printMetabolite-methods, 129
- printObjFunc, 110, 113, 115, 117, 120, 121, 123, 125
- printObjFunc (modelorg-class), 77
- printObjFunc, modelorg-method (modelorg-class), 77
- printReaction (printReaction-methods), 130
- printReaction, modelorg, ANY-method (printReaction-methods), 130
- printReaction, summaryOptsol, modelorg-method (printReaction-methods), 130
- printReaction-methods, 130
- problem (sysBiolAlg-class), 171
- problem, sysBiolAlg-method (sysBiolAlg-class), 171
- probType (optObj-class), 103
- probType, optObj-method (optObj-class), 103
- promptSysBiolAlg, 131
- rate (netFlux-class), 88
- rate, netFlux-method (netFlux-class), 88
- react (optsol_blockedReact-class), 113
- react, optsol_blockedReact-method (optsol_blockedReact-class), 113
- react, optsol_fluxVar-method (optsol_fluxVar-class), 117
- react<- (optsol_blockedReact-class), 113
- react<- , optsol_blockedReact-method (optsol_blockedReact-class), 113
- react<- , optsol_fluxVar-method (optsol_fluxVar-class), 117
- react_attr (modelorg-class), 77
- react_attr, modelorg-method (modelorg-class), 77
- react_attr<- (modelorg-class), 77
- react_attr<- , modelorg-method (modelorg-class), 77
- react_de (modelorg-class), 77
- react_de, modelorg-method (modelorg-class), 77
- react_de<- (modelorg-class), 77
- react_de<- , modelorg-method (modelorg-class), 77
- react_id, 12
- react_id (modelorg-class), 77
- react_id, modelorg-method (modelorg-class), 77
- react_id, netFlux-method (netFlux-class), 88
- react_id, optsol-method (optsol-class), 110
- react_id, reactId-method (reactId-class), 132
- react_id<- (modelorg-class), 77
- react_id<- , modelorg-method (modelorg-class), 77
- react_id<- , netFlux-method (netFlux-class), 88
- react_id<- , optsol-method (optsol-class), 110
- react_id<- , reactId-method (reactId-class), 132
- react_name (modelorg-class), 77
- react_name, modelorg-method (modelorg-class), 77
- react_name<- (modelorg-class), 77
- react_name<- , modelorg-method (modelorg-class), 77
- react_num (modelorg-class), 77
- react_num, modelorg-method (modelorg-class), 77
- react_num<- (modelorg-class), 77
- react_num<- , modelorg-method (modelorg-class), 77
- react_pos (reactId-class), 132
- react_pos, reactId-method (reactId-class), 132
- react_pos<- (reactId-class), 132
- react_pos<- , reactId-method (reactId-class), 132
- react_rev, 42
- react_rev (modelorg-class), 77
- react_rev, modelorg-method (modelorg-class), 77
- react_rev<- (modelorg-class), 77
- react_rev<- , modelorg-method (modelorg-class), 77
- react_single (modelorg-class), 77
- react_single, modelorg-method (modelorg-class), 77
- react_single<- (modelorg-class), 77
- react_single<- , modelorg-method

