SimBiology® 3
Reference

MATLAB®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

SimBiology® Reference

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

Contents

Methods — Alphabetical List

4

Property Reference

5

Properties — Alphabetical List

6

Index

1

Function Reference

Modeling, Simulation, and Analysis Tools

sbioconsmoiety	Find conserved moieties in SimBiology model
sbiodesktop	Open SimBiology modeling and simulation GUI
sbioensembleplot	Show results of ensemble run using 2-D or 3-D plots
sbioensemblerun	Multiple stochastic ensemble runs of SimBiology model
sbioensemblestats	Get statistics from ensemble run data
sbiogetmodel	Get model object that generated simulation data
sbiogetnamedstate	Get state and time data from simulation results
sbiohelp	Help for SimBiology functions
sbiolasterror	SimBiology last error message
sbiolastwarning	SimBiology last warning message
sbionlinfit	Nonlinear least-squares regression using SimBiology models
sbionlmefit	Estimate nonlinear mixed effects using SimBiology models
sbioparamestim	Perform parameter estimation
sbioplot	Plot simulation results in one figure
sbioreset	Delete all model and simulation objects
sbioselect	Search for objects with specified constraints
sbiosimulate	Simulate model object
sbiosubplot	Plot simulation results in subplots

`sbiotrellis`	Plot simulation results in trellis plot
`sbioupdate`	Update SimBiology model version
`simbiology`	Open SimBiology modeling and simulation GUI

Project Opening and Saving

sbioaddtolibrary	Add to user-defined library
sbiocopylibrary	Copy library to disk
sbioloadproject	Load project from file
sbioremovefromlibrary	Remove abstract kinetic law, unit, or unit prefix from library
sbiosaveproject	Save all models in root object
sbiowhos	Show contents of project file, library file, or SimBiology root object

SBML Model Reading and Writing

sbmlexport	Export SimBiology model to SBML file
sbmlimport	Import SBML-formatted file

Object Construction

sbioabstractkineticlaw	Construct abstract kinetic law object
sbiomodel	Construct model object
sbioroot	Return SimBiology root object
sbiovariant	Construct variant object

Pharmacokinetic Modeling

`sbionlinfit`	Nonlinear least-squares regression using SimBiology models
`sbionlmefit`	Estimate nonlinear mixed effects using SimBiology models
`sbiosetdosingprofile`	Add objects to model for dosing

Units and Unit Prefixes

sbioconvertunits	Convert unit and unit value to new unit
sbioregisterunitprefix	Create user-defined unit prefix
sbioshowunitprefixes	Show unit prefixes in library
sbioshowunits	Show units in library
sbiounit	Create user-defined unit
sbiounitcalculator	Convert value between units
sbiounitprefix	Create user-defined unit prefix

Functions — Alphabetical List

sbioabstractkineticlaw

Purpose   Construct abstract kinetic law object

Syntax

```
abstkineticlawObj = sbioabstractkineticlaw('Name')
abstkineticlawObj = sbioabstractkineticlaw('Name',
    'Expression')
abstkineticlawObj = sbioabstractkineticlaw(...'PropertyName',
PropertyValue...)
```

Arguments

Name	Enter a name for the abstract kinetic law. Name must be unique in the user-defined kinetic law library. Name is referenced by *kineticlawObj*.
Expression	The mathematical expression that defines the kinetic law.

Description   A SimBiology abstract kinetic law defines a reaction rate expression, species variables, and parameter variables for a kinetic law. *abstkineticlawObj* = sbioabstractkineticlaw('*Name*') creates an abstract kinetic law object, with the name *Name* and returns it to *abstkineticlawObj*.

The *abstract kinetic law* provides a mechanism for applying a specific rate law to multiple reactions. It acts as a mapping template for the reaction rate. The abstract kinetic law defines a reaction rate expression, which is shown in the property Expression, and the species and parameter variables used in the expression. The species variables are defined in the SpeciesVariables property, and the parameter variables are defined in the ParameterVariables property of the abstract kinetic law object.

In order to use *abstkineticlawObj* when constructing a kinetic law object with the method addkineticlaw, *abstkineticlawObj* must be added to the user-defined library with the sbioaddtolibrary function. To get the abstract kinetic law objects in the user-defined library, use the command get(sbioroot, 'UserDefinedKineticLaws').

abstkineticlawObj = sbioabstractkineticlaw('*Name*','*Expression*')
constructs a SimBiology abstract kinetic law object, *abstkineticlawObj*
with the name '*Name*' and with the expression '*Expression*' and
returns it to *abstkineticlawObj*.

abstkineticlawObj = sbioabstractkineticlaw(...'*PropertyName*',
PropertyValue...) defines optional properties. The property
name/property value pairs can be in any format supported by the
function set (for example, name-value string pairs, structures, and
name-value cell array pairs).

Additional *abstkineticlawObj* properties can be viewed with the get
command. *abstkineticlawObj* properties can be modified with the
set command.

Method Summary		
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	set (any object)	Set object properties

Property Summary		
	Annotation	Store link to URL or file
	Expression	Expression to determine reaction rate equation
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	ParameterVariables	Parameters in abstract kinetic law
	Parent	Indicate parent object
	SpeciesVariables	Species in abstract kinetic law

sbioabstractkineticlaw

Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples

1 Create an abstract kinetic law.

```
abstkineticlawObj = sbioabstractkineticlaw('ex_mylaw1', '(k1*s)/(k2+k1+s)');
```

2 Assign the parameter and species variables in the expression.

```
set (abstkineticlawObj, 'SpeciesVariables', {'s'});
set (abstkineticlawObj, 'ParameterVariables', {'k1', 'k2'});
```

3 Add the new abstract kinetic law to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

sbioaddtolibrary adds the abstract kinetic law to the user-defined library. You can verify this using sbiowhos.

```
sbiowhos -kineticlaw -userdefined

SimBiology Abstract Kinetic Law Array

   Index:   Library:     Name:        Expression:
   1        UserDefined  ex_mylaw1    (k1*s)/(k2+k1+s)
```

4 Use the new abstract kinetic law when defining a reaction's kinetic law.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A + B <-> B + C');
kineticlawObj = addkineticlaw(reactionObj, 'ex_mylaw1');
```

> **Note** Remember to specify the SpeciesVariableNames and the ParameterVariableNames in kineticlawObj to fully define the ReactionRate of the reaction.

See Also   addkineticlaw, addparameter, addreaction, sbiomodel

sbioaddtolibrary

Purpose   Add to user-defined library

Syntax   sbioaddtolibrary (*abstkineticlawObj*)
                sbioaddtolibrary (*unitObj*)
                sbioaddtolibrary (*unitprefixObj*)

Arguments

abstkineticlawObj	Specify the abstract kinetic law object. The Name of the abstract kinetic law must be unique in the user-defined kinetic law library. Name is referenced by *kineticlawObj*. For more information about creating *kineticlawObj*, see sbioabstractkineticlaw.
unitObj	Specify the user-defined unit to add to the library. For more information about creating *unitObj*, see sbiounit.
unitprefixObj	Specify the user-defined unit prefix to add to the library. For more information about creating *unitprefixObj*, see sbiounitprefix.

Description   The function sbioaddtolibrary adds abstract kinetic laws, units, and unit prefixes to the user-defined library.

sbioaddtolibrary (*abstkineticlawObj*) adds the abstract kinetic law object (abstkineticlawObj) to the user-defined library.

sbioaddtolibrary (*unitObj*) adds the user-defined unit (unitObj) to the user-defined library.

sbioaddtolibrary (*unitprefixObj*) adds the user-defined unit prefix (unitprefixObj) to the user-defined library.

The sbioaddtolibrary function adds any abstract kinetic law, unit, or unit prefix to the root object's UserDefinedLibrary property. These

library components become available automatically in future MATLAB sessions.

Use the abstract kinetic law objects in the built-in and user-defined library to construct a kinetic law object with the method addkineticlaw.

To get a component of the built-in and user-defined libraries, use the commands get(sbioroot, 'BuiltInLibrary') and (get(sbioroot, 'UserDefinedLibrary')).

To remove the library component from the user-defined library, use the function sbioremovefromlibrary. You cannot remove an abstract kinetic law object being used by a kinetic law object.

Examples   This example shows how to create an abstract kinetic law and add it to the user-defined library.

1 Create an abstract kinetic law.

```
abstkineticlawObj = sbioabstractkineticlaw('ex_mylaw1', '(k1*s)/(k2+k1+s)');
```

2 Assign the parameter and species variables in the expression.

```
set (abstkineticlawObj, 'SpeciesVariables', {'s'});
set (abstkineticlawObj, 'ParameterVariables', {'k1', 'k2'});
```

3 Add the new abstract kinetic law to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

The function adds the abstract kinetic law to the user-defined library. You can verify this using sbiowhos.

```
sbiowhos -kineticlaw -userdefined

SimBiology Abstract Kinetic Law Array

Index:    Library:      Name:     Expression:
1         UserDefined   mylaw1    (k1*s)/(k2+k1+s)
```

4 Use the new abstract kinetic law when defining a reaction's kinetic law.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A + B <-> B + C');
kineticlawObj = addkineticlaw(reactionObj, 'ex_mylaw1');
```

Note Remember to specify the SpeciesVariableNames and the ParameterVariableNames in kineticlawObj to fully define the ReactionRate of the reaction.

See Also   addkineticlaw, sbioabstractkineticlaw, sbioremovefromlibrary, sbioroot, sbiounit, sbiounitprefix

sbioconsmoiety

Purpose		Find conserved moieties in SimBiology model
Syntax		[*G, Sp*] = sbioconsmoiety(*modelObj*)
		[*G, Sp*] = sbioconsmoiety(*modelObj*, *alg*)
		H = sbioconsmoiety(*modelObj*, *alg*,'p')
		H = sbioconsmoiety(*modelObj*, *alg*,'p', *FormatArg*)
		[*SI, SD, L0, NR, ND*] = sbioconsmoiety(*modelObj*,'link')

Arguments

G	An m-by-n matrix, where m is the number of conserved quantities found and n is the number of species in the model. Each row of *G* specifies a linear combination of species whose rate of change over time is zero.
Sp	Cell array of species names that labels the columns of *G*. If the species are in multiple compartments, species names are qualified with the compartment name in the form compartmentName.speciesName. For example, nucleus.DNA, cytoplasm.mRNA.
modelObj	Model object to be evaluated for conserved moieties.
alg	Specify algorithm to use during evaluation of conserved moieties. Valid values are 'qr', 'rreduce', or 'semipos'.
H	Cell array of strings containing the conserved moieties.
p	Prints the output to a cell array of strings.
FormatArg	Specifies formatting for the output *H*. *FormatArg* should either be a C-style format string, or a positive integer specifying the maximum number of digits of precision used.
SI	Cell array containing the names of independent species in the model.

SD	Cell array containing the names of dependent species in the model.
LO	Link matrix relating SI and SD. The link matrix *LO* satisfies ND = LO*NR. For the 'link' functionality, species with their BoundaryCondition or ConstantAmount properties set to true are treated as having stoichiometry of zero in all reactions.
NR	Reduced stoichiometry matrices containing one row for each independent species. The concatenated matrix [NR;ND] is a row-permuted version of the full stoichiometry matrix of *modelObj*.
ND	Reduced stoichiometry matrices containing one row for each dependent species. The concatenated matrix [*NR;ND*] is a row-permuted version of the full stoichiometry matrix of *modelObj*.

Description   [*G, Sp*] = sbioconsmoiety(*modelObj*) calculates a complete set of linear conservation relations for the species in the SimBiology model object *modelObj*.

sbioconsmoiety computes conservation relations by analyzing the structure of the model object's stoichiometry matrix. Thus, sbioconsmoiety does not include species that are governed by algebraic or rate rules.

[*G, Sp*] = sbioconsmoiety(*modelObj*, *alg*) provides an algorithm specification. For *alg*, specify 'qr', 'rreduce', or 'semipos'.

- When you specify 'qr', sbioconsmoiety uses an algorithm based on QR factorization. From a numerical standpoint, this is the most efficient and reliable approach.

- When you specify 'rreduce', sbioconsmoiety uses an algorithm based on row reduction, which yields better numbers for smaller models. This is the default.

- When you specify `'semipos'`, sbioconsmoiety returns conservation relations in which all the coefficients are greater than or equal to 0, permitting a more transparent interpretation in terms of physical quantities.

For larger models, the QR-based method is recommended. For smaller models, row reduction or the semipositive algorithm may be preferable. For row reduction and QR factorization, the number of conservation relations returned equals the row rank degeneracy of the model object's stoichiometry matrix. The semipositive algorithm may return a different number of relations. Mathematically speaking, this algorithm returns a generating set of vectors for the space of semipositive conservation relations.

H = sbioconsmoiety(*modelObj*, *alg*,'p') returns a cell array of strings *H* containing the conserved quantities in *modelObj*.

H = sbioconsmoiety(*modelObj*, *alg*,'p', *FormatArg*) specifies formatting for the output H. *FormatArg* should either be a C-style format string, or a positive integer specifying the maximum number of digits of precision used.

[*SI, SD, L0, NR, ND*] = sbioconsmoiety(*modelObj*,'link') uses a QR-based algorithm to compute information relevant to the dimensional reduction, via conservation relations, of the reaction network in *modelObj*.

Examples

Example 1

This example shows conserved moieties in a cycle.

1 Create a model with a cycle. For convenience use arbitrary reaction rates, as this will not affect the result.

```
modelObj = sbiomodel('cycle');
modelObj.addreaction('a -> b','ReactionRate','1');
modelObj.addreaction('b -> c','ReactionRate','b');
modelObj.addreaction('c -> a','ReactionRate','2*c');
```

2 Look for conserved moieties.

```
[g sp] = sbioconsmoiety(modelObj)

g =

    1    1    1


sp =

    'a'
    'b'
    'c'
```

Example 2

Explore semipositive conservation relations in the `oscillator` model.

```
modelObj = sbmlimport('oscillator');
sbioconsmoiety(modelObj,'semipos','p')

ans =

    'pol + pol_OpA + pol_OpB + pol_OpC'
    'OpB + pol_OpB + pA_OpB1 + pA_OpB_pA + pA_OpB2'
    'OpA + pol_OpA + pC_OpA1 + pC_OpA2 + pC_OpA_pC'
    'OpC + pol_OpC + pB_OpC1 + pB_OpC2 + pB_OpC_pB'
```

See Also   "Moiety Conservation" in the SimBiology User's Guide documentation

SimBiology method `getstoichmatrix`

Purpose   Convert unit and unit value to new unit

Syntax   sbioconvertunits(*Obj*, '*unit*')

Description   sbioconvertunits(*Obj*, '*unit*') converts the current *Units property
                   on SimBiology object, *Obj* to the unit, *unit*. This function configures
                   the *Units property to *unit* and updates the corresponding value
                   property. For example, sbioconverunits on a speciesObj updates the
                   InitialAmount property value and the InitialAmountUnits property
                   value.

                   Obj can be an array of SimBiology objects. *Obj* must be a SimBiology
                   object that contains a unit property. The SimBiology objects that
                   contain a unit property are compartment, parameter, and species
                   objects. For example, if *Obj* is a species object with InitialAmount
                   configured to 1 and InitialAmountUnits configured to mole, after the
                   call to sbioconvertunits with *unit* specified as molecule, speciesObj
                   InitialAmount is 6.0221e23 and InitialAmountUnits is molecule.

Examples   Convert the units of the initial amount of glucose from molecule to
                   mole.

                   1 Create the species 'glucose' and assign an initial amount of 23
                   molecule.

                   At the command prompt, type:

```
modelObj = sbiomodel('cell');
compObj = addcompartment(modelObj, 'C');
speciesObj = addspecies (compObj, 'glucose', 23, 'InitialAmountUnits', 'molecule')


SimBiology Species Array

   Index:    Compartment:    Name:     InitialAmount:    InitialAmountUnits:
   1         C               glucose   23                molecule
```

2 Convert the `InitialAmountUnits` of glucose from `molecule` to `mole`.

```
sbioconvertunits (speciesObj, 'mole')
```

3 Verify the conversion of units and `InitialAmount` value.

Units are converted from molecule to mole.

```
get (speciesObj, 'InitialAmountUnits')
```

```
ans =
```

```
mole
```

The `InitialAmount` value is changed.

```
get (speciesObj, 'InitialAmount')
```

```
ans =
```

```
  3.8192e-023
```

See Also   sbioshowunits

sbiocopylibrary

Purpose	Copy library to disk

Syntax

```
sbiocopylibrary ('kineticlaw','LibraryFileName')
sbiocopylibrary ('unit','LibraryFileName')
```

Description sbiocopylibrary copies all user-defined abstract kinetic laws to a file. sbiocopylibrary ('kineticlaw','*LibraryFileName*') copies all user-defined abstract kinetic laws to the file LibraryFileName.sbklib and places the copied file in the current directory.

sbiocopylibrary ('unit','*LibraryFileName*') copies all user-defined units and unit prefixes to the file LibraryFileName.sbulib.

To get the abstract kinetic law objects in the built-in and user-defined libraries, use the commands get(sbioroot, 'BuiltInKineticLaws') and get(sbioroot, 'UserDefinedKineticLaws'). To add an abstract kinetic law to the user-defined library, use the method sbioaddtolibrary.

To add a unit to the user-defined library, use the sbioregisterunit function. To add a unit prefix to the user-defined library, use the sbioregisterunitprefix function.

Examples Create an abstract kinetic law, add it to the user-defined library, and then copy the user-defined kinetic law library to a .sbklib file.

1 Create an abstract kinetic law.

```
abstkineticlawObj = sbioabstractkineticlaw('mylaw1', '(k1*s)/(k2+k1+s)');
```

2 Add the new abstract kinetic law to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

sbioaddtolibrary adds the abstract kinetic law to the user-defined library. You can verify this using sbiowhos.

```
sbiowhos -kineticlaw -userdefined
```

```
SimBiology Abstract Kinetic Law Array

Index:     Library:       Name:        Expression:
  1         UserDefined    mylaw1     (k1*s)/(k2+k1+s)
```

3 Copy the user-defined kinetic law library.

```
sbiocopylibrary ('kineticlaw','myLibFile')
```

4 Verify with sbiowhos.

```
sbiowhos -kineticlaw myLibFile
```

See Also   sbioaddtolibrary, sbioabstractkineticlaw, sbioregisterunit, sbioregisterunitprefix, sbioremovefromlibrary

sbiodesktop

Purpose   Open SimBiology modeling and simulation GUI

Syntax
```
sbiodesktop
sbiodesktop(modelObj)
```

Arguments

modelObj	Model object or an array of model objects. Enter the variable name for a top-level SimBiology model object. If you enter an array of model objects, the SimBiology desktop opens with each model object in a separate model session.

Description   sbiodesktop opens the SimBiology desktop, which lets you do the following:

- Build a SimBiology model by representing reaction pathways and entering kinetic data for the reactions.

- Import or export SimBiology models to and from the MATLAB workspace or from a Systems Biology Markup Language (SBML) file.

- Modify an existing SimBiology model.

- Simulate a SimBiology model through individual or ensemble runs.

- View results from the simulation.

- Perform analysis tasks such as sensitivity analysis, parameter and species scans, and calculate conserved moieties.

- Create and/or modify user-defined units and unit prefixes.

- Create and/or modify user-defined kinetic laws.

sbiodesktop(*modelObj*) opens the SimBiology desktop with a top-level SimBiology model object (*modelObj*). If there is a project open in the SimBiology desktop, this command adds the model (*modelObj*) to the project. A top-level SimBiology model object has its property Parent set to the SimBiology root object. Thus, querying sbioroot at the command

sbiodesktop

line shows you all models in the MATLAB workspace, including the models available in the desktop. Any changes you make to the model at the command line are reflected in the desktop because both are pointing to the same model object in the Root object.

Note The sbioreset command removes all models from the root object and therefore this command also removes all models from the SimBiology desktop.

Examples   Create a SimBiology model in the MATLAB workspace, and then open the GUI with the model.

```
modelObj = sbiomodel('cell');
sbiodesktop(modelObj)
```

See Also   sbioroot, simbiology

Purpose   Show results of ensemble run using 2-D or 3-D plots

Syntax
```
sbioensembleplot(simdataObj)
sbioensembleplot(simdataObj, Names)
sbioensembleplot(simdataObj, Names, Time)
FH = sbioensembleplot(simdataObj, Names)
FH = sbioensembleplot(simdataObj, Names, Time)
```

Arguments

simdataObj	An object that contains simulation data. You can generate a *simdataObj* object using the function sbioensemblerun. All elements of *simdataObj* must contain data for the same states in the same model.
Names	Either a string or a cell array of strings. *Names* may include qualified names such as '*CompartmentName.SpeciesName*' or '*ReactionName.ParameterName*' to resolve ambiguities. Specifying {} for *Names* plots data for all states contained in *simdataObj*.
Time	A numeric scalar value. If the specified *Time* is not an element of the time vectors in *simdataObj*, then the function resamples *simdataObj* as necessary using linear interpolation.
FH	Array of handles to figure windows.

Description   sbioensembleplot(*simdataObj*) shows a 3-D shaded plot of time-varying distribution of all logged states in the SimData array *simdataObj*. The sbioensemblerun function plots an approximate distribution created by fitting a normal distribution to the data at every time step.

sbioensembleplot(*simdataObj*, *Names*) plots the distribution for the data specified by *Names*.

sbioensembleplot

sbioensembleplot(*simdataObj*, *Names*, *Time*) plots a 2-D histogram of the actual data of the ensemble distribution of the states specified by *Names* at the particular time point *Time*.

FH = sbioensembleplot(*simdataObj*, *Names*) returns an array of handles *FH*, to the figure window for the 3-D distribution plot.

FH = sbioensembleplot(*simdataObj*, *Names*, *Time*) returns an array of handles *FH*, to the figure window for the 2-D histograms.

Examples   This example shows how to plot data from an ensemble run without interpolation.

1 The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject('radiodecay.sbproj','m1');
```

2 Change the solver of the active configuration set to be ssa. Also, adjust the LogDecimation property on the SolverOptions property of the configuration set to reduce the size of the data generated.

```
cs = getconfigset(m1, 'active');
set(cs, 'SolverType', 'ssa');
so = get(cs, 'SolverOptions');
set(so, 'LogDecimation', 10);
```

3 Perform an ensemble of 20 runs with no interpolation.

```
simdataObj = sbioensemblerun(m1, 20);
```

4 Create a 2-D distribution plot of the species 'z' at time = 1.0.

```
FH1 = sbioensembleplot(simdataObj, 'z', 1.0);
```

5 Create a 3-D shaded plot of both species.

```
FH2 = sbioensembleplot(simdataObj, {'x','z'});
```

See Also   sbioensemblerun, sbioensemblestats, sbiomodel

Purpose   Multiple stochastic ensemble runs of SimBiology model

Syntax   simdataObj = sbioensemblerun(*modelObj*, *Numruns*)
               simdataObj = sbioensemblerun(*modelObj*, *Numruns*,
                 Interpolation)
               simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*)
               simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*,
                 Interpolation)
               simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *variantObj*)
               simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *variantObj*,
                 Interpolation)
               simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*,
                 variantObj)
               simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*,
                 variantObj, *Interpolation*)

Arguments

simdataObj	An object that contains simulation data generated by sbioensemblerun. All elements of *simdataObj* must contain data for the same states in the same model.
modelObj	Model object to be simulated.
Numruns	Integer scalar representing the number of stochastic runs to make.
Interpolation	String variable denoting the interpolation scheme to be used if data should be interpolated to get a consistent time vector. Valid values are 'linear' (linear interpolation), 'zoh' (zero-order hold), or 'off' (no interpolation). Default is 'off'. If interpolation is on, the data is interpolated to match the time vector with the smallest simulation stop time.

sbioensemblerun

configsetObj	Specify the configuration set object to use in the ensemble simulation. For more information about configuration sets, see `Configset object`.
variantObj	Specify the variant object to apply to the model during the ensemble simulation. For more information about variant objects, see `Variant object`.

Description   simdataObj = sbioensemblerun(*modelObj*, *Numruns*) performs a stochastic ensemble run of the SimBiology model object (*modelObj*), and returns the results in the SimData object (*simdataObj*). The active configset and the active variants are used during simulation and are saved in the output, SimData object (*simdataObj*).

sbioensemblerun uses the settings in the active configset on the model object (*modelObj*) to perform the repeated simulation runs. The SolverType property of the active configset must be set to one of the stochastic solvers: 'ssa', 'expltau', or 'impltau'. sbioensemblerun generates an error if the SolverType property is set to any of the deterministic (ODE) solvers.

simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *Interpolation*) performs a stochastic ensemble run of a model object (*modelObj*), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (*Interpolation*).

simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*) performs an ensemble run of a model object (*modelObj*), using the specified configuration set (*configsetObj*).

simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*, *Interpolation*) performs an ensemble run of a model object (*modelObj*), using the specified configuration set (*configsetObj*), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (*Interpolation*).

simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *variantObj*) performs an ensemble run of a model object (*modelObj*), using the variant object or array of variant objects (*variantObj*).

simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *variantObj*, *Interpolation*) performs an ensemble run of a model object (*modelObj*), using the variant object or array of variant objects (*variantObj*), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (*Interpolation*).

simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*, *variantObj*) performs an ensemble run of a model object (*modelObj*), using the configuration set (*configsetObj*), and the variant object or array of variant objects (*variantObj*). If the configuration set object (*configsetObj*) is empty, the active configset on the model is used for simulation. If the variant object (*variantObj*) is empty, then no variant (not even the active variants in the model) is used for the simulation.

simdataObj = sbioensemblerun(*modelObj*, *Numruns*, *configsetObj*, *variantObj*, *Interpolation*) performs an ensemble run of a model object (*modelObj*), using the configuration set (*configsetObj*), and the variant object or array of variant objects (*variantObj*), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (*Interpolation*).

Examples This example shows how to perform an ensemble run and generate a 2-D distribution plot.

1 The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject('radiodecay.sbproj','m1');
```

2 Change the solver of the active configset to be ssa. Also, adjust the LogDecimation property on the SolverOptions property of the configuration set.

```
cs = getconfigset(m1, 'active');
set(cs, 'SolverType', 'ssa');
```

```
so = get(cs, 'SolverOptions');
set(so, 'LogDecimation', 10);
```

> **Tip** The LogDecimation property lets you define how often the
> simulation data is recorded as output. If your model has high
> concentrations or amounts of species, or a long simulation time (for
> example, 600s), you can record simulation data less often to manage
> the amount of data generated. Be aware that by doing so you might
> miss some transitions if your model is very dynamic. Try setting
> LogDecimation to 10 or more.

3 Perform an ensemble of 20 runs with linear interpolation to get
a consistent time vector.

```
simdata = sbioensemblerun(m1, 20, 'linear');
```

4 Create a 2-D distribution plot of the species 'z' at a time = 1.0.

```
FH = sbioensembleplot(simdata, 'z', 1.0);
```

See Also   addconfigset, getconfigset, sbioensemblestats,
sbioensembleplot, setactiveconfigset, SimData object

Purpose	Get statistics from ensemble run data

Syntax

```
[t,m] = sbioensemblestats(simDataObj)
[t,m,v] = sbioensemblestats(simDataObj)
[t,m,v,n] = sbioensemblestats(simDataObj)
```

Arguments

t	Vector of doubles that holds the common time vector after interpolation.
m	Matrix of mean values from the ensemble data. The number of rows in *m* is the length of the common time vector *t* after interpolation and the number of columns is equal to the number of species. The species order corresponding to the columns of *m* can be obtained from any of the SimData objects in *simDataObj* using sbiogetnamedstate.
simDataObj	A cell array of SimData objects, where each SimData object holds data for a separate simulation run. All elements of *simDataObj* must contain data for the same states in the same model. When the time vectors of the elements of *simDataObj* are not identical, *simDataObj* is first resampled onto a common time vector (see *interpolation* below).
v	Matrix of variance obtained from the ensemble data. *v* has the same dimensions as *m*.
n	Cell array of strings that holds names whose mean and variance are returned in *m* and *v*, respectively. The number of elements in *n* is the same as the number of columns of *m* and *v*. The order of names in *n* corresponds to the order of columns of *m* and *v*.

sbioensemblestats

names	Either a string or a cell array of strings. *names* may include qualified names such as `'CompartmentName.SpeciesName'` or `'ReactionName.ParameterName'` to resolve ambiguities. If you specify empty {} for *names*, sbioensemblestats returns statistics on all time courses contained in *simDataObj*.
interpolation	String variable denoting the interpolation method to be used if data is to be interpolated to get a consistent time vector. See `resample` for a list of interpolation methods. Default is `'off'`. If interpolation is on, the data is interpolated to match the time vector with the smallest simulation stop time.

Description   [*t*,*m*] = sbioensemblestats(*simDataObj*) computes the time-dependent ensemble mean *m* of the ensemble data *simDataObj* obtained by running sbioensemblerun.

[*t*,*m*,*v*] = sbioensemblestats(*simDataObj*) computes the time-dependent ensemble mean *m* and variance *v* for the ensemble run data *simDataObj*.

[*t*,*m*,*v*,*n*] = sbioensemblestats(*simDataObj*) computes the time-dependent ensemble mean *m* and variance *v* for the ensemble run data *simDataObj*. Each column of *m* or *v* describes the ensemble mean or variance of some state as a function of time.

Examples   The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

1 Load a SimBiology model m1 from a SimBiology project file.

```
sbioloadproject('radiodecay.sbproj','m1');
```

2 Change the solver of the active configuration set to be `ssa`. Also, adjust the `LogDecimation` property on the `SolverOptions` property of the configuration set.

```
cs = getconfigset(m1, 'active');
set(cs, 'SolverType', 'ssa');
so = get(cs, 'SolverOptions');
set(so, 'LogDecimation', 10);
```

3 Perform an ensemble of 20 runs with no interpolation.

```
simDataObj = sbioensemblerun(m1, 20);
```

4 Get ensemble statistics for all species using the default interpolation method.

```
[T,M,V] = sbioensemblestats(simDataObj);
```

5 Get ensemble statistics for a specific species using the default interpolation scheme.

```
[T2,M2,V2] = sbioensemblestats(simDataObj, {'z'});
```

See Also   sbioensemblerun, sbioensembleplot, sbiogetnamedstate, sbiomodel

sbioevent

Purpose	Construct event object

> **Note** sbioevent has been removed and produces an error. Use
> addevent instead.

Syntax

```
eventObj = sbioevent(TriggerValue, EventFcnsValue)
eventObj = sbioevent(...'PropertyName', PropertyValue...)
```

Arguments

TriggerValue	Required property to specify a trigger condition. Must be a MATLAB expression that evaluates to a logical value.
EventFcnsValue	A string or a cell array of strings, each of which specifies an assignment of the form '*objectname* = *expression*', where *objectname* is the name of a valid SimBiology object.
PropertyName	Property name for an event object from "Property Summary" on page 2-29.
PropertyValue	Property value. For more information on property values, see the property reference for each property listed in "Property Summary" on page 2-29.

Description eventObj = sbioevent(*TriggerValue*, *EventFcnsValue*) creates a
SimBiology event object, assigns a value (*TriggerValue*) for the
property Trigger, assigns a value (*EventFcnsValue*) to the property
EventFcns, and returns the object (eventObj).

During model simulation, an event is triggered and its EventFcns are
evaluated when the Trigger transitions from false to true. In order for
an event to be used in a simulation, the event object must be added to a
SimBiology model object with the copyobj function.

The preferred way to work with events is to add an event to a SimBiology model with the addevent function.

For details on how events are handled during a simulation, see "Changing Model Component Values Using Events" in the SimBiology User's Guide documentation.

eventObj = sbioevent(...*'PropertyName'*, *PropertyValue*...) defines optional properties. The property name/property value pairs can be any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

Method Summary	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	set (any object)	Set object properties

Property Summary	Active	Indicate object in use during simulation
	Annotation	Store link to URL or file
	EventFcns	Event expression
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	Parent	Indicate parent object
	Tag	Specify label for SimBiology object

Trigger	Event trigger
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples **1** Create an event object.

```
eventObj = sbioevent('time>= 5', 'OpC = 200');
```

2 Get a list of properties for the event object.

```
get(eventObj)
```

MATLAB displays a list of event properties.

```
     Active: 1
 Annotation: ''
  EventFcns: {'OpC = 200'}
       Name: ''
      Notes: ''
     Parent: [1x1 SimBiology.Model]
        Tag: ''
    Trigger: 'time >= 5'
       Type: 'event'
   UserData: []
```

See Also addevent, copyobj, Event object

Purpose	Get model object that generated simulation data

Syntax	*modelObj* = sbiogetmodel(*simDataObj*)

Arguments

simDataObj	SimData object returned by the function sbiosimulate or by sbioensemblerun.
modelObj	Model object associated with the SimData object.

Description *modelObj* = sbiogetmodel(*simDataObj*) returns the SimBiology model (*modelObj*) associated with the results from a simulation run (*simDataObj*). You can use this function to find the model object associated with the specified SimData object when you load a project with several model objects and SimData objects.

If the SimBiology model used to generate the SimData object (*simDataObj*) is not currently loaded, *modelObj* is empty.

Example Retrieve the model object that generated the SimData object.

1 Create a model object, simulate, and then return the results as a SimData object.

```
modelObj = sbmlimport('oscillator');
simDataObj = sbiosimulate(modelObj);
```

2 Get the model that generated the simulation results.

```
modelObj2 = sbiogetmodel(simDataObj)
SimBiology Model - Oscillator

Model Components:
  Models:          0
  Parameters:      0
  Reactions:       42
```

```
           Rules:              0
           Species:           23
```

3 Check that the two models are the same.

```
modelObj == modelObj2
ans =
        1
```

See Also   sbiosimulate

Purpose   Get state and time data from simulation results

Note sbiogetnamedstate produces a warning and will be removed in a future version. Use selectbyname instead.

Syntax

```
[t,x] = sbiogetnamedstate(simDataObj)
[t,x] = sbiogetnamedstate(simDataObj,'Name')
[t,x,Name] = sbiogetnamedstate(...)
```

Description   sbiogetnamedstate returns state and time data from simulation results. [t,x] = sbiogetnamedstate(simDataObj) returns the time and state data associated with the simulation results (simDataObj) and returns to t and x respectively. simDataObj is a SimData object returned by the sbiosimulate function.

- t is an n-by-1 vector of time samples labeling the rows of x.

- x is an n-by-m matrix, where n is the number of times the reactions fired and m is the number of states logged during simulation. Each column of x defines the variation in the quantity of a species over time.

[t,x] = sbiogetnamedstate(simDataObj,'Name') returns the state data associated with the name Name from the SimData object (smDataObj), and returns it to x. Name can be a cell array names. If a name, Name, does not exist, you see a warning.

[t,x,Name] = sbiogetnamedstate(...) returns the names associated with each column of x to Name.

See Also   sbiosimulate

sbiogetsensmatrix

Purpose   3-D sensitivity matrix from simulation results

Note sbiogetsensmatrix produces a warning and will be removed in a future version. Use getsensmatrix instead.

Syntax   [*T,R,States,Inpfacs*] = sbiogetsensmatrix(*simDataObj*)
                     [*T,R,Outputs,Inpfacs*] = sbiogetsensmatrix(*imDataobj*,
                       OutNames, *InpFacNames*)

Arguments

T	Column vector of length m specifying time points for the sensitivity data in *R*.
R	An m-by-n-by-p array of sensitivity data with times, outputs, and input factors labeling its first, second, and third dimensions respectively.
Outputs	Contains names of the species states that label the second dimension of *R*. R(:,i,j) is the time course for the sensitivity of state *Outputs{i}* to the input factor *Inpfacs{j}*. When *simdataObj* contains more than one element, the output arguments are cell arrays in which each cell contains data for the corresponding element of *simdataObj*.
Inpfacs	Contains names of the input factors that label the third dimension of *R*. R(:,i,j) is the time course for the sensitivity of states *Outputs{i}* to the input factor *Inpfacs{j}*.
simDataObj	SimData object returned by sbiosimulate. Contains sensitivity data when sensitivity analysis is enabled.