- (modelorg-class), 77
- reactId, 22, 25, 28, 34, 39, 47, 55, 81, 90, 92, 93, 95, 126, 130, 135, 145, 146, 156
- reactId (reactId-class), 132
- reactId-class, 132
- reactId_Exch, 30, 45, 155
- reactId_Exch (reactId_Exch-class), 134
- reactId_Exch-class, 134
- read.table, 86, 140, 141, 143
- readProb, 105, 193
- readProb (readProb-methods), 136
- readProb, optObj_clpAPI, character-method (readProb-methods), 136
- readProb, optObj_cplexAPI, character-method (readProb-methods), 136
- readProb, optObj_glpkAPI, character-method (readProb-methods), 136
- readProb, optObj_lpSolveAPI, character-method (readProb-methods), 136
- readProb-methods, 136
- readTSVmod, 5, 38, 77, 85, 137, 157
- require, 32
- resetChanges, 19
- resetChanges (resetChanges-methods), 144
- resetChanges, sysBioAlg-method (resetChanges-methods), 144
- resetChanges, sysBioAlg_room-method (resetChanges-methods), 144
- resetChanges-methods, 144
- return_codeLPSOLVE (optObj_lpSolveAPI-class), 109
- rev2irrev (modelorg_irrev-class), 86
- rev2irrev, modelorg_irrev-method (modelorg_irrev-class), 86
- rev2irrev<- (modelorg_irrev-class), 86
- rev2irrev<-, modelorg_irrev-method (modelorg_irrev-class), 86
- rmReact, 13, 145
- robAna, 124, 126, 146
- room, 100, 168
- room (sysBioAlg_room-class), 189
- rowSums, 139
- rxnGeneMat (modelorg-class), 77
- rxnGeneMat, modelorg-method (modelorg-class), 77
- rxnGeneMat<- (modelorg-class), 77
- rxnGeneMat<-, modelorg-method (modelorg-class), 77
- S (modelorg-class), 77
- S, modelorg-method (modelorg-class), 77
- S<- (modelorg-class), 77
- S<-, modelorg-method (modelorg-class), 77
- scaleProb, 105, 171, 174, 179, 182, 185, 187, 190
- scaleProb (scaleProb-methods), 147
- scaleProb, optObj_clpAPI-method (scaleProb-methods), 147
- scaleProb, optObj_cplexAPI-method (scaleProb-methods), 147
- scaleProb, optObj_glpkAPI-method (scaleProb-methods), 147
- scaleProb, optObj_lpSolveAPI-method (scaleProb-methods), 147
- scaleProb-methods, 147
- sensitivityAnalysis, 106
- sensitivityAnalysis (sensitivityAnalysis-methods), 148
- sensitivityAnalysis, optObj_cplexAPI-method (sensitivityAnalysis-methods), 148
- sensitivityAnalysis, optObj_glpkAPI-method (sensitivityAnalysis-methods), 148
- sensitivityAnalysis-methods, 148
- setColsNames (setColsNames-methods), 149
- setColsNames, optObj_clpAPI, numeric, character-method (setColsNames-methods), 149
- setColsNames, optObj_cplexAPI, numeric, character-method (setColsNames-methods), 149
- setColsNames, optObj_glpkAPI, numeric, character-method (setColsNames-methods), 149
- setColsNames, optObj_lpSolveAPI, numeric, character-method (setColsNames-methods), 149
- setColsNames-methods, 149
- setObjDir, 106
- setObjDir (setObjDir-methods), 150
- setObjDir, optObj_clpAPI, character-method (setObjDir-methods), 150
- setObjDir, optObj_clpAPI, numeric-method (setObjDir-methods), 150
- setObjDir, optObj_cplexAPI, character-method (setObjDir-methods), 150
- setObjDir, optObj_cplexAPI, integer-method (setObjDir-methods), 150
- setObjDir, optObj_cplexAPI, numeric-method