OutNames	Specify outputs to get sensitivity data from *simDataObj*. Can be an empty array, or a single name, or a cell array of names. When an empty array is specified, returns the sensitivity data on all species states contained in *simDataObj*.
InpFacNames	Specify input factors to get sensitivity data from *simDataObj*. Can be an empty array, or a single name, or a cell array of names. When an empty array is specified, returns the sensitivity data for all input factors contained in *simDataObj*.

Description   [*T,R,States,Inpfacs*] = sbiogetsensmatrix(*simDataObj*) gets time and sensitivity data from the SimData object *simDataObj* generated by simulating a SimBiology model object using sbiosimulate. sbiogetsensmatrix can only return sensitivity data that is contained in *simDataObj*.

The sensitivity data that is logged in *simDataObj* is set at simulation time by the active configuration set that is used during the simulation. Note that the sensitivity data *R* returned by sbiogetsensmatrix may be normalized, as specified at simulation time.

[*T,R,Outputs,Inpfacs*] = sbiogetsensmatrix(*imDataobj*, *OutNames*, *InpFacNames*) gets sensitivity data for the outputs specified by *OutNames* and the input factors specified by *InpFacNames*.

See Also   getsensmatrix , sbiogetnamedstate, sbiohelp, sbiosimulate

sbiohelp

Purpose	Help for SimBiology functions
Syntax	sbiohelp('*FunctionName*') h = sbiohelp ('*FunctionName*')
Description	sbiohelp('*FunctionName*') displays information for a SimBiology function (*FunctionName*).

h = sbiohelp ('*FunctionName*') returns the help for the SimBiology function *FunctionName* to h.

You can get general information on the SimBiology software by specifying *FunctionName* as 'sbio'. General information about a SimBiology object can be returned by specifying *FunctionName* as one of the following: 'AbstractKineticLaw', 'Compartment', 'CompileOptions', 'Configset', 'Event', 'ExplicitTauSolverOptions', 'ImplicitTauSolverOptions', 'KineticLaw', 'ODESolverOptions', 'Model', 'Parameter', 'PKData', 'PKCompartment', 'PKModelDeign', 'PKModelMap', 'Reaction', 'Root', 'Rule', 'RuntimeOptions', 'SensitivityAnalysisOptions', 'SimData', 'Species', 'SSASolverOptions', 'Library', 'Unit', 'UnitPrefix', or 'Variant'.

Examples
```
sbiohelp('addreaction')
sbiohelp addreaction
sbiohelp reaction
sbiohelp('sbioshowunits')
```

See Also MATLAB function help

Purpose	SimBiology last error message

Syntax
```
sbiolasterror
diagstruct = sbiolasterror
sbiolasterror([])
sbiolasterror(diagstruct)
```

Arguments

diagstruct	The diagnostic structure holding Type, Message ID, and Message for the errors.

Description sbiolasterror or *diagstruct* = sbiolasterror return a SimBiology diagnostic structure array containing the last error(s) generated by the software. The fields of the diagnostic structure are:

Type	'error'
MessageID	The message ID for the error (for example, 'SimBiology:ConfigSetNameClash')
Message	The error message

sbiolasterror([]) resets the SimBiology last error so that it will return an empty array until the next SimBiology error is encountered.

sbiolasterror(*diagstruct*) will set the SimBiology last error(s) to those specified in the diagnostic structure (*diagstruct*).

Examples This example shows how to use verify and sbiolasterror.

1 Import a model.

```
a = sbmlimport('radiodecay.xml')

SimBiology Model - RadioactiveDecay

 Model Components:
   Models:          O
```

```
Parameters:          1
Reactions:           1
Rules:               0
Species:             2
```

2 Change the `ReactionRate` of a reaction to make the model invalid.

```
a.reactions(1).reactionrate = 'x*y'

SimBiology Model - RadioactiveDecay

 Model Components:
   Models:              0
   Parameters:          1
   Reactions:           1
   Rules:               0
   Species:             2
```

3 Use the function `verify` to validate the model.

```
 a.verify

??? Error using ==> simbio\private\odebuilder>buildPatternSubStrings
The object y does not resolve on reaction with expression 'x*y'.

Error in ==> sbiogate at 22
 feval(varargin{:});

??? --> Error reported from Expression Validation :
The object 'y' in reaction 'Reaction1' does not resolve to any in-scope species
                                                      or parameters.
--> Error reported from Dimensional Analysis :
Could not resolve species, parameter or model object 'y' during dimensional analysis.
--> Error reported from ODE Compilation:
Error using ==> simbio\private\odebuilder>buildPatternSubStrings
The object y does not resolve on reaction with expression 'x*y'.
```

4 Retrieve the error diagnostic `struct`.

```
p = sbiolasterror

p =

1x3 struct array with fields:
  Type
  MessageID
  Message
```

5 Display the first error `ID` and `Message`.

```
p(1)

ans =

        Type: 'Error'
   MessageID: 'SimBiology:ReactionObjectDoesNotResolve'
     Message: 'The object 'y' in reaction 'Reaction1' does not
              resolve to any in-scope species or parameters.'
```

6 Reset the `sbiolasterror`.

```
sbiolasterror([])


ans =

     []
```

7 Set `sbiolasterror` to the diagnostic `struct`.

```
sbiolasterror(p)


ans =
```

sbiolasterror

```
1x3 struct array with fields:
    Type
    MessageID
    Message
```

See Also   sbiolastwarning, verify

Purpose   SimBiology last warning message

Syntax
```
sbiolastwarning
diagstruct = sbiolastwarning
sbiolastwarning([])
sbiolastwarning(diagstruct)
```

Arguments

diagstruct	The diagnostic structure holding Type, Message ID, and Message for the warnings.

Description   sbiolastwarning or *diagstruct* = sbiolastwarning return a SimBiology diagnostic structure array containing the last warnings generated by the software. The fields of the diagnostic structure are:

Type	'warning'
MessageID	The message ID for the warning (for example, 'SimBiology:DANotPerformedReactionRate')
Message	The warning message

sbiolastwarning([]) resets the SimBiology last warning so that it will return an empty array until the next SimBiology warning is encountered.

sbiolastwarning(*diagstruct*) will set the SimBiology last warnings to those specified in the diagnostic structure (*diagstruct*).

See Also   sbiolasterror, verify

sbioloadproject

Purpose   Load project from file

Syntax   sbioloadproject('*projFilename*')
                 sbioloadproject ('*projFilename*','*variableName*')
                 sbioloadproject *projFilename variableName1 variableName2...*
                 s = sbioloadproject (...)

Description sbioloadproject('*projFilename*') loads a SimBiology project
                 from a project file (*projFilename*). If no extension is specified,
                 sbioloadproject assumes a default extension of .sbproj.
                 Alternatively, the command syntax is sbioloadproject *projFilename*.

                 sbioloadproject ('*projFilename*','*variableName*') loads only the
                 variable *variableName* from the project file.

                 sbioloadproject *projFilename variableName1 variableName2...* loads
                 the specified variables from the project.

                 s = sbioloadproject (...) returns the contents of *projFilename* in
                 a variable s. s is a struct containing fields matching the variables
                 retrieved from the SimBiology project.

                 You can display the contents of the project file using the sbiowhos
                 command.

See Also   sbioaddtolibrary, sbioremovefromlibrary, sbiosaveproject,
                 sbiowhos

Purpose   Construct model object

Syntax   *modelObj* = sbiomodel(*'NameValue'*)
modelObj = sbiomodel(...*'PropertyName'*, *PropertyValue*...)

Arguments

NameValue	Required property to specify a unique name for a model object. Enter a character string.
PropertyName	Property name for a Model object from "Property Summary" on page 2-45.
PropertyValue	Property value. Valid value for the specified property.

Description   *modelObj* = sbiomodel(*'NameValue'*) creates a model object and returns the model object (*modelObj*). In the model object, this method assigns a value (*NameValue*) to the property Name.

modelObj = sbiomodel(...*'PropertyName'*, *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

Simulate *modelObj* with the function sbiosimulate.

Add objects to a model object using the methods addkineticlaw, addmodel, addparameter, addreaction, addrule, and addspecies.

All SimBiology model objects can be retrieved from the SimBiology root object. A SimBiology model object has its Parent property set to the SimBiology root object.

Method Summary

addcompartment (model, compartment)	Create compartment object
addconfigset (model)	Create configuration set object and add to model object

sbiomodel

addevent (model)	Add event object to model object
addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
addreaction (model)	Create reaction object and add to model object
addrule (model)	Create rule object and add to model object
addvariant (model)	Add variant to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getadjacencymatrix (model)	Get adjacency matrix from model object
getconfigset (model)	Get configuration set object from model object
getstoichmatrix (model)	Get stoichiometry matrix from model object
getvariant (model)	Get variant from model
removeconfigset (model)	Remove configuration set from model
removevariant (model)	Remove variant from model
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties

setactiveconfigset (model)	Set active configuration set for model object
verify (model, variant)	Validate and verify SimBiology model

Property Summary

Annotation	Store link to URL or file
Compartments	Array of compartments in model or compartment
Events	Contain all event objects
Models	Contain all model objects
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parameters	Array of parameter objects
Parent	Indicate parent object
Reactions	Array of reaction objects
Rules	Array of rules in model object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples

1 Create a SimBiology model object.

```
modelObj = sbiomodel('cell', 'Tag', 'mymodel');
```

2 List all modelObj properties and the current values.

```
get(modelObj)
```

MATLAB returns:

```
   Annotation: ''
       Models: [0x1 double]
         Name: 'cell'
        Notes: ''
   Parameters: [0x1 double]
       Parent: [1x1 SimBiology.Root]
      Species: [0x1 double]
    Reactions: [0x1 double]
        Rules: [0x1 double]
          Tag: 'mymodel'
         Type: 'sbiomodel'
     UserData: []
```

3 Display a summary of modelObj contents.

```
modelObj

  SimBiology Model - cell

  Model Components:
    Models:            0
    Parameters:        0
    Reactions:         0
    Rules:             0
    Species:           0
```

See Also   addcompartment, addconfigset, addevent, addkineticlaw, addmodel, addparameter, addreaction, addrule, addspecies, copyobj, get, sbioroot, sbiosimulate, set

Purpose   Nonlinear least-squares regression using SimBiology models

Syntax
```
results = sbionlinfit(modelObj, modelMapObject, pkDataObj,
    Beta0)
results = sbionlinfit(modelObj, modelMapObject, pkDataObj,
    Beta0, ...)
[results, SimDataI] = sbionlinfit(...)
```

Arguments   **Input Arguments**

modelObject	SimBiology model object used to fit observed data.
modelMapObject	PKModelMap object that defines the roles of the model components in the estimation. For more information, see PKModelMap object.
pkDataObj	PKData object that defines the data to use in fitting, and the roles of the data columns used for estimation. For more information, see PKData object.
Beta0	A vector of initial estimates for the fixed effects. The length of *Beta0* must equal at least the length of *modelMapObject*.Estimated. The final estimates for the fixed effects defined in *Beta0* are log transformed in the results. For more information on specifying initial estimates, see "Setting Initial Estimates" in the SimBiology User's Guide.

sbionlinfit

Output Arguments

results	A structure with the following fields:
	• beta — Fitted coefficients
	• R — Residuals
	• J — Jacobian of the model
	• COVB — Estimated covariance matrix for the fitted coefficients
	• mse — Estimate of the error of the variance term
SimDataI	*SimDataI* contains simulation results for individuals.

Description

Note This function requires nlinfit in Statistics Toolbox™ (Version 7.0 or later).

results = sbionlinfit(*modelObj*, *modelMapObject*, *pkDataObj*, *Beta0*) performs least-squares regression using the SimBiology model, *modelObj* and returns estimated results in the *results* structure.

results = sbionlinfit(*modelObj*, *modelMapObject*, *pkDataObj*, *Beta0*, ...) lets you supply additional arguments accepted by nlinfit. See nlinfit in the Statistics Toolbox User's Guide for information on the accepted properties.

[*results*, *SimDataI*] = sbionlinfit(...) returns simulations of the SimBiology model, *modelObj*, using the estimated values of the parameters.

See Also

SimBiology User's Guide, PKData object, PKModelDesign object, PKModelMap object, Model object, sbionlmefit, nlinfit in the Statistics Toolbox User's Guide

sbionlmefit

Purpose   Estimate nonlinear mixed effects using SimBiology models

Syntax
results = sbionlmefit(*modelObj*, *modelMapObject*, *pkDataObject*, *Beta0*)
results = sbionlmefit(*modelObj*, *modelMapObject*, *pkDataObject*, *Beta0*, ...)
[*results*, *SimDataI*, *SimDataP*] = sbionlmefit(...)

Arguments   **Input Arguments**

modelObject	SimBiology model object used to fit observed data.
modelMapObject	PKModelMap object that defines the roles of the model components used for estimation. For more information, see PKModelMap object.
pkDataObject	PKData object that defines the data to use in fitting and the roles of the columns used for estimation. *pkDataObject* must define target data for at least two groups. For more information, see PKData object.
Beta0	A vector of initial estimates for the fixed effects. The length of *Beta0* must equal *modelMapObject*.Estimated if you are not defining any additional covariates. The final estimates for the fixed and random effects for these parameters are log-transformed in the results. For more information on specifying initial estimates see "Setting Initial Estimates" in the SimBiology User's Guide.

sbionlmefit

Output Arguments

results	A structure with the following fields: • beta — Log-transformed estimates for the fixed effects • psi — Estimated covariance matrix of the random effects • stats — Structure with statistics such as AIC, BIC. For a complete list, refer to nlmefit in the Statistics Toolbox User's Guide. • b — Log-transformed estimates of the random effects for each group in *pkDataObject*.
SimDataI	*SimDataI* contains the data from simulating the model using the estimated parameter values, for individuals.
SimDataP	*SimDataP* contains the data from simulating the model using the estimated parameter values, for the population.

Description

Note This function requires nlmefit in Statistics Toolbox (Version 7.0 or later).

results = sbionlmefit(*modelObj*, *modelMapObject*, *pkDataObject*, *Beta0*) performs nonlinear mixed effects estimation using the SimBiology model, *modelObj* and returns estimated results in the *results* structure.

results = sbionlmefit(*modelObj*, *modelMapObject*, *pkDataObject*, *Beta0*, ...) lets you supply additional arguments accepted by nlmefit. Use the FEGroupDesign property to specify the design matrix for each

of the groups. See "Specifying the Covariate Model" in the SimBiology User's Guide.

sbionlmefit does not support the following arguments:

- FEConstDesign
- FEObsDesign
- FEParamsSelect
- REConstDesign
- REGroupDesign
- REObsDesign
- Vectorization
- Jacobian

See nlmefit in the Statistics Toolbox User's Guide for information on the accepted properties.

[*results*, *SimDataI*, *SimDataP*] = sbionlmefit(...) returns simulation data of the SimBiology model, *modelObj*, using the estimated values of the parameters.

See Also "Fitting Pharmacokinetic Model Parameters at the Command Line" in the SimBiology User's Guide, PKData object, PKModelDesign object, PKModelMap object, Model object, sbionlinfit, nlmefit in the Statistics Toolbox User's Guide

sbioparamestim

Purpose	Perform parameter estimation

Syntax

```
[k, result]= sbioparamestim(modelObj, tspan, xtarget,
    species_array, parameter_array)
[...]= sbioparamestim(..., species_array, parameter_array,
    k0)
[...]= sbioparamestim(..., species_array, parameter_array,
    k0, method)
```

Arguments

k	Vector of estimated parameter values.
result	struct with fields that provide information about the progress of optimization.
tspan	An n-by-1 vector representing the time span of the target data *xtarget*.
xtarget	An n-by-m matrix, where n is the number of time samples and m is the number of states you would like to match during the simulation. States can only be species varying with time. You cannot use time varying (nonconstant) parameters. The number of rows of *xtarget* must be the same as the number of rows of *tspan*.
species_array	Either an array of species objects or a cell array of names of species in *modelObj* whose amounts should be matched during the estimation process. The length of the *species_array* must be the same as the number of columns in *xtarget*. If there are species with duplicate names in different compartments, either use qualified names to identify the species correctly or use an array of species objects to identify the species correctly. sbioparamestim assumes that the order of the species in *species_array* is the same as the order used to specify columns of *xtarget*. For example, a qualified name for

	a species named sp1 that is in a compartment named comp2 is comp2.sp1.
parameter_array	Either an array of parameter objects or a cell array of names of parameters in *modelObj* whose values should be estimated. If you do not specify *parameter_array*, sbioparamestim estimates all the parameters in the model. When a vector of parameter initial values (*k0*) is not specified, sbioparamestim takes the initial values from *modelObj*. When there are parameters with duplicate names, use either parameter objects or qualified parameter names to identify the right parameter object. For example, for a parameter named param1 used in a reaction named reaction1 and at the kinetic law level, the qualified name is reaction1.param1.
k0	Array of doubles that holds initial values of parameters to be estimated. The length of *k0* is same as that of *parameter_array*. When you specify *k0*, sbioparamestim ignores any initial values specified in active variants attached to the model. If left unspecified, sbioparamestim takes initial values for parameters from the model (*modelObj*) or, if there are active variants, sbioparamestim uses any initial values specified in the active variants. See Variant object for more information about variants.
method	Either a string or a cell array. If it is a string, it must be the name of the optimization algorithm to be used during the estimation process. Valid values are 'fminsearch', 'lsqcurvefit', 'lsqnonlin', 'fmincon', 'patternsearch', 'patternsearch_hybrid', 'ga', or 'ga_hybrid'.

If it is a cell array, it must have two elements: the first one is the name of the optimization method as described before and the second element is a MATLAB `struct` as returned by `optimset`, `gaoptimset`, or `psoptimset`.

`sbioparamestim` uses the cell array option to specify user-defined optimization options. If you do not specify this argument, then it defaults to `'lsqcurvefit'` if the Optimization Toolbox™ is available; otherwise it defaults to `'fminsearch'`.

`'fminsearch'` is a part of basic MATLAB and does not require the Optimization Toolbox. Note that `'fminsearch'` is an unconstrained optimization method and this could result in negative values for parameters. In that case, use another optimization method.

Description

`[k, result]= sbioparamestim(`*modelObj*`, `*tspan*`, `*xtarget*`, `*species_array*`, `*parameter_array*`)` estimates parameters of the SimBiology model object (*modelObj*), specified in *parameter_array*, so as to match species given by *species_array* with the target state (*xtarget*), whose time variation is given by the time span *tspan*. *modelObj* must be a top-level SimBiology model. A top-level SimBiology model object has its `Parent` property set to the SimBiology root object.

`[...]= sbioparamestim(..., `*species_array*`, `*parameter_array*`, `*k0*`)` lets you specify the initial values of parameters.

`[...]= sbioparamestim(..., `*species_array*`, `*parameter_array*`, `*k0*`, `*method*`)` lets you specify the optimization method to use.

Examples

Example 1

Given a model and some target data, estimate all of its parameters without having to specify any initial values. This is the simplest case. Estimate all of its parameters using the default method.

1 Load a model from the project, gprotein_norules.sbproj. The project contains two models, one for the wild-type strain (stored in variable m1), and one for the mutant strain (stored in variable m2). Load the G protein model for the wild-type strain.

```
sbioloadproject gprotein_norules m1;
```

2 Store the target data in a variable.

```
Gt = 10000;
tspan  = [0 10 30 60 110 210 300 450 600]';
Ga_frac = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
xtarget = Ga_frac * Gt;
```

3 Store all model parameters in an array.

```
p_array = sbioselect(m1,'Type','parameter');
```

4 Store the species that should match target.

```
Ga = sbioselect(m1,'Type','species','Name','Ga');
% In this example only one species is selected.
% To match more than one targeted species data
% replace with selected species array.
```

5 Estimate the parameters.

```
[k, result] = sbioparamestim(m1, tspan, xtarget, Ga, p_array)

k =

    0.1988
    0.0000
    0.0045
    6.2859
    0.0040
    0.9726
    0.0000
    0.1164
```

sbioparamestim

```
result =

            fval: 8.7248e+005
        residual: [9x1 double]
        exitflag: 2
      iterations: 2
       funccount: 27
       algorithm: 'large-scale: trust-region reflective Newton'
         message: [1x77 char]
```

Example 2

Estimate parameters specified in p_array and species specified in
sp_array using different algorithms. This example uses the data from
"Example 1" on page 2-54.

```
[k1,r1] = sbioparamestim(m1, tspan, xtarget, Ga, p_array, ...
          {}, 'fmincon');
[k2,r2] = sbioparamestim(m1, tspan, xtarget, Ga, p_array, ...
          {}, 'patternsearch');
[k3,r3] = sbioparamestim(m1, tspan, xtarget, Ga, p_array, ...
          {}, 'ga')
```

Example 3

Estimate parameters specified in p_array, species specified in
sp_array, and change default optimization options to use user-specified
options. This example uses the data from "Example 1" on page 2-54.

```
myopt1 = optimset('Display','iter');
[k1,r1] = sbioparamestim(m1, tspan, xtarget, ...
          sp_array, p_array, {},{'fmincon', myopt1});

myopt2.Tolmesh = 1.0e-4;
[k2,r2] = sbioparamestim(m1, tspan, xtarget, ...
          sp_array, p_array, {},{'patternsearch', myopt2});

myopt3.PopulationSize = 50;
```

```
myopt3.Generations = 20;
[k3,r3] = sbioparamestim(m1, tspan, xtarget, ...
            sp_array, p_array, {},{'ga', myopt3});
```

Reference   Tau-Mu Yi, Hiroaki Kitano, and Melvin I. Simon. PNAS (2003) vol. 100, 10764–10769.

See Also   SimBiology functions `sbiomodel`, `sbiogetnamedstate`

MATLAB function `optimset`

Genetic Algorithm and Direct Search Toolbox™ functions `gaoptimset`, `psoptimset`

sbioparameter

Purpose   Construct parameter object

Note sbioparameter has been removed and produces an error. Use addparameter instead.

Syntax

```
parameterObj = sbioparameter(Obj, NameValue)
parameterObj = sbioparameter(Obj, NameValue, ValueValue)
parameterObj = sbioparameter(...'PropertyName', PropertyValue...)
```

Arguments

Obj	Model object or kinetic law object.
NameValue	Property for a parameter object. Enter a unique character string. Since objects can use this property to reference a parameter, a parameter object must have a unique name at the level it is created. For example, a kinetic law object cannot contain two parameter objects named kappa. However, the model object that contains the kinetic law object can contain a parameter object named kappa along with the kinetic law object.
	You can use the function sbioselect to find an object with a specific Name property value.
	For information on naming parameters, see Name.
ValueValue	Value of a parameter object. Enter a number.

Description   parameterObj = sbioparameter(*Obj*, *NameValue*) constructs a SimBiology parameter object, enters a value (*NameValue*) for the required property Name, and returns the object (parameterObj).

To use a parameter object (paramaterObj) in a simulation, you must add the object to a SimBiology model, or kinetic law object with the method

copyobj. You can use the addparameter method to simultaneously create and assign a parameter to a model or kinetic law object.

parameterObj = sbioparameter(Obj, *NameValue*, *ValueValue*) creates a parameter object, assigns a value (*NameValue*) to the property Name, assigns the value (*ValueValue*) to the property Value and returns the parameter object to a variable (parameterObj).

parameterObj = sbioparameter(...*'PropertyName'*, *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

Copy a SimBiology parameter object to a SimBiology model or kinetic law object with the method copyobj. Remove a parameter object from a model or kinetic law object with the method delete.

View additional parameter object properties with the get command. Modify additional parameter object properties with the set command. You can find help for parameterObj properties with the help *PropertyName* command and help for functions with the sbiohelp *FunctionName* command.

Method Summary

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

sbioparameter

Property Summary

Annotation	Store link to URL or file
ConstantValue	Specify variable or constant parameter value
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object
Value	Assign value to parameter object
ValueUnits	Parameter value units

Examples

1 Construct a parameter object.

```
parameterObj = sbioparameter('kappa', 1);
% View the help for the parameter object's Value property.
help(parameterObj, 'Value')
```

2 View parameter object properties.

```
get(parameterObj)
```

MATLAB returns:

```
     Annotation: ''
  ConstantValue: 1
           Name: 'kappa'
          Notes: ''
```

```
            Parent: [1x1 SimBiology.Reaction]
               Tag: ''
              Type: 'parameter'
          UserData: []
             Value: 4
        ValueUnits: '
```

See Also   addparameter, copyobj, sbiomodel

sbioplot

Purpose	Plot simulation results in one figure

Syntax

```
sbioplot(simDataObj)
sbioplot(simDataObj, fcnHandleValue, xArgsValue, yArgsValue)
```

Arguments

simDataObj	SimBiology data object.
fcnHandleValue	Function handle.
xArgsValue	Cell array with the names of the states.
yArgsValue	Cell array with the names of the states.

Description sbioplot(*simDataObj*) plots each simulation run for SimBiology data object, *simDataObj*, in the same figure. The plot is a time plot of each state in *simDataObj*. The figure also shows a hierarchical display of all the runs in a tree, with the ability of choosing which trajectories to show.

sbioplot(*simDataObj*, *fcnHandleValue*, *xArgsValue*, *yArgsValue*) plots each simulation run for the SimBiology data object, *simDataObj*, in the same figure. The plot is created by calling the function handle, *fcnHandleValue*, with input arguments *simDataObj*, *xArgsValue*, and *yArgsValue*.

xArgsValue and *yArgsValue* should be cell arrays with the names of the states. The function represented by the function handle should return an array of handles and names. The signature of the function is shown below.

```
function [handles, names] = functionName(simDataObj, xArgsValue, YArgsValue)
```

The output argument handles is a two-dimensional array of handles to the lines plotted by the function. Each column corresponds to a run and each row corresponds to the lines being plotted for a state. names is a one-dimensional cell array that contains the names to be displayed on the nodes which are children of a Run Node. The length of names should be equal to the number of rows in the handles array returned.

Examples   This example shows how to plot data from an ensemble run without interpolation.

```
% Load the radiodecay model.
        sbioloadproject('radiodecay.sbproj','m1');

        % Configure the model to run with the stochastic solver.
        cs = getconfigset(m1, 'active');
        set(cs, 'SolverType', 'ssa');
        set(cs.SolverOptions, 'LogDecimation', 100);

        % Run an ensemble simulation and view the results.
        simDataObj = sbioensemblerun(m1, 10, 'linear');
        sbioplot(simDataObj);
```

See Also   sbiosubplot

sbioreaction

Purpose Construct reaction object

Note sbioreaction has been removed and produces an error. Use addreaction instead.

Syntax

reactionObj = sbioreaction('*ReactionValue*')
reactionObj = sbioreaction('*ReactantsValue*',
    '*ProductsValue*')
reactionObj = sbioreaction('*ReactantsValue*',
    RStoichCofficients, '*ProductsValue*', *PStoichCofficients*)
reactionObj = sbioreaction(...'*PropertyName*', *PropertyValue*...)

Arguments

ReactionValue	Specify the reaction equation. Enter a character string. A hyphen preceded by a space and followed by a right angle bracket (->) indicates reactants going forward to products. A hyphen with left and right angle brackets (<->) indicates a reversible reaction. Coefficients before reactant or product names must be followed by a space. Examples are 'A -> B', 'A + B -> C', '2 A + B -> 2 C', 'A <-> B'.
ReactantsValue	A string defining the species name, a cell array of strings, a species object or an array of species objects.
ProductsValue	A string defining the species name, a cell array of strings, a species object or an array of species objects.

2-64

RStoichCofficients	Stoichiometric coefficients for reactants, length of array equal to length of *ReactantsValue*.
PStoichCofficients	Stoichiometric coefficients for products, length of array equal to length of *ProductsValue*.

Description *reactionObj* = sbioreaction('*ReactionValue*') creates a SimBiology reaction object, assigns a value (ReactionValue) to the property Reaction, and returns the reaction object (*reactionObj*).

To use reactionObj in a simulation, you must add reactionObj to a SimBiology model object using copyobj. You can use addreaction to simultaneously create a reaction object and add it to a model object. A SimBiology model object is constructed with the function sbiomodel.

reactionObj = sbioreaction('*ReactantsValue*', '*ProductsValue*') constructs a SimBiology reaction object that contains reactant species (Reactants) and product species (Products). The stoichiometric values are assumed to be 1. Reactants and Products can be a string defining the species name, a cell array of strings, a species object, or an array of species objects.

reactionObj = sbioreaction('*ReactantsValue*', *RStoichCofficients*, '*ProductsValue*', *PStoichCofficients*) adds stoichiometric coefficients (*RStoichCofficients*) for reactant species, and stoichiometric coefficients (PStoichCofficients) for product species, to the property Stoichiometry. The length of Reactants and RCofficients must be equal, and the length of Products and PCofficients must be equal.

reactionObj = sbioreaction(...'*PropertyName*', *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

View additional reactionObj properties with the get command. Modify additional reactionObj properties with the set command. You can find

help for `reactionObj` properties with the `help` *PropertyName* command and help for functions with the `sbiohelp` *FunctionName* command.

A reaction object that does not have a parent can contain only species objects that do not have a parent. If a parented species object is added to an unparented reaction object, a copy of the species object will be made and added to the reaction as an unparented species.

When an unparented reaction object is added to a model, the method checks the model for the required species. If the model contains the species, the reaction object now uses the model's species object. If the model does not contain the species, the species object is added to the model and the reaction object uses it.

Method Summary

addkineticlaw (reaction)	Create kinetic law object and add to reaction object
addproduct (reaction)	Add product species object to reaction object
addreactant (reaction)	Add species object as reactant to reaction object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmproduct (reaction)	Remove species object from reaction object products
rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

Property Summary	Active	Indicate object in use during simulation
	Annotation	Store link to URL or file
	KineticLaw	Show kinetic law used for `ReactionRate`
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	Parent	Indicate parent object
	Products	Array of reaction products
	Reactants	Array of reaction reactants
	Reaction	Reaction object reaction
	ReactionRate	Reaction rate equation in reaction object
	Reversible	Specify whether reaction is reversible or irreversible
	Stoichiometry	Species coefficients in reaction
	Tag	Specify label for SimBiology object
	Type	Display top-level SimBiology object type
	UserData	Specify data to associate with object

Examples **1** Construct reaction objects.

```
reactionObj1 = sbioreaction('a + 3 b -> 2 c');
reactionObj2 = sbioreaction({'a', 'b'}, [1 3], 'c', 2);
% View the help for the reaction object's Reversible property.
```

sbioreaction

```
help(reactionObj1, 'Reversible')
```

2 View the property summary for reactionObj1.

```
get(reactionObj1)

            Active: 1
        Annotation: ''
        KineticLaw: []
              Name: ''
             Notes: ''
        Parameters: [0x1 double]
            Parent: []
          Products: [1x1 SimBiology.Species]
         Reactants: [2x1 SimBiology.Species]
          Reaction: 'a + 3 b -> 2 c'
      ReactionRate: ''
        Reversible: 0
     Stoichiometry: [-1 -3 2]
               Tag: ''
              Type: 'reaction'
          UserData: []
```

See Also   addreaction, sbiomodel

Purpose   Create user-defined unit

Note sbioregisterunit has been removed and produces an error.
Use sbiounit followed by sbioaddtolibrary instead.

Syntax   sbioregisterunit('*Name*', '*Composition*', *Multiplier*)
               sbioregisterunit('*Name*', '*Composition*', *Multiplier*, *Offset*)

Description   sbioregisterunit('*Name*', '*Composition*', *Multiplier*)
               creates a unit with the name *Name*, where the unit is defined
               as Multiplier*Composition and records the unit in the
               UserDefinedUnits vector of sbioroot and adds it to the user-defined
               library.

               sbioregisterunit('*Name*', '*Composition*', *Multiplier*, *Offset*)
               creates a unit with the specified offset. You can list available units
               with the sbioshowunits function.

               • *Name* is the name of the user-defined unit. *Name* must begin with
                 characters and can contain characters, underscores or numbers. *Name*
                 can be any valid MATLAB variable name.

               • *Composition* shows the combination of base and derived units that
                 defines the unit *Name*. For example, molarity is mole/liter. Base
                 units are the set of units used to define all unit quantity equations.
                 Derived units are defined using base units or mixtures of base and
                 derived units.

               • *Multiplier* is the numerical value that defines the relationship
                 between the unit *Name* and the base unit as a product of
                 the *Multiplier* and the base unit. For example, 1 mole is
                 6.0221e23*molecule. The *Multiplier* is 6.0221e23.

               • *Offset* is the numerical value by which the unit composition
                 is modified from the base unit. For example, Celsius =
                 (5/9)*(Fahrenheit-32); *Multiplier* is 5/9 and *Offset* is 32.

sbioregisterunit

See Also   sbioaddtolibrary, sbioremovefromlibrary, sbioshowunits, sbiounit

Purpose   Create user-defined unit prefix

> **Note** sbioregisterunitprefix has been removed and produces an error. Use sbiounitprefix followed by sbioaddtolibrary instead.

Syntax   sbioregisterunitprefix('*NameValue*', *Exponent*)

Description   sbioregisterunitprefix('*NameValue*', *Exponent*) creates a unit prefix with the name *NameValue* and with a multiplicative factor of 10^Exponent, and adds it to the UserDefinedUnitPrefixes vector in sbioroot and to the user-defined library. You can see the available unit prefixes with the sbioshowunitprefixes function.

- *NameValue* is the name of the prefix. Valid names must begin with a letter and can contain characters, underscores, or numbers. Built-in prefixes are defined based on the International System of Units (SI).

- *Exponent* shows the value of 10^Exponent that defines the relationship of the unit *Name* to the base unit. For example, for the unit picomole, Exponent is 12.

See Also   sbioaddtolibrary, sbioremovefromlibrary, sbioshowunitprefixes, sbiounitprefix

sbioremovefromlibrary

Purpose   Remove abstract kinetic law, unit, or unit prefix from library

Syntax   sbioremovefromlibrary (Obj)
                   sbioremovefromlibrary ('Type', 'Name')

Description   sbioremovefromlibrary (Obj) removes the abstract kinetic law, unit, or unit prefix object (Obj) from the user-defined library. The removed component will no longer be available automatically in future MATLAB sessions.

sbioremovefromlibrary does not remove an abstract kinetic law that is being used in a model.

You can use a built-in or user-defined abstract kinetic law when you construct a kinetic law object with the method addkineticlaw.

sbioremovefromlibrary ('Type', 'Name') removes the object of type 'Type' with name 'Name' from the corresponding user-defined library. Type can be 'kineticlaw', 'unit' or 'unitprefix'.

To get a component of the built-in and user-defined libraries, use the commands get(sbioroot, 'BuiltInLibrary') and get(sbioroot, 'UserDefinedLibrary').

To create an abstract kinetic law, unit, or unit prefix, use sbioabstractkineticlaw, sbiounit, or sbiounitprefix respectively.

To add an abstract kinetic law, unit, or unit prefix to the user-defined library, use the function sbioaddtolibrary.

Example   This example shows how to remove an abstract kinetic law from the user-defined library.

1 Create an abstract kinetic law.

```
abstkineticlawObj = sbioabstractkineticlaw('mylaw1', '(k1*s)/(k2+k1+s)');
```

2 Add the new abstract kinetic law to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

sbioaddtolibrary adds the abstract kinetic law to the user-defined library. You can verify this using sbiowhos.

```
sbiowhos -kineticlaw -userdefined

SimBiology Abstract Kinetic Law Array

Index:    Library:      Name:      Expression:
1         UserDefined   mylaw1     (k1*s)/(k2+k1+s)
```

3 Remove the abstract kinetic law.

```
sbioremovefromlibrary('kineticlaw', 'mylaw1');
```

See Also   sbioaddtolibrary, sbioabstractkineticlaw, sbiounit, sbiounitprefix

sbioreset

Purpose   Delete all model and simulation objects

Syntax   `sbioreset`

Description   `sbioreset` deletes all SimBiology model and simulation objects at the root level. You cannot use a SimBiology model or simulation object after it is deleted. You should remove objects from the MATLAB workspace with the function `clear`.