- (setObjDir-methods), 150
- setObjDir, optObj_glpkAPI, character-method (setObjDir-methods), 150
- setObjDir, optObj_glpkAPI, integer-method (setObjDir-methods), 150
- setObjDir, optObj_glpkAPI, numeric-method (setObjDir-methods), 150
- setObjDir, optObj_lpSolveAPI, character-method (setObjDir-methods), 150
- setObjDir, optObj_lpSolveAPI, numeric-method (setObjDir-methods), 150
- setObjDir-methods, 150
- setRhsZero, 106
- setRhsZero (setRhsZero-methods), 152
- setRhsZero, optObj_clpAPI-method (setRhsZero-methods), 152
- setRhsZero, optObj_cplexAPI-method (setRhsZero-methods), 152
- setRhsZero, optObj_glpkAPI-method (setRhsZero-methods), 152
- setRhsZero, optObj_lpSolveAPI-method (setRhsZero-methods), 152
- setRhsZero-methods, 152
- setRowsNames (setRowsNames-methods), 153
- setRowsNames, optObj_clpAPI, numeric, character-method (setRowsNames-methods), 153
- setRowsNames, optObj_cplexAPI, numeric, character-method (setRowsNames-methods), 153
- setRowsNames, optObj_glpkAPI, numeric, character-method (setRowsNames-methods), 153
- setRowsNames, optObj_lpSolveAPI, numeric, character-method (setRowsNames-methods), 153
- setRowsNames-methods, 153
- setSolverParm, 106
- setSolverParm (setSolverParm-methods), 154
- setSolverParm, optObj_clpAPI-method (setSolverParm-methods), 154
- setSolverParm, optObj_cplexAPI-method (setSolverParm-methods), 154
- setSolverParm, optObj_glpkAPI-method (setSolverParm-methods), 154
- setSolverParm, optObj_lpSolveAPI-method (setSolverParm-methods), 154
- setSolverParm-methods, 154
- show, checksol-method (checksol-class), 35
- show, modelorg-method (modelorg-class), 77
- show, optsol-method (optsol-class), 110
- shrinkMatrix (shrinkMatrix-methods), 155
- shrinkMatrix, modelorg-method (shrinkMatrix-methods), 155
- shrinkMatrix-methods, 155
- singletonMetabolites (singletonMetabolites-methods), 156
- singletonMetabolites, modelorg-method (singletonMetabolites-methods), 156
- singletonMetabolites-methods, 156
- sink, 101
- Snnz (modelorg-class), 77
- Snnz, modelorg-method (modelorg-class), 77
- solveLp, 106
- solveLp (solveLp-methods), 157
- solveLp, optObj_clpAPI-method (solveLp-methods), 157
- solveLp, optObj_cplexAPI-method (solveLp-methods), 157
- solveLp, optObj_glpkAPI-method (solveLp-methods), 157
- solveLp, optObj_lpSolveAPI-method (solveLp-methods), 157
- solveLp-methods, 157
- SOLVER_SYBIL_SETTINGS, 167
- solver, 106
- solver (optsol-class), 110
- solver, optObj-method (optObj-class), 103
- solver, optsol-method (optsol-class), 110
- solver<- (optsol-class), 110
- solver<-, optsol-method (optsol-class), 110
- SOLVER_CTRL_PARM (SYBIL_SETTINGS), 167
- status_code (checksol-class), 35
- status_code, checksol-method (checksol-class), 35
- status_code<- (checksol-class), 35
- status_code<-, checksol-method (checksol-class), 35
- status_meaning (checksol-class), 35
- status_meaning, checksol-method (checksol-class), 35
- status_meaning<- (checksol-class), 35
- status_meaning<-, checksol-method