The SimBiology root object contains a list of SimBiology model objects, available units, unit prefixes, and kinetic law objects. A SimBiology model object has its `Parent` property set to the SimBiology root object.

To add an abstract kinetic law to the SimBiology root user-defined library, use the `sbioaddtolibrary` function. To add a unit to the SimBiology user-defined library on the root, use the `sbioregisterunit` function. To add a unit prefix to the SimBiology user-defined library on the root, use the `sbioregisterunitprefix` function.

Example   This example shows the difference between `sbioreset` and `clear all`.

1 Import a model into the workspace.

```
modelObj = sbmlimport('oscillator');
```

Note that the workspace contains `modelObj` and if you query the SimBiology root, there is one model on the root object.

```
rootObj = sbioroot

SimBiology Root Contains:

 Models:                          1
 Builtin Abstract Kinetic Laws:   3
 User Abstract Kinetic Laws:      0
 Builtin Units:                   54
 User Units:                      0
 Builtin Unit Prefixes:           13
```

```
        User Unit Prefixes:              0
```

2 Use `clear all` to clear the workspace. The `modelObj` still exists on the `rootObj`.

```
clear all

rootObj


SimBiology Root Contains:

 Models:                          1
 Builtin Abstract Kinetic Laws:   3
 User Abstract Kinetic Laws:      0
 Builtin Units:                   54
 User Units:                      0
 Builtin Unit Prefixes:           13
 User Unit Prefixes:              0
```

3 Use `sbioreset` to delete the `modelObj` from the root.

```
sbioreset
rootObj


SimBiology Root Contains:

 Models:                          0
 Builtin Abstract Kinetic Laws:   3
 User Abstract Kinetic Laws:      0
 Builtin Units:                   54
 User Units:                      0
 Builtin Unit Prefixes:           13
 User Unit Prefixes:              0
```

See Also   sbioroot

sbioroot

Purpose	Return SimBiology root object

Syntax   *rootObj* = sbioroot

Arguments

rootObj	Return sbioroot to this object.

Description   *rootObj* = sbioroot returns the SimBiology root object to root. The SimBiology root object contains a list of the top-level SimBiology model objects, available units, unit prefixes, and available abstract kinetic law objects.

The units define the set of built-in units and user-defined units. See Unit object for more information.

The unit prefixes define the set of built-in prefixes and user-defined prefixes. See Unit Prefix object for more information.

The abstract kinetic law objects define the built-in abstract kinetic law objects and user-defined abstract kinetic law objects. The process of defining a reaction requires the use of abstract kinetic law objects when configuring a SimBiology reaction object's KineticLaw property with the addkineticlaw function.

To add a unit, prefix or abstract kinetic law to the root (in the user-defined library), use the sbioaddtolibrary function. To remove, use sbioremovefromlibrary.

The models opened in the SimBiology desktop are stored in the root object.

Method Summary

copyobj (any object)	Copy SimBiology object and its children
get (any object)	Get object properties

reset (root)	Delete all model objects from root object
set (any object)	Set object properties

Property Summary

BuiltInLibrary	Library of built-in components
Models	Contain all model objects
Type	Display top-level SimBiology object type
UserDefinedLibrary	Library of user-defined components

See Also   addkineticlaw, sbiomodel, sbioreset

sbiorule

Purpose   Construct rule object

Note sbiorule has been removed and produces an error. Use addrule instead.

Syntax   *ruleObj* = sbiorule('*RuleValue*')
              ruleObj = sbiorule(*RuleValue*, '*RuleTypeValue*')
              ruleObj = sbiorule(...'*PropertyName*', *PropertyValue*...)

Arguments

RuleValue	Enter a character string within quotation marks. For example, enter the algebraic rule 'Va*Ea + Vi*Ei - K2'.
RuleTypeValue	Enter 'algebraic', 'initialassignment', 'repeatedAssignment', or 'rate'. See RuleType for more information.

Description   A SimBiology rule is a mathematical expression that modifies a species amount, or a parameter value. A rule is a MATLAB expression that uses species, and parameters.

ruleObj = sbiorule('*RuleValue*') creates a rule object, assigns a value (*RuleValue*) to the property Rule, assigns the value 'algebraic' to the property RuleType, and assigns the root object to the property Parent.

To use ruleObj in a simulation, ruleObj must be added to a model object with the function copyobj. Note that a rule can also be added to a SimBiology model with the addrule function. A model object is constructed with the function sbiomodel.

ruleObj = sbiorule(*RuleValue*, '*RuleTypeValue*') in addition to the above, this syntax enables you to specify RuleType.

ruleObj = sbiorule(...'*PropertyName*', *PropertyValue*...) defines optional properties. The property name/property value pairs can be in

any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

View additional rule properties with the function `get`, and modify rule properties with the function `set`. View the rules in a model (`modelObj`) with `get(modelObj, 'Rules')`.

Method Summary		
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	set (any object)	Set object properties

Property Summary		
	`Active`	Indicate object in use during simulation
	`Annotation`	Store link to URL or file
	`Name`	Specify name of object
	`Notes`	HTML text describing SimBiology object
	`Parent`	Indicate parent object
	`Rule`	Specify species and parameter interactions
	`RuleType`	Specify type of rule for rule object
	`Tag`	Specify label for SimBiology object

sbiorule

Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples

Example 1

Construct a rule object and copy it to a model object.

```
ruleObj = sbiorule('Enzt - Enzi - Enza)');
modelObj = sbiomodel('cell')
ruleObj_copy = copyobj(ruleObj, modelObj);
```

Example 2

View the help for the rule object's `RuleType` property.

```
help(ruleObj, 'RuleType')
```

Example 3

List the properties for a rule.

```
get(ruleObj)

        Active: 1
    Annotation: ''
          Name: ''
         Notes: ''
        Parent: []
          Rule: 'myrule'
      RuleType: 'algebraic'
           Tag: ''
          Type: 'rule'
      UserData: []
```

See Also   addrule, copyobj, sbiomodel

Purpose   Save all models in root object

Syntax   sbiosaveproject *projFilename*
sbiosaveproject *projFilename* *variableName*
sbiosaveproject *projFilename* *variableName1* *variableName2* ...

Description sbiosaveproject *projFilename* saves all models in the
SimBiology root object to the binary SimBiology project file
named *projFilename*.sbproj. The project can be loaded
with sbioloadproject. sbiosaveproject returns an error if
projFilename.sbproj is not writable.

sbiosaveproject creates the binary SimBiology project file named
simbiology.sbproj. sbiosaveproject returns an error if this is not
writable.

sbiosaveproject *projFilename* *variableName* saves only *variableName*.
variableName can be a SimBiology model or any MATLAB variable.

sbiosaveproject *projFilename* *variableName1* *variableName2* ...
saves the specified variables in the project.

Use the functional form of sbiosaveproject when the file name or
variable names are stored in a string. For example, if the file name
is stored in the variable *fileName* and you want to store MATLAB
variables *variableName1* and *variableName2*, type sbiosaveproject
(*projFileName*, '*variableName1*', '*variableName2*') at the
command line.

Examples   **1** Import an SBML file and simulate (default configset object is used).

```
modelObj = sbmlimport ('oscillator.xml');
timeseriesObj = sbiosimulate(modelObj);
```

2 Save the model and the simulation results to a project.

```
sbiosaveproject myprojectfile modelObj timeseriesObj
```

sbiosaveproject

See Also   sbioaddtolibrary, sbioloadproject, sbioremovefromlibrary, sbiowhos

Purpose   Search for objects with specified constraints

Syntax   *Out* = sbioselect('*PropertyName*', *PropertyValue*)
                  Out = sbioselect('Where', '*PropertyName*', '*Condition*',
                     PropertyValue)
                  Out = sbioselect(*Obj*, '*PropertyName*', *PropertyValue*)
                  Out = sbioselect(*Obj*, 'Type', '*TypeValue*', '*PropertyName*',
                     PropertyValue)
                  Out = sbioselect(*Obj*, 'Where', '*PropertyName*', '*Condition*',
                     PropertyValue)
                  Out = sbioselect(*Obj*, 'Where', '*PropertyName1*', '*Condition1*',
                     PropertyValue1, 'Where', '*PropertyName2*', '*Condition2*',
                     PropertyValue2,...)
                  Out = sbioselect(*Obj*, 'Depth', *DepthValue*,...)

Arguments

Out	Object or array of objects returned by the sbioselect function. *Out* might contain a mixture of object types (for example, species and parameters), depending on the selection you specify. If *PropertyValue* is a cell array, then the function returns all objects with the property '*PropertyName*' that matches any element of *PropertyValue*.
Obj	SimBiology object or array of objects to search. If an object is not specified, sbioselect searches the root.
PropertyName	Any property of the object being searched.
PropertyValue	Specify *PropertyValue* to include in the selection criteria.
TypeValue	Type of object to include in the selection, for example, sbiomodel, species, reaction, or kineticlaw.

sbioselect

Condition		Constraint to impose on the search. See the table under "Description" on page 2-84 for a list of conditions.
DepthValue		Specify the depth number to search. Valid numbers are positive integer values and `inf`. If *DepthValue* is `inf`, sbioselect searches *Obj* and all of its children. If *DepthValue* is `1`, sbioselect only searches *Obj* and not its children. By default, *DepthValue* is `inf`.

Description   sbioselect searches for objects with specified constraints.

Out = sbioselect('*PropertyName*', *PropertyValue*) searches the root object (including all model objects contained by the root object) and returns the objects with the property name (*PropertyName*) and property value (*PropertyValue*) contained by the root object.

Out = sbioselect('Where', '*PropertyName*', '*Condition*', *PropertyValue*) searches the root object and finds objects that have a property name (*PropertyName*) and value (*PropertyValue*) that matches the condition (*Condition*).

Out = sbioselect(*Obj*, '*PropertyName*', *PropertyValue*) returns the objects with the property name (*PropertyName*) and property value (*PropertyValue*) found in any object (*Obj*).

Out = sbioselect(*Obj*, 'Type', '*TypeValue*', '*PropertyName*', *PropertyValue*) finds the objects of type (TypeValue), with the property name (*PropertyName*) and property value (*PropertyValue*) found in any object (*Obj*). *TypeValue* is the type of SimBiology object to be included in the selection, for example, `species`, `reaction`, or `kineticlaw`.

Out = sbioselect(*Obj*, 'Where', '*PropertyName*', '*Condition*', *PropertyValue*) finds objects that have a property name (*PropertyName*) and value (*PropertyValue*) that match the condition (*Condition*).

If you search for a string property value without specifying a condition, you must use the same format as get returns. For example, if get returns the Name as 'MyObject', sbioselect will not find an object

with a `Name` property value of `'myobject'`. Therefore, for this example, you must specify:

```
modelObj = sbioselect ('Name', 'MyObject')
```

Instead, if you use a condition, you can specify:

```
modelObj = sbioselect ('Where', 'Name', '==i', 'myobject')
```

Thus, conditions let you control the specificity of your selection.

`sbioselect` searches for model objects on the root in both cases.

The conditions, with examples of property names and corresponding examples of property values that you can use, are listed in the following tables. This table shows you conditions for numeric properties.

Conditions for Numeric Properties	Example Syntax

==	Search in the model object (modelObj), and return parameter objects that have Value equal to 0.5. sbioselect returns parameter objects because only parameter objects have a property called Value.
	
	`parameterObj = sbioselect (modelObj,...`
	`  'Where', 'Value', '==', 0.5)`
	
	In the case of ==, this is equivalent to omitting the condition as shown:
	
	`parameterObj = sbioselect (modelObj,...`
	`  'Value', 0.5)`
	
	Search in the model object (modelObj), and return parameter objects that have ConstantValue false (nonconstant parameters).
	
	`parameterObj = sbioselect (modelObj,...`
	`  'Where', 'ConstantValue', '==', false)`

~=	Search in the model object (modelObj), and return parameter objects that do not have Value equal to 0.5.
	
	`parameterObj = sbioselect (modelObj,...`
	`  'Where', 'Value', '~=', 0.5)`

Conditions for Numeric Properties	Example Syntax

>,<,>=,<=	Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) greater than 50.
	
	``speciesObj = sbioselect (modelObj, ...
	  'Where', 'InitialAmount', '>', 50)```
	
	Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) less than or equal to 50.
	
	``speciesObj = sbioselect (modelObj,...
	  'Where', 'InitialAmount', '<=', 50)```

between	Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) between 200 and 300.
	
	``speciesObj = sbioselect (modelObj,...
	  'Where', 'InitialAmount',...
	  'between', [200 300])```

~between	Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) that is not between 200 and 300.
	
	``speciesObj = sbioselect (modelObj,...
	  'Where', 'InitialAmount',...
	  '~between', [200 300])```

The following table shows you conditions for properties whose values are strings.

Conditions for String Properties	Example Syntax
==	Search in the model object (modelObj), and return species objects named 'Glucose'. ```speciesObj = sbioselect (modelObj,... 'Type', 'species', 'Where',... 'Name', '==', 'Glucose')```
~=	Search in the model object (modelObj), and return species objects that are not named 'Glucose'. ```speciesObj = sbioselect (modelObj,... 'Type', 'species', 'Where',... 'Name', '~=', 'Glucose')```
==i	Same as ==; in addition, this is case insensitive.
~=i	Search in the model object (modelObj), and return species objects that are not named 'Glucose', ignoring case. ```speciesObj = sbioselect (modelObj,... 'Type', 'species', 'Where',... 'Name', '~=i', 'glucose')```

Conditions for String Properties	Example Syntax
regexp. Supports expressions supported by the functions regexp and regexpi.	Search in the model object (modelObj), and return objects that have 'ese' or 'ase' anywhere within the name. ``` Obj = sbioselect (modelObj, 'Where',... 'Name', 'regexp', '[ea]se') ``` Search in the root, and return objects that have kinase anywhere within the name. ``` Obj = sbioselect ('Where',... 'Name', 'regexp', 'kinase') ``` Note that this query could result in a mixture of object types (for example, species and parameters).
regexpi	Same as regexp; in addition, this is case insensitive.
~regexp	Search in the model object (modelObj), and return objects that do not have kinase anywhere within the name. ``` Obj = sbioselect (modelObj, 'Where',... 'Name', '~regexp', 'kinase') ```
~regexpi	Same as ~regexp; in addition, this is case insensitive.

The condition 'contains' can be used only for those properties whose values are an array of SimBiology objects. The following table shows you an example of using contains.

Condition	Example Syntax

'contains'	Search in the model object and return reaction objects whose Reactant property contains the specified species.
	
	`Out = sbioselect(modelObj, 'Where',...`
	`'Reactants', 'contains',...`
	`modelObj.Species(1))`

Out = sbioselect(*Obj*, 'Where', *'PropertyName1'*, *'Condition1'*, *PropertyValue1*, 'Where', *'PropertyName2'*, *'Condition2'*, *PropertyValue2*,...) finds objects contained by Obj that matches all the conditions specified.

You can combine any number of property name/property value pairs and conditions in the sbioselect command.

Out = sbioselect(*Obj*, 'Depth', *DepthValue*,...) finds objects using a model search depth of *DepthValue*.

Examples

1 Import a model.

```
modelObj = sbmlimport('oscillator');
```

2 Find and return an object named pA.

```
Obj = sbioselect(modelObj, 'Name', 'pA');
```

3 Find and return species objects whose Name starts with p and have A or B as the next letter in the name.

```
speciesObj = sbioselect(modelObj, 'Type', 'species', 'Where',...
 'Name', 'regexp', '^p[AB]');
```

4 Find a cell array. Note how cell array values must be specified inside another cell array.

```
modelObj.Species(2).UserData = {'a' 'b'};
Obj = sbioselect(modelObj,'UserData',{{'a' 'b'}})

SimBiology Species Array

Index:   Compartment:   Name:   InitialAmount:  InitialAmountUnits:
1        unnamed        pB      0
```

See Also   regexp

sbiosetdosingprofile

Purpose	Add objects to model for dosing

Syntax

modModelObj = sbiosetdosingprofile(*modelObj*, *DoseVariable*,
    DosingTable, *DosingType*)

Arguments

modModelObj	SimBiology model object to which sbiosetdoingprofile adds event objects for dosing.
modelObj	SimBiology model object to which you want to add dosing information.
DoseVariable	Name of a species or parameter object in *modelObject*. sbiosetdosingprofile adds events that change the amount or value of the *DoseVariable* as defined by the *DosingTable* and *DosingType*.
DosingTable	An n x 2 matrix with the first column defining the time of the dose and the second column defining the dose amount. If *DosingType* is 'Infusion', then *DosingTable* must be nx3 with a third column that defines the rate of infusion.
DosingType	Specifies the mechanism for drug dosing. Valid options are 'Bolus', 'Infusion', 'ZeroOrder', 'FirstOrder' and ''. For more information on *DosingType* see "About Dosing Types" in the SimBiology User's Guide.

Description   *modModelObj* = sbiosetdosingprofile(*modelObj*, *DoseVariable*,
DosingTable, *DosingType*) adds SimBiology event and rule objects
to the model object *modelObj* to perform the dosing as specified by
DoseVariable, *DosingTable*, and *DosingType* and returns the modified
model object, *modModelObj*.

While SimBiology performs dimensional checks and unit conversions
for components in a model, it does not handle units for quantities in

DosingTable. For more information about units in *DosingTable*, see "Unit Conversion for Imported Data and the Model" in the SimBiology User's Guide.

To facilitate the removal of the modified components from the model object, sbiosetdosingprofile adds the keyword `'DOSE_COMPONENT'` to the `Tag` property for the events and the associated parameters added to the model object, *modelObject*.

To remove the modified components from the modified model object, *modModelObject*, use `delete(sbioselect(modModelObject, 'Tag', 'DOSE_COMPONENT'));`.

Model Components Added by sbiosetdosingprofile

sbiosetdosingprofile adds the following model components based on the dosing type you selected.

Bolus:

- A parameter, `BolusDoseAsAmount`, that holds the dose amount for the current dose.

- For a dose at time = 0

  - An initial assignment rule that increases the initial drug concentration (by default = 0), by `BolusDoseAsAmount` (set to the dose table amount for time = 0) divided by the compartment volume.

- For doses at time > 0

  - A pair of parameters, `fire_parameter_on` and `fire_parameter_off`, that allow SimBiology to trigger a second event from an initial event for each dose.

  - A parameter for each dose, `DoseAmount_n`, that holds the dose amount at that time.

  - A parameter for each dose, `DoseTime_n`, that holds that dose time.

  - An event at `DoseTime_n` that sets `BolusDoseAsAmount` to `DoseAmount_n` and triggers the secondary event.

- A second event that increases the drug concentration by `BolusDoseAsAmount` divided by the compartment volume at that time.

Infusion:

- For a dose at time = 0

  - A parameter, `InfusionRate_01`, that represents the infusion rate for the first dose.

  - An initial assignment rule to set the parameter representing the rate of infusion (`kInf`) to `InfusionRate_01`.

  - A parameter, `InfusionStopTime_01`, that represents the end of the dose. The parameter's value is defined by `amount/rate of infusion`. Amount and rate are values that are specified in the second and third columns respectively of *DosingTable*.

  - An event at `InfusionStopTime_01` that reduces the value of `kInf` by `InfusionRate_01`.

- For each dose at time > 0

  - A parameter, `InfusionStartTime_n`, that represents the start of the dose.

  - A parameter, `InfusionStopTime_n`, that represents the end of the dose. This value is defined as `InfusionStartTime_n + amount/rate of infusion`. Amount and rate are values that are specified in the second and third columns respectively of *DosingTable*.

  - A parameter, `InfusionRate_n`, that represents the infusion rate for the dose.

  - An event at `InfusionStartTime_n` that increases the value of `kInf` by `InfusionRate_n`.

  - An event at `InfusionStopTime_n` that reduces the value of `kInf` by `InfusionRate_n`.

Zero-order:

Note For custom models, include a parameter named Tk0, representing the duration of drug absorption.

- A parameter, ZeroOrderDoseAsAmount, that represents the current dose amount.

- A pair of parameters, fire_parameter_on and fire_parameter_off, that enable triggering a second event from an initial event for each dose.

- For a dose at time = 0

  - An initial assignment rule to set the parameter representing absorption rate constant (ka) to ZeroOrderDoseAsAmount/Tk0.

  - A parameter, DoseAmount_01, that represents the dose amount for the first dose. Amount is specified in the second column of *DosingTable*.

  - An event at Tk0 that reduces the value of ZeroOrderDoseAsAmount by DoseAmount_01, and triggers the secondary event.

  - A second event immediately following the change in ZeroOrderDoseAsAmount to set ka to ZeroOrderDoseAsAmount/Tk0.

- For each dose at time > 0

  - A parameter, DoseAmount_n), that represents the dose amount for the dose.

  - A parameter, DoseStartTime_n, that represents the start of the dose. This value is defined in the first column of *DosingTable*.

  - An event at DoseStartTime_n that increases the value of ZeroOrderDoseAsAmount by DoseAmount_n, and triggers a secondary event.

- A second event immediately following the change in `ZeroOrderDoseAsAmount` to set `ka` to `ZeroOrderDoseAsAmount/TkO`.

- An event at `DoseStartTime_n + TkO` that reduces the value of `ZeroOrderDoseAsAmount` by `DoseAmount_n`, and triggers the secondary event.

- A second event immediately following the change in `ZeroOrderDoseAsAmount` to set `ka` to `ZeroOrderDoseAsAmount/TkO`.

First-order:

- For a dose at time = 0

  - An initial assignment rule that increases `Dose` by the dose amount at `time = O`.

- For each dose at time > 0

  - A parameter, `DoseStartTime_n`, that represents the time of the dose.

  - A parameter, `DoseAmount_n`, that holds amount of the dose.

  - An event at `DoseStartTime_n` that increases the value of `Dose` by `DoseAmount_n`.

See Also   "Simulating a Model Containing Dosing Information" in the SimBiology User's Guide, `Event object`, `Model object`

Purpose	Show unit prefixes in library

Syntax

```
UnitPrefixObjs = sbioshowunitprefixes
[Name, Multiplier] = sbioshowunitprefixes
[Name, Multiplier, Builtin] = sbioshowunitprefixes
[Name, Multiplier, Builtin] = sbioshowunitprefixes('Name')
```

Arguments

unitPrefixObjs	Vector of unit prefix objects from the BuiltInLibrary and UserDefinedLibrary properties of the Root object.
Name	Name of the built-in or user-defined unit prefix. Built-in prefixes are defined based on the International System of Units (SI).
Multiplier	Shows the value of 10^Exponent that defines the relationship of the unit prefix *Name* to the base unit. For example, the multiplier in picomole is 10e-12.
Builtin	An array of logical values. If *Builtin* is true for a unit prefix, the unit prefix is built in. If *Builtin* is false for a unit prefix, the unit prefix is user defined.

Description

sbioshowunitprefixes returns information about unit prefixes in the SimBiology library.

UnitPrefixObjs = sbioshowunitprefixes returns the unit prefixes in the library as a vector of unit prefix objects in *UnitPrefixObjs*.

[*Name, Multiplier*] = sbioshowunitprefixes returns the multiplier for each prefix in *Name* to *Multiplier* as a cell array of strings.

[*Name, Multiplier, Builtin*] = sbioshowunitprefixes returns whether the unit prefix is built in or user defined for each unit prefix in *Name* to *Builtin*.

sbioshowunitprefixes

[*Name, Multiplier, Builtin*] = sbioshowunitprefixes('*Name*')
returns the name, multiplier, and built-in status for the unit prefix with
name *Name*. *Name* can be a cell array of strings.

Examples

```
[name, multiplier] = sbioshowunitprefixes;
[name, multiplier] = sbioshowunitprefixes('nano');
```

See Also   sbioconvertunits, sbioshowunits, sbiounitprefix

Purpose   Show units in library

Syntax   *unitObjs* = sbioshowunits
                  [*Name, Composition*] = sbioshowunits
                  [*Name, Composition, Multiplier*] = sbioshowunits
                  [*Name, Composition, Multiplier, Offset*] = sbioshowunits
                  [*Name, Composition, Multiplier, Offset,*
                     Builtin] = sbioshowunits
                  [*Name, Composition, Multiplier, Offset,*
                     Builtin] = sbioshowunits('*Name*')

Arguments

unitObjs	Vector of unit objects from the BuiltInLibrary and UserDefinedLibrary properties of the Root object.
Name	Name of the built-in or user-defined unit.
Composition	Shows the combination of base and derived units that defines the unit *Name*. For example, molarity is mole/liter.
Multiplier	The numerical value that defines the relationship between the unit *Name* and the base or derived unit as a product of the *Multiplier* and the base unit or derived unit. For example, 1 mole is 6.0221e23*molecule. The *Multiplier* is 6.0221e23.
Offset	Numerical value by which the unit composition is modified from the base unit. For example, Celsius = (5/9)*(Fahrenheit-32); *Multiplier* is 5/9 and *Offset* is 32.
Builtin	An array of logical values. If *Builtin* is true for a unit, the unit is built in. If *Builtin* is false for a unit, the unit is user defined.

sbioshowunits

Description   *unitObjs* = sbioshowunits returns the units in the library to *unitObjs* as a vector of unit objects.

[*Name, Composition*] = sbioshowunits returns the composition for each unit in *Name* to *Composition* as a cell array of strings.

[*Name, Composition, Multiplier*] = sbioshowunits returns the multiplier for the unit with name *Name* to *Multiplier*.

[*Name, Composition, Multiplier, Offset*] = sbioshowunits returns the offset for the unit with name *Name* to *Offset*. The unit is defined as *Multiplier*Composition+Offset*.

[*Name, Composition, Multiplier, Offset, Builtin*] = sbioshowunits returns whether the unit is built in or user defined for each unit in *Name* to *Builtin*.

[*Name, Composition, Multiplier, Offset, Builtin*] = sbioshowunits('*Name*') returns the name, composition, multiplier, offset and built-in status for the unit with name *Name*. *Name* can be a cell array of strings.

Examples
```
[name, composition] = sbioshowunits;
[name, composition] = sbioshowunits('molecule');
```

See Also   sbioconvertunits, sbioshowunitprefixes, sbiounit

Purpose   Simulate model object

Syntax   [t,x,names] = sbiosimulate(*modelObj*)
                   simDataObj = sbiosimulate(*modelObj*)
                   ... = sbiosimulate(*modelObj*, *configsetObj*)
                   ... = sbiosimulate(*modelObj*, *variantObj*)
                   ... = sbiosimulate(*modelObj*, *configsetObj*, *variantObj*)

Arguments   **Output Arguments**

t	An n-by-1 vector of time points. Shows the simulation time steps.
x	An n-by-m data array, where n is the number of time samples and m is the number of states logged in the simulation. Each column of x describes the variation in the quantity of a state over time.
names	An m-by-1 cell array of names. If the species are in multiple compartments, species names are qualified with the compartment name in the form compartmentName.speciesName. For example, nucleus.DNA, cytoplasm.mRNA.
	Parameter names are qualified with the reaction name if the parameter is scoped to the reaction's kinetic law. For example, Transcription.k1, denotes that the parameter k1 is scoped to the kinetic law for the reaction Transcription.
simdataObj	An object that holds time and state data as well as metadata, such as the types and names for the logged states or the configuration set used during simulation. You can access time, data, and names stored in *simdataObj* through *simdataObj* properties. See SimData object for more information.

sbiosimulate

Input Arguments

modelObj	Model object to be simulated.
configsetObj	Specify the configuration set object to use in the simulation. For more information about configuration sets, see Configset object.
variantObj	Specify the variant object to apply to the model during the simulation. For more information about variant objects, see Variant object.

Description

[t,x,names] = sbiosimulate(*modelObj*) simulates a model object (modelObj) using the active configuration set attached to the model (*modelObj*) and returns the specified outputs as described in "Output Arguments" on page 2-101.

simDataObj = sbiosimulate(*modelObj*) simulates the Simbiology model object (*modelObj*) and returns the results to a SimData object.

... = sbiosimulate(*modelObj*, *configsetObj*) simulates a model object (*modelObj*) using a configuration set (*configset*) that overrides the active configuration set attached to the model (*modelObj*). After the command is executed this override does not exist; the configuration set that is defined as 'active' is reinstated. To get the configuration sets attached to a model, use getconfigset. To attach a new or existing configuration set to a model, use addconfigset. To set the active configuration set of a model, use setactiveconfigset. For more information about configuration sets, see Configset object.

... = sbiosimulate(*modelObj*, *variantObj*) simulates a model object (*modelObj*), using the variant object or array of variant objects (*variantObj*).

... = sbiosimulate(*modelObj*, *configsetObj*, *variantObj*) simulates a model object (*modelObj*), using the configuration set object *configsetObj* and the variant object or array of variant objects (*variantObj*).

Property Summary	Configuration set property summary	
	Active	Indicate object in use during simulation
	CompileOptions	Dimensional analysis and unit conversion options
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	RuntimeOptions	Options for logged species
	SensitivityAnalysisOptions	Specify sensitivity analysis options
	SolverOptions	Specify model solver options
	SolverType	Select solver type for simulation
	StopTime	Set stop time for simulation
	StopTimeType	Specify type of stop time for simulation
	TimeUnits	Show stop time units for simulation
	Type	Display top-level SimBiology object type

Examples   The following examples show how to change solver settings.

Example 1

Create a SimBiology model from an SBML file, simulate the model using a solver other than the default solver (default is ode15s), and view the results.

1 Read the file for the oscillator model.

```
modelObj = sbmlimport('oscillator.xml');
```

2 Get the active `configset`.

```
configsetObj = getconfigset(modelObj, 'active');
```

3 Configure the `SolverType` to `ode45` and set `StopTime` to `10`.

```
set(configsetObj, 'SolverType', 'ode23s');
set(configsetObj, 'StopTime', 10);
```

4 Simulate the `modelObj`.

```
[t,x]= sbiosimulate(modelObj);
```

5 Plot the results of the simulation.

```
plot(t, x)
```

Example 2

Simulate the above example with `DimensionalAnalysis` off (set to `false`).

1 Repeat steps 1 and 2 above, then set dimensional analysis and unit conversion off in the `configset` object. `DimensionalAnalysis` and `UnitConversion` are properties of the `CompileOptions` object in the `configset` object.

```
set(configsetObj.CompileOptions, 'UnitConversion', false);
set(configsetObj.CompileOptions, 'DimensionalAnalysis', false);
```

2 Simulate the `modelObj`.

```
simDataObj = sbiosimulate(modelObj);
```

3 Plot the results of the simulation.

```
plot(simDataObj.Time, simDataObj.Data);
legend(simDataObj.DataNames)
```

See Also   `addconfigset`, `sbiomodel`

Purpose   Construct species object

> **Note** sbiospecies has been removed and produces an error. Use addspecies instead.

Syntax   *speciesObj* = sbiospecies('*NameValue*')
speciesObj = sbiospecies('*NameValue*'), *InitialAmountValue*)
speciesObj = sbiospecies(...'*PropertyName*', *PropertyValue*...)

Arguments

NameValue	Name for a species object. Enter a character string unique to the level of object creation. Species objects are identified by Name within ReactionRate and Rule property strings. You can use the function sbioselect to find an object with a specific Name property value.
	For information on naming species, see Name.
InitialAmountValue	Initial amount value for the species object. Enter double. Positive real number, default = 0.

Description   *speciesObj* = sbiospecies('*NameValue*') constructs a SimBiology.Species object, enters a value (*NameValue*) for the property Name, and returns the object (speciesObj).

speciesObj = sbiospecies('*NameValue*'), *InitialAmountValue*) in addition to the above, assigns an initial amount *(InitialAmountValue)* for the species.

Species are entities that take part in reactions. A species object represents these entities. There are reserved characters you cannot use in the species object name (*NameValue*).

In order for a species object to be used in a simulation, you must add the species object to a SimBiology model object using copyobj. You

can use addspecies to simultaneously create a species object and add it to a compartment object. A compartment object is constructed with the function addcompartment.

speciesObj = sbiospecies(...'*PropertyName*', *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

View species object properties with the function get, and change properties with the function set. You can find help for speciesObj properties with the help *PropertyName* command and help for functions with the sbiohelp *FunctionName* command.

A *species* is a chemical or entity that participates in reactions, for example, DNA, ATP, Pi, creatine , G-Protein, or Mitogen-Activated Protein Kinase (MAPK). Species amounts can vary or remain constant during a simulation.

If you change the Name property of a species you must configure all applicable elements, such as rules that use the species, any user-specified ReactionRate, or the kinetic law object property SpeciesVariableNames. Use the method setspecies to configure SpeciesVariableNames.

To update species names in the SimBiology graphical user interface, access each appropriate pane through the **Project Explorer**. You can also use the **Find** feature to locate the names that you want to update. The **Output** pane opens with the results of **Find**. Double-click a result row to go to the location of the model component.

Species names are automatically updated for reactions that use the MassAction kinetic law. See Name for more information about specifying species names.

Method Summary	Methods for species objects	
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	rename (compartment, parameter, species)	Rename object and update expressions
	set (any object)	Set object properties

Property Summary	Properties for species object	
	Annotation	Store link to URL or file
	BoundaryCondition	Indicate species boundary condition
	ConstantAmount	Specify variable or constant species amount
	InitialAmount	Species initial amount
	InitialAmountUnits	Species initial amount units
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	Parent	Indicate parent object
	Tag	Specify label for SimBiology object

sbiospecies

Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples   **Example 1**

Create a species (H2O) and view properties for the object.

1 Create a species object named H2O and an initial amount of 1000.

```
speciesObj = sbiospecies('H2O', 1000);
% View the help for the species object's InitialAmount property.
help(speciesObj, 'InitialAmount')
```

2 View properties for the species object.

```
get(speciesObj)
            Annotation: ''
     BoundaryCondition: 0
        ConstantAmount: 0
         InitialAmount: 1000
    InitialAmountUnits: ''
                  Name: 'H2O'
                 Notes: ''
                Parent: []
                   Tag: ''
                  Type: 'species'
              UserData: []
```

Example 2

Create two species: one is a reactant and the other is the enzyme catalyzing the reaction.

1 Create two species objects named glucose_6_phosphate and glucose_6_phosphate_dehydrogenase.

```
speciesObj1 = sbiospecies ('glucose_6_phosphate');
speciesObj2 = sbiospecies ('glucose_6_phosphate_dehydrogenase');
```

2 Set the initial amount of glucose_6_phosphate to 100 and verify.

```
set(speciesObj1, 'InitialAmount', 100);
get(speciesObj1, 'InitialAmount')
```

MATLAB returns:

```
ans =

    100
```

See Also   addspecies

MATLAB functions get, set

sbiosubplot

Purpose	Plot simulation results in subplots

Syntax

```
sbiosubplot(simDataObj)
sbiosubplot(simDataObj, fcnHandleValue, xArgsValue,
    yArgsValue)
sbiosubplot(simDataObj, fcnHandleValue, xArgsValue,
    yArgsValue, showLegendValue)
```

Arguments

simDataObj	SimBiology data object.
fcnHandleValue	Function handle.
xArgsValue	Cell array with the names of the states.
yArgsValue	Cell array with the names of the states.
showLegendValue	Boolean (default is false).

Description

sbiosubplot(*simDataObj*) plots each simulation run for SimBiology data object, *simDataObj* into its own subplot. The subplot is a time plot of each state in *simDataObj*. A legend is included.

sbiosubplot(*simDataObj*, *fcnHandleValue*, *xArgsValue*, *yArgsValue*) plots each simulation run for the SimBiology data object, *simDataObj*, into its own subplot. The subplot is plotted by calling the function handle, *fcnHandleValue*, with input arguments *simDataObj*, *xArgsValue*, and *yArgsValue*.

sbiosubplot(*simDataObj*, *fcnHandleValue*, *xArgsValue*, *yArgsValue*, *showLegendValue*) plots each simulation run for the SimBiology data object, *simDataObj*, into its own subplot. The subplot is plotted by calling the function handle, *fcnHandleValue*, with input arguments *simDataObj*, *xArgsValue*, and *yArgsValue*. *showLegendValue* indicates if a legend is shown in the plot. *showLegendValue* can be either true or false. By default, *showLegendValue* is false.