- (checksol-class), 35
- status_num (checksol-class), 35
- status_num, checksol-method (checksol-class), 35
- status_num<- (checksol-class), 35
- status_num<-, checksol-method (checksol-class), 35
- stclear (sybilStack), 165
- stexists (sybilStack), 165
- stfirst (sybilStack), 165
- stinit (sybilStack), 165
- stlength (sybilStack), 165
- stlist (sybilStack), 165
- stpop (sybilStack), 165
- stpush (sybilStack), 165
- stseek (sybilStack), 165
- stshift (sybilStack), 165
- stunshift (sybilStack), 165
- subSys, 12
- subSys (modelorg-class), 77
- subSys, modelorg-method (modelorg-class), 77
- subSys<- (modelorg-class), 77
- subSys<-, modelorg-method (modelorg-class), 77
- summary, 159
- summaryOptsol, 36, 130, 131, 158, 159, 161
- summaryOptsol-class, 159
- suppressMessages, 101
- sybil (sybil-package), 5
- sybil-deprecated, 161
- sybil-package, 5
- SYBIL_SETTINGS, 17, 22, 32, 37, 40, 42, 50, 69, 90, 92, 94, 97, 102–104, 106–109, 167, 170, 171, 196
- sybilError, 128, 164
- sybilError (sybilError-class), 162
- sybilError-class, 162
- sybilLog (sybilLog-class), 163
- sybilLog-class, 163
- sybilStack, 163, 165
- sysBiolAlg, 18, 19, 21, 94–99, 101–103, 106, 131, 132, 144, 168, 169, 169, 170, 173–175, 177, 179–191
- sysBiolAlg-class, 171
- sysBiolAlg_fba, 147, 173
- sysBiolAlg_fba (sysBiolAlg_fba-class), 174
- sysBiolAlg_fba-class, 174
- sysBiolAlg_fbaEasyConstraint (sysBiolAlg_fbaEasyConstraint-class), 176
- sysBiolAlg_fbaEasyConstraint-class, 176
- sysBiolAlg_fv, 47, 173
- sysBiolAlg_fv (sysBiolAlg_fv-class), 178
- sysBiolAlg_fv-class, 178
- sysBiolAlg_lmoma, 173
- sysBiolAlg_lmoma (sysBiolAlg_lmoma-class), 181
- sysBiolAlg_lmoma-class, 181
- sysBiolAlg_moma, 173
- sysBiolAlg_moma (sysBiolAlg_moma-class), 184
- sysBiolAlg_moma-class, 184
- sysBiolAlg_mtf, 173
- sysBiolAlg_mtf (sysBiolAlg_mtf-class), 186
- sysBiolAlg_mtf-class, 186
- sysBiolAlg_mtfEasyConstraint (sysBiolAlg_fbaEasyConstraint-class), 176
- sysBiolAlg_mtfEasyConstraint-class (sysBiolAlg_fbaEasyConstraint-class), 176
- sysBiolAlg_room, 19, 95, 144, 173
- sysBiolAlg_room (sysBiolAlg_room-class), 189
- sysBiolAlg_room-class, 189
- TOLERANCE (SYBIL_SETTINGS), 167
- trellis.object, 160
- upgradeModelorg, 37, 192
- uppbnd (modelorg-class), 77
- uppbnd, modelorg-method (modelorg-class), 77
- uppbnd, reactId_Exch-method (reactId_Exch-class), 134
- uppbnd<- (modelorg-class), 77
- uppbnd<-, modelorg-method (modelorg-class), 77
- uppbnd<-, reactId_Exch-method (reactId_Exch-class), 134
- uptake (reactId_Exch-class), 134
- uptake, reactId_Exch-method (reactId_Exch-class), 134

uptake<- (reactId_Exch-class), 134
uptake<- ,reactId_Exch-method
 (reactId_Exch-class), 134
uptMet (reactId_Exch-class), 134
uptMet,reactId_Exch-method
 (reactId_Exch-class), 134
uptReact (reactId_Exch-class), 134
uptReact,reactId_Exch-method
 (reactId_Exch-class), 134
USE_NAMES (SYBIL_SETTINGS), 167

verblevel (sybilLog-class), 163
verblevel, sybilLog-method
 (sybilLog-class), 163
verblevel<- (sybilLog-class), 163
verblevel<- ,sybilLog-method
 (sybilLog-class), 163
version (modelorg-class), 77
version,modelorg-method
 (modelorg-class), 77
version<- (modelorg-class), 77
version<- ,modelorg-method
 (modelorg-class), 77

wireframe, 124
write.table, 83
writeProb, 106, 137
writeProb (writeProb-methods), 192
writeProb,optObj_clpAPI,character-method
 (writeProb-methods), 192
writeProb,optObj_cplexAPI,character-method
 (writeProb-methods), 192
writeProb,optObj_glpkAPI,character-method
 (writeProb-methods), 192
writeProb,optObj_lpSolveAPI,character-method
 (writeProb-methods), 192
writeProb-methods, 192
wrong_solver_msg (optObj-class), 103
wrong_type_msg (optObj-class), 103

ypd, 194