Examples

This example shows how to plot data from an ensemble run without interpolation.

```
% Load the radiodecay model.
        sbioloadproject('radiodecay.sbproj','m1');

        % Configure the model to run with the stochastic solver.
        cs = getconfigset(m1, 'active');
        set(cs, 'SolverType', 'ssa');
        set(cs.SolverOptions, 'LogDecimation', 100);

        % Run an ensemble simulation and view the results.
        simDataObj = sbioensemblerun(m1, 10, 'linear');
        sbiosubplot(simDataObj);
```

See Also   sbioplot

sbiotrellis

Purpose   Plot simulation results in trellis plot

Syntax
```
trellisplot = sbiotrellis(Dataset, GroupCol, XCol, YCol)
trellisplot = sbiotrellis(Dataset, GroupCol, XCol, YCol, ...)
trellisplot = sbiotrellis(Dataset, fcnHandleValue, GroupCol,
    XCol, YCol)
trellisplot = sbiotrellis(simDataObj, fcnHandleValue, XCol,
    YCol)
```

Arguments

trellisplot	Object returned by sbiotrellis. Use trellisplot together with the plot method to overlay trellis plots. See "Description" on page 2-112 for information about the plot method.
Dataset	dataset containing grouped data to plot.
GroupCol	Column containing groups.
Xcol	Data column to plot on x-axis.
YCol	Data column to plot on y-axis.
fcnHandleValue	Function handle. If '' (empty), default is @plot.
simDataObj	SimBiology data object.

Description   trellisplot = sbiotrellis(*Dataset*, *GroupCol*, *XCol*, *YCol*) plots each group in *Dataset* by the data column *GroupCol* into its own subplot. The data defined by column *XCol* is plotted against the data defined by column *YCol*.

trellisplot = sbiotrellis(*Dataset*, *GroupCol*, *XCol*, *YCol*, ...) takes optional property/value pairs that are supported by the plot command. Refer to plot in the MATLAB Reference documentation for more information on available properties.

trellisplot = sbiotrellis(*Dataset*, *fcnHandleValue*, *GroupCol*, *XCol*, *YCol*) plots each group in *Dataset* as defined by the data column *GroupCol* into its own subplot. sbiotrellis creates the subplot by calling the function handle, *fcnHandleValue*, with input arguments defined by the *Dataset* columns *XCol* and *YCol*.

trellisplot = sbiotrellis(*simDataObj*, *fcnHandleValue*, *XCol*, *YCol*) plots each group in the SimData object (*simdataObj*) into its own subplot. sbiotrellis creates the subplot by calling the function handle, *fcnHandleValue* with input arguments defined by the columns *XCol* and *YCol*.

Use the plot method to overlay a SimData object or a dataset on an existing sbiotrellis plot. The command, plot(trellisplot, ...) adds a plot to the trellis plot defined by the sbiotrellis object, trellisplot. The SimData or dataset object that is being plotted must have the same number of elements/groups as the trellis plot. The plot method has the same input arguments as sbiotrellis. For an example, see steps 3 and 4 in "Performing Population Fitting" in the SimBiology User's Guide.

See Also   sbioplot, sbiosubplot

sbiounit

Purpose   Create user-defined unit

Syntax

```
unitObject = sbiounit('NameValue')
unitObject = sbiounit('NameValue', 'CompositionValue')
unitObject = sbiounit('NameValue','CompositionValue',
    MultiplierValue)
unitObject = sbiounit('NameValue','CompositionValue',
    MultiplierValue,OffsetValue)
unitObject = sbiounit('NameValue','CompositionValue',
    ...'PropertyName', PropertyValue...)
```

Arguments

NameValue	Name of the user-defined unit. *NameValue* must begin with characters and can contain characters, underscores, or numbers. *NameValue* can be any valid MATLAB variable name.
CompositionValue	Shows the combination of base and derived units that defines the unit *NameValue*. For example molarity is mole/liter. Base units are the set of units used to define all unit quantity equations. Derived units are defined using base units or mixtures of base and derived units.
MultiplierValue	Numerical value that defines the relationship between the user-defined unit *NameValue* and the base unit as a product of the *MultiplierValue* and the base unit. For example, 1 mole is 6.0221e23*molecule. The *MultiplierValue* is 6.0221e23.
OffsetValue	Numerical value by which the unit composition is modified. For example, Celsius = (5/9)*(Fahrenheit-32); Fahrenheit is Composition; *MultiplierValue* is 5/9 and *OffsetValue* is 32.

PropertyName	Name of the unit object property, for example, 'Notes'.
PropertyValue	Value of the unit object property, for example, '*New unit for GPCR model*'.

Description

unitObject = sbiounit('*NameValue*') constructs a SimBiology unit object named *NameValue*. Valid names must begin with a letter, and be followed by letters, underscores, or numbers.

unitObject = sbiounit('*NameValue*', '*CompositionValue*') allows you to specify the name and the composition of the unit.

unitObject = sbiounit('*NameValue*','*CompositionValue*',*MultiplierValue*) creates a unit with the name *NameValue* where the unit is defined as MultiplierValue*CompositionValue.

unitObject = sbiounit('*NameValue*','*CompositionValue*',*MultiplierValue*,*OffsetValue*) creates a unit with the specified offset.

unitObject = sbiounit('*NameValue*','*CompositionValue*',...'*PropertyName*', *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

In order to use *unitObject*, you must add it to the user-defined library with the sbioaddtolibrary function. To get the unit object into the user-defined library, use the following command:

    sbioaddtolibrary(*unitObject*);

You can view additional *unitObject* properties with the get command. You can modify additional properties with the set command. For more information about unit object properties and methods, see Unit object.

sbiounit

Use the `sbiowhos` function to list the units available in the user-defined library.

Examples   This example shows you how to create a user-defined unit, add it to the user-defined library, and query the library.

1 Create units for the rate constants of a first-order and a second-order reaction.

```
unitObj1 = sbiounit('firstconstant', '1/second', 1);
unitObj2 = sbiounit('secondconstant', '1/molarity*second', 1);
```

2 Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj1);
sbioaddtolibrary(unitObj2);
```

3 Query the user-defined library in the root object.

```
rootObj = sbioroot;

rootObj.UserDefinedLibrary.Units

SimBiology UserDefined Units
```

Index:	Name:	Composition:	Multiplier:	Offset:
1	firstconstant	1/second	1.000000	0.000000
2	secondconstant	1/molarity*second	1.000000	0.000000

Alternatively, use the `sbiowhos` command.

```
sbiowhos -userdefined -unit


SimBiology UserDefined Units
```

Index:	Name:	Composition:	Multiplier:	Offset:
1	firstconstant	1/second	1.000000	0.000000
2	secondconstant	1/molarity*second	1.000000	0.000000

See Also   sbioaddtolibrary, sbioshowunits, sbiounitprefix, sbiowhos

sbiounitcalculator

Purpose   Convert value between units

Syntax   *result* = sbiounitcalculator('*fromUnits*', '*toUnits*', *Value*)

Description   *result* = sbiounitcalculator('*fromUnits*', '*toUnits*', *Value*) converts the value, *Value*, which is defined in the units, *fromUnits*, to the value, *result*, which is defined in the units, *toUnits*.

Example   result = sbiounitcalculator('mile/hour','meter/second',1)

See Also   sbioshowunits

Purpose   Create user-defined unit prefix

Syntax
unitprefixObject = sbiounitprefix('*NameValue*')
unitprefixObject = sbiounitprefix('*NameValue*',
   '*ExponentValue*')
unitprefixObject = sbiounitprefix('*NameValue*',
   ...'*PropertyName*', *PropertyValue* ...)

Arguments

NameValue	Name of the user-defined unit prefix. *NameValue* must begin with characters and can contain characters, underscores, or numbers. *NameValue* can be any valid MATLAB variable name.
ExponentValue	Shows the value of 10^Exponent that defines the relationship of the unit *Name* to the base unit. For example, for the unit picomole, Exponent is 12.
PropertyName	Name of the unit prefix object property. For example, 'Notes'.
PropertyValue	Value of the unit prefix object property. For example, *'New unitprefix for GPCR model'*.

Description   *unitprefixObject* = sbiounitprefix('*NameValue*') constructs a SimBiology unit prefix object with the name *NameValue*. Valid names must begin with a letter, and be followed by letters, underscores, or numbers.

unitprefixObject = sbiounitprefix('*NameValue*', '*ExponentValue*') creates a unit-prefix object with a multiplicative factor of 10^'*ExponentValue*'.

unitprefixObject = sbiounitprefix('*NameValue*', ...'*PropertyName*', *PropertyValue* ...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

sbiounitprefix

In order to use *unitprefixObject*, you must add it to the user-defined library with the sbioaddtolibrary function. To get the unit-prefix object into the user-defined library, use the following command:

```
sbioaddtolibrary(unitprefixObject);
```

You can view additional *unitprefixObject* properties with the get command. You can modify additional properties with the set command.

Use the sbioshowunitprefixes function to list the units available in the user-defined library.

Examples   This example shows how to create a user-defined unit prefix, add it to the user-defined library, and query the library.

1 Create a unit prefix.

```
unitprefixObj1 = sbiounitprefix('peta', 15);
```

2 Add the unit prefix to the user-defined library.

```
sbioaddtolibrary(unitprefixObj1);
```

3 Query the user-defined library in the root object.

```
rootObj = sbioroot;

rootObj.UserDefinedLibrary.UnitPrefixes

Unit Prefix Array

   Index:  Library:     Name:            Exponent:

    1       UserDefined  peta               15
```

Alternatively, use the sbiowhos command.

```
sbiowhos -userdefined -unitprefix
```

```
SimBiology UserDefined Unit Prefixes

Index:  Name:         Multiplier:
1       peta          1.000000e+015
```

See Also   sbioaddtolibrary, sbioshowunits, sbiounit, sbiowhos

sbiounregisterunit

Purpose Remove user-defined unit from root and library

> **Note** sbiounregisterunit has been removed and produces an error. Use sbioremovefromlibrary instead.

Syntax sbiounregisterunit('*Name*')

Description sbiounregisterunit('*Name*') removes the user-defined unit with the name *Name* from the user-defined library. You cannot remove a unit from the built-in library. If *Name* is a user-defined unit, then it is removed from the UserDefinedUnits vector on the SimBiology root object and also from the user library. Once unregistered, this unit is not available in future MATLAB sessions. You can list the available units and find information on whether the unit is built in or user defined using sbiowhos or sbioshowunits.

See Also sbioremovefromlibrary, sbioshowunits, sbiounit, sbiowhos

Purpose   Remove user-defined unit prefix from root and library

> **Note** sbiounregisterunitprefix has been removed and produces an
> error. Use sbioremovefromlibrary instead.

Syntax   sbiounregisterunitprefix('*Name*')

Description sbiounregisterunitprefix('*Name*') removes the user-defined unit prefix
with the name *Name* from the user-defined library. You cannot remove a
unit prefix from the built-in library. If *Name* is a user-defined unit prefix,
it is removed from the UserDefinedUnits vector on the SimBiology root
object and also from the user library. Once unregistered, this unit prefix
is not available in future MATLAB sessions. You can list the available
unit prefixes and find information on whether the unit prefix is built in
or user defined using sbiowhos or sbioshowunitprefixes.

See Also   sbioremovefromlibrary, sbioroot, sbioshowunitprefixes,
sbiounitprefix, sbiowhos

sbioupdate

Purpose   Update SimBiology model version

Syntax
```
modelsObj = sbioupdate(modelObj)
simdataObj = sbioupdate(tsObj)
```

Arguments

modelsObj	sbioupdate output. Contains an array of model objects that includes the top-level model object and a model object for each previously existing submodel.
modelObj	Model object with submodels to be converted into separate model objects.
simdataObj	sbioupdate output. Contains a SimData object converted from previous time series object.
tsObj	Time series object to be converted to a SimData object. Can be a 1-by-n cell array of time series objects.

Description   *modelsObj* = sbioupdate(*modelObj*) converts a top level SimBiology model object (*modelObj*) that has submodels into an array of SimBiology model objects which do not have any submodels.

There is one model for the top model and one for each of the submodels. Each model created, has a copy of all the parameters used by the model, including those that belonged to the parent model. Updating deletes any unused parameters in the parent model.

Each model created from the previously existing submodel has empty StatesToLog, SpeciesInputFactors, ParameterInputFactors, and SpeciesOutputs property values.

simdataObj = sbioupdate(*tsObj*) converts a time series object (*tsObj*) obtained from simulation of a SimBiology model into a SimData object. If *tsObj* is a cell array of time series objects, then *simdataObj* is an array of SimData objects, having one element for each of the time series objects in *tsObj*.

Purpose	Construct variant object

Syntax

variantObj = sbiovariant('*NameValue*')
variantObj = sbiovariant('*NameValue*', '*ContentValue*')
variantObj = sbiovariant(...'*PropertyName*', *PropertyValue*...)

Arguments

modelObj	Specify the model object to which you want add a variant.
variantObj	Variant object to create and add to the model object.
NameValue	Name of the variant object. *NameValue* is assigned to the Name property of the variant object.

Description

variantObj = sbiovariant('*NameValue*') creates a SimBiology variant object (*variantObj*) with the name *NameValue*. The variant object Parent property is assigned [] (empty).

variantObj = sbiovariant('*NameValue*', '*ContentValue*') creates a SimBiology variant object (*variantObj*) with the Content property set to *ContentValue*.

To add a variant to a model use the copyobj method. A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see Variant object.

variantObj = sbiovariant(...'*PropertyName*', *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

View properties for a variant object with the get command, and modify properties for a variant object with the set command.

sbiovariant

> **Note** Remember to use the addcontent method instead of using the set method on the Content property because the set method replaces the data in the Content property, whereas addcontent appends the data.

Method Summary

addcontent (variant)	Append content to variant object
commit (variant)	Commit variant contents to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmcontent (variant)	Remove contents from variant object
set (any object)	Set object properties
verify (model, variant)	Validate and verify SimBiology model

Property Summary

Active	Indicate object in use during simulation
Annotation	Store link to URL or file
Content	Contents of variant object
Name	Specify name of object
Notes	HTML text describing SimBiology object

Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples

1 Create a variant object.

```
variantObj = sbiovariant('p1');
```

2 Add content to the variant object that varies the InitialAmount property of a species named A.

```
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

See Also addvariant, copyobj, getvariant

sbiowhos

Purpose   Show contents of project file, library file, or SimBiology root object

Syntax
```
sbiowhos flag
sbiowhos ('flag')
sbiowhos flag1 flag2...
sbiowhos FileName
```

Description   sbiowhos shows contents of the SimBiology root object. This includes the built-in and user-defined abstract kinetic laws, units, and unit prefixes.

sbiowhos flag shows specific information about the SimBiology root object as defined by flag. Valid flags are described in this table.

Flag	Description
-builtin	Built-in abstract kinetic laws, units, and unit prefixes
-data	Data saved in file
-kineticlaw	Built-in and user-defined abstract kinetic laws
-unit	Built-in and user-defined units
-unitprefix	Built-in and user-defined unit prefixes
-userdefined	User-defined abstract kinetic laws, units, and unit prefixes

You can also specify the functional form sbiowhos ('flag').

sbiowhos flag1 flag2... shows information about the SimBiology root object as defined by flag1, flag2,... .

sbiowhos FileName shows the contents of the SimBiology project or library defined by Name.

Examples

```
% Show contents of the SimBiology root object
  sbiowhos

% Show abstract kinetic laws on the SimBiology root object
  sbiowhos -kineticlaw

% Show the builtin units of the SimBiology root object.
  sbiowhos -builtin -unit

% Show all contents of project file.
  sbiowhos myprojectfile

% Show abstract kinetic laws from a library file.
  sbiowhos -kineticlaw mylibraryfile

% Show all contents of multiple files.
  sbiowhos myfile1 myfile2
```

See Also   MATLAB function whos

sbmlexport

Purpose   Export SimBiology model to SBML file

Syntax
```
sbmlexport(modelObj)
sbmlexport(modelObj, 'FileName')
```

Arguments

modelObj	Model object. Enter a variable name for a model object.
FileName	XML file with a Systems Biology Markup Language (SBML) format. Enter either a file name or a path and file name supported by your operating system. If the file name does not have the extension .xml, then .xml is appended to end of the file name.

Description   sbmlexport(modelObj) exports a SimBiology model object (modelObj) to a file with a Systems Biology Markup Language (SBML) Level 2 Version 1 format. The default file extension is .xml and the file name matches the model name.

sbmlexport(modelObj, 'FileName') exports a SimBiology model object (modelObj) to an SBML file named *FileName* . The default file extension is .xml.

A SimBiology model can also be written to a SimBiology project with the sbiosaveproject function to save features not supported by SBML.

See "SBML Support" in the SimBiology Getting Started Guide for more information.

Example   Export a model (modelObj) to a file (gene_regulation.xml) in the current working directory.

```
sbmlexport(modelObj,'gene_regulation.xml');
```

Reference   Finney, A., Hucka, M., (2003), *Systems Biology Markup Language (SBML) Level 2: Structures and facilities for model definitions.* Accessed from SBML.org

See Also   sbiomodel, sbiosaveproject, sbmlimport

sbmlimport

Purpose	Import SBML-formatted file
Syntax	modelObj = sbmlimport('*FileName*')

Arguments

FileName	XML file with a Systems Biology Markup Language (SBML) format. Enter either a file name or a path and file name supported by your operating system.

Description   modelObj = sbmlimport('*FileName*') imports an SBML-formatted file with name *FileName* into MATLAB and creates a model object modelObj. FileName extensions can be .sbml or .xml. The modelObj properties can be viewed with the get command. modelObj properties can be modified with the set command. At the command line, help for modelObj functions can be returned with the sbiohelp command. sbmlimport supports SBML Levels 1 and Level 2 Version 1.

See "SBML Support" in the SimBiology Getting Started Guide for more information.

Example   sbmlobj = sbmlimport('oscillator.xml');

Reference   Finney, A., Hucka, M., (2003), *Systems Biology Markup Language (SBML) Level 2: Structures and facilities for model definitions.* Accessed from SBML.org.

See Also   get, sbmlexport, sbiosimulate, set

Purpose   Open SimBiology modeling and simulation GUI

Syntax   simbiology
                 simbiology(*modelObj*)

Arguments

modelObj	Model object or an array of model objects. Enter the variable name for a top-level SimBiology model object. If you enter an array of model objects, the SimBiology desktop opens with each model object in a separate model session.

Description   simbiology opens the SimBiology desktop, which lets you do the following:

- Build a SimBiology model by representing reaction pathways and entering kinetic data for the reactions.

- Import or export SimBiology models to and from the MATLAB workspace or from a Systems Biology Markup Language (SBML) file.

- Modify an existing SimBiology model.

- Simulate a SimBiology model through individual or ensemble runs.

- View results from the simulation.

- Perform analysis tasks such as sensitivity analysis, parameter and species scans, and calculate conserved moieties.

- Create and/or modify user-defined units and unit prefixes.

- Create and/or modify user-defined kinetic laws.

simbiology(*modelObj*) opens the SimBiology desktop with a top-level SimBiology model object (*modelObj*). If there is a project open in the SimBiology desktop, this command adds the model (*modelObj*) to the project. A top-level SimBiology model object has its property Parent set to the SimBiology root object. Thus, querying sbioroot at the command

line shows you all models in the MATLAB workspace, including the models available in the desktop. Any changes you make to the model at the command line are reflected in the desktop because both are pointing to the same model object in the Root object.

Note The sbioreset command removes all models from the root object and therefore this command also removes all models from the SimBiology desktop.

Examples   Create a SimBiology model in the MATLAB workspace, and then open the GUI with the model.

```
modelObj = sbiomodel('cell');
simbiology(modelObj)
```

See Also   sbioroot

3

Method Reference

Objects

Abstract Kinetic Laws

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Compartments

addcompartment (model, compartment)	Create compartment object
addspecies (compartment)	Create species object and add to compartment object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties

Configuration Sets

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
set (any object)	Set object properties

Events

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Kinetic Laws

addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getparameters (kineticlaw)	Get specific parameters in kinetic law object
getspecies (kineticlaw)	Get specific species in kinetic law object
set (any object)	Set object properties
setparameter (kineticlaw)	Specify specific parameters in kinetic law object
setspecies (kineticlaw)	Specify species in kinetic law object

Models

addcompartment (model, compartment)	Create compartment object
addconfigset (model)	Create configuration set object and add to model object
addevent (model)	Add event object to model object
addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
addreaction (model)	Create reaction object and add to model object
addrule (model)	Create rule object and add to model object
addvariant (model)	Add variant to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getadjacencymatrix (model)	Get adjacency matrix from model object
getconfigset (model)	Get configuration set object from model object
getstoichmatrix (model)	Get stoichiometry matrix from model object
getvariant (model)	Get variant from model
removeconfigset (model)	Remove configuration set from model
removevariant (model)	Remove variant from model
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties

setactiveconfigset (model)	Set active configuration set for model object
verify (model, variant)	Validate and verify SimBiology model

Parameters

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

Pharmacokinetic Modeling

addCompartment (PKModelDesign)	Add compartment to `PKModelDesign` object
construct (PKModelDesign)	Construct SimBiology model from `PKModelDesign` object
get (any object)	Get object properties
PKCompartment object	Used by `PKModelDesign` to create SimBiology model
PKData object	Define roles of data set columns
PKModelDesign object	Helper object to construct pharmacokinetic model
PKModelMap object	Define SimBiology model components roles
set (any object)	Set object properties

Reactions

addkineticlaw (reaction)	Create kinetic law object and add to reaction object
addproduct (reaction)	Add product species object to reaction object
addreactant (reaction)	Add species object as reactant to reaction object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmproduct (reaction)	Remove species object from reaction object products
rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

Root

copyobj (any object)	Copy SimBiology object and its children
get (any object)	Get object properties
reset (root)	Delete all model objects from root object
set (any object)	Set object properties

Rules

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

SimData

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getdata (SimData)	Get data from `SimData` object array
getsensmatrix (SimData)	Get 3-D sensitivity matrix from SimData array
resample (SimData)	Resample SimData object array onto new time vector
select (SimData)	Select data from SimData object
selectbyname (SimData)	Select data by name from SimData object array
set (any object)	Set object properties

Species

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

Units and Unit Prefixes

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Variants

addcontent (variant)	Append content to variant object
commit (variant)	Commit variant contents to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object

display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmcontent (variant)	Remove contents from variant object
set (any object)	Set object properties
verify (model, variant)	Validate and verify SimBiology model

Using Object Methods

Command-line syntax for using methods with SimBiology objects

Constructing (Creating) Objects

Create an object that is not referenced by a model using the constructor
functions sbioabstractkineticlaw, sbiomodel, sbioparameter,
sbioreaction, sbioroot, sbiorule, and sbiospecies.

```
ObjectName = ConstructorFunction(RequiredParameters,...
                                 'PropertyName',  PropertyValue')
```

To create objects referenced by a model, use the model object methods
addconfigset, addmodel, addparameter, addreaction, addrule, and
addspecies.

```
ObjectName = ModelName.Method(Arguments)
```

To create objects referenced by a reaction, use the reaction object methods
addkineticlaw, addparameter, addproduct, and addreactant.

```
ObjectName = ReactionName.Method(Arguments)
```

Note that ObjectName is not a copy of the object, but a pointer to the created
object.

Using Object Methods

Using MATLAB function notation:

```
MethodName(ObjectName, arguments, ...)
```

Using object dot notation:

```
ObjectName.MethodName(arguments, ...)
```

Help for Objects, Methods, and Properties

Display information for SimBiology object methods and properties in the MATLAB Command Window.

`help sbio`	Display a list of functions and methods.
`help FunctionName`	Display function information.
`sbiohelp('MethodName')`	Display method information.
`sbiohelp('PropertyName')`	Display property information.

Methods — Alphabetical List

The object that the methods apply to are listed in parenthesis after the method name.

AbstractKineticLaw object

Purpose Kinetic law information in library

Description The abstract kinetic law object represents an *abstract kinetic law*, which provides a mechanism for applying a rate law to multiple reactions. The information in this object acts as a mapping template for the reaction rate. The abstract kinetic law defines a mathematical relationship that defines the rate at which reactant species are produced and product species are consumed in the reaction. The expression is shown in the property Expression. The species variables are defined in the SpeciesVariables property, and the parameter variables are defined in the ParameterVariables property of the abstract kinetic law object. For an explanation of how the abstract kinetic law object relates to the kinetic law object, see KineticLaw object.

Define your own abstract kinetic law and add it to the abstract kinetic law library with the sbioaddtolibrary function. You can then use the abstract kinetic law when constructing a kinetic law object with the method addkineticlaw. To retrieve an abstract kinetic law object from the user-defined library, use the command get(sbioroot, 'UserDefinedKineticLaws').

See "Property Summary" on page 4-3 for links to abstract kinetic law object property reference pages.

Properties define the characteristics of an object. For example, an abstract kinetic law object includes properties for the expression, the name of the law, parameter variables, and species variables. Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

Constructor Summary

sbioabstractkineticlaw	Construct abstract kinetic law object

Method Summary	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	set (any object)	Set object properties

Property Summary	Annotation	Store link to URL or file
	Expression	Expression to determine reaction rate equation
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	ParameterVariables	Parameters in abstract kinetic law
	Parent	Indicate parent object
	SpeciesVariables	Species in abstract kinetic law
	Tag	Specify label for SimBiology object
	Type	Display top-level SimBiology object type
	UserData	Specify data to associate with object

See Also   Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object

addcompartment (model, compartment)

Purpose

Create compartment object

Syntax

compartmentObj = addcompartment(*modelObj*, '*NameValue*')
compartmentObj = addcompartment(*owningCompObj*, '*NameValue*')
compartmentObj = addcompartment(*modelObj*, '*NameValue*',
   CapacityValue)
compartmentObj = addcompartment(...'*PropertyName*',
PropertyValue...)

Arguments

modelObj	Model object.
owningCompObj	Compartment object that contains the newly created compartment object.
NameValue	Name for a compartment object. Enter a character string unique to the model object. For information on naming compartments, see Name.
CapacityValue	Capacity value for the compartment object. Enter double. Positive real number, default = 1.
PropertyName	Enter the name of a valid property. Valid property names are listed in "Property Summary" on page 4-6.
PropertyValue	Enter the value for the property specified in *PropertyName*. Valid property values are listed on each property reference page.

Description

compartmentObj = addcompartment(*modelObj*, '*NameValue*') creates a compartment object and returns the compartment object (*compartmentObj*). In the compartment object, this method assigns a value (*NameValue*) to the property Name, and assigns the model object (*modelObj*) to the property Parent. In the model object, this method assigns the compartment object to the property Compartments.

compartmentObj = addcompartment(*owningCompObj*, '*NameValue*') in addition to the above, adds the newly created compartment within a compartment object (*owningCompObj*), and assigns this compartment object (*owningCompObj*) to the Owner property of the newly created compartment object (*compartmentObj*). The parent model is the model that contains the owning compartment (*owningCompObj*).

compartmentObj = addcompartment(*modelObj*, '*NameValue*', *CapacityValue*), in addition to the above, this method assigns capacity *(CapacityValue)* for the compartment.

If you define a reaction within a model object (modelObj) that does not contain any compartments, the process of adding a reaction generates a default compartment object and assigns the reaction species to the compartment. If there is more than one compartment, you must specify which compartment the species should be assigned to using the format *CompartmentName.SpeciesName*.

View properties for a compartment object with the get command, and modify properties for a compartment object with the set command. You can view a summary table of compartment objects in a model (modelObj) with get(modelObj, 'Compartments') or the properties of the first compartment with get(modelObj.Compartments(1)).

compartmentObj = addcompartment(...'*PropertyName*', *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs). "Property Summary" on page 4-6 lists the properties. The Owner property is one exception; you cannot set the Owner property in the addcompartment syntax because, addcompartment requires the owning model or compartment to be specified as the first argument and uses this information to set the Owner property. After adding a compartment, you can change the owner using the function set.

addcompartment (model, compartment)

Method Summary	Methods for compartment objects	
	addcompartment (model, compartment)	Create compartment object
	addspecies (compartment)	Create species object and add to compartment object
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	rename (compartment, parameter, species)	Rename object and update expressions
	reorder (model, compartment)	Reorder component lists
	set (any object)	Set object properties
Property Summary	Properties for compartment objects	
	Annotation	Store link to URL or file
	Capacity	Compartment capacity
	CapacityUnits	Compartment capacity units
	Compartments	Array of compartments in model or compartment
	ConstantCapacity	Specify variable or constant compartment capacity
	Name	Specify name of object
	Notes	HTML text describing SimBiology object

addcompartment (model, compartment)

Owner	Owning compartment
Parent	Indicate parent object
Species	Array of species in compartment object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples

1 Create a model object (`modelObj`).

```
modelObj = sbiomodel('cell');
```

2 Add two compartments to the model object.

```
compartmentObj1 = addcompartment(modelObj, 'nucleus');
compartmentObj2 = addcompartment(modelObj, 'mitochondrion');
```

3 Add a compartment to one of the compartment objects.

```
compartmentObj3 = addcompartment(compartmentObj2, 'matrix');
```

4 Display the Compartments property in the model object.

```
get(modelObj, 'Compartments')

SimBiology Compartment Array

   Index:    Name:           Capacity:    CapacityUnits:
   1         nucleus         1
   2         mitochondrion   1
   3         matrix          1
```

addcompartment (model, compartment)

5 Display the Compartments property in the compartment object.

```
get(compartmentObj2, 'Compartments')

SimBiology Compartment - matrix

   Compartment Components:
     Capacity:         1
     CapacityUnits:
     Compartments:     0
     ConstantCapacity: true
     Owner:            mitochondrion
     Species:          0
```

See Also   addproduct, addreactant, addreaction, addspecies, get, set

addCompartment (PKModelDesign)

Purpose	Add compartment to `PKModelDesign` object
Syntax	*PKCompartmentObj* = addCompartment(*PKModelDesignObj*, *Name*, '*DosingType*','*EliminationType*', *HasResponseVariable*)

Arguments

Input Arguments

PKModelDesignObj	The `PKModelDesign` object to which you want to add a compartment
Name	Name of the compartment object that is constructed
DosingType	Specifies the mechanism for drug absorption. Valid options are `'Bolus'`, `'Infusion'`, `'ZeroOrder'`, `'FirstOrder'` and `''`. You can only specify one compartment in a model with a dosing type other than `''` (empty).For more information on *DosingType*, see "About Dosing Types" in the SimBiology User's Guide.
EliminationType	Specifies the mechanism for drug elimination. Valid options are `'Linear'`, `'Linear-Clearance'`, `'Enzymatic'`, and `''`. For more information on *EliminationType*, see "About Elimination Types" in the SimBiology User's Guide.
HasResponseVariable	Logical indicating if the drug concentration in this compartment is reported. Only one compartment can have this set to `true`.

Output Arguments

PKCompartmentObj	`PKCompartment` object

addCompartment (PKModelDesign)

Description	*PKCompartmentObj* = addCompartment(*PKModelDesignObj*, *Name*, '*DosingType*','*EliminationType*', *HasResponseVariable*) adds a compartment with the specified name, dosing and elimination type, and specifies whether the compartment contains a measured (or observed) response.	

Method Summary	get (any object)	Get object properties
	set (any object)	Set object properties

Property Summary	DosingType	Drug dosing type in compartment
	EliminationType	Drug elimination type from compartment
	HasResponseVariable	Compartment drug concentration reported
	Name	Specify name of object

See Also	"Creating PK Models at the Command Line" in the SimBiology User's Guide, PKCompartment object PKModelDesign object

Purpose   Create configuration set object and add to model object

Syntax   *configsetObj* = addconfigset(*modelObj*, '*NameValue*')
configsetObj = addconfigset(..., '*PropertyName*',
PropertyValue, ...)

Arguments

modelObj	Model object. Enter a variable name.
NameValue	Descriptive name for a configuration set object. Reserved words 'active' and 'default' are not allowed.
configsetObj	Configuration set object.

Description   *configsetObj* = addconfigset(*modelObj*, '*NameValue*') creates a configuration set object and returns to *configsetObj*.

In the configuration set object, this method assigns a value (*NameValue*) to the property Name.

configsetObj = addconfigset(..., '*PropertyName*', *PropertyValue*, ...) constructs a configuration set object, *configsetObj*, and configures *configsetObj* with property value pairs. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs). The *configsetObj* properties are listed in "Property Summary" on page 4-12.

A configuration set stores simulation specific information. A model object can contain multiple configuration sets, with one being active at any given time. The active configuration set contains the settings that are used during a simulation. *configsetObj* is not automatically set to active. Use the function setactiveconfigset to define the active configset for modelObj.

Use the method copyobj to copy a configset object and add it to the modelObj.

addconfigset (model)

You can additionally view configuration set object properties with the command get. You can modify additional configuration set object properties with the command set.

Method Summary

Methods for configuration set objects

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
set (any object)	Set object properties

Property Summary

Properties for configuration set objects

Active	Indicate object in use during simulation
CompileOptions	Dimensional analysis and unit conversion options
Name	Specify name of object
Notes	HTML text describing SimBiology object
RuntimeOptions	Options for logged species
SensitivityAnalysisOptions	Specify sensitivity analysis options
SolverOptions	Specify model solver options
SolverType	Select solver type for simulation
StopTime	Set stop time for simulation
StopTimeType	Specify type of stop time for simulation

TimeUnits	Show stop time units for simulation
Type	Display top-level SimBiology object type

Examples

1 Create a model object by reading the file oscillator.xml and add a configuration set that simulates the model for 3000 seconds.

```
modelObj  = sbmlimport('oscillator');
configsetObj = addconfigset(modelObj, 'myset');
```

2 Configure the configsetObj StopTime to 3000.

```
set(configsetObj, 'StopTime', 3000)
get(configsetObj)
```

```
              Active: 0
      CompileOptions: [1x1 SimBiology.CompileOptions]
                Name: 'myset'
               Notes: ''
      RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
       SolverOptions: [1x1 SimBiology.ODESolverOptions]
          SolverType: 'ode15s'
            StopTime: 3000
        StopTimeType: 'simulationTime'
           TimeUnits: 'second'
                Type: 'configset'
```

3 Set the new configset to be active, simulate the model using the new configset, and plot the result.

```
 setactiveconfigset(modelObj, configsetObj);
 [t,x] = sbiosimulate(modelObj);
plot (t,x)
```

See Also   get, getconfigset, removeconfigset, set, setactiveconfigset

addcontent (variant)

Purpose	Append content to variant object
Syntax	addcontent(*variantObj*, *contents*) addcontent(*variantObj1*, *variantObj2*)

Arguments

variantObj	Specify the variant object to which you want to append data. The Content property is modified to add the new data.
contents	Specify the data you want to add to a variant object. Contents can either be a cell array or an array of cell arrays. A valid cell array should have the form {'*Type*', '*Name*', '*PropertyName*', *PropertyValue*}, where *PropertyValue* is the new value to be applied for the *PropertyName*. Valid *Type*, *Name*, and *PropertyName* values are as follows.

'Type'	*'Name'*	*'PropertyName'*
'species'	Name of the species. If there are multiple species in the model with the same name, specify the species as [compartmentName.speciesName], where compartmentName is the name of the compartment containing the species.	'InitialAmount'
'parameter'	If the parameter scope is a model, specify the parameter name. If the parameter scope is a kinetic law, specify [reactionName.parameterName].	'Value'
'compartment'	Name of the compartment.	'Capacity'

addcontent (variant)

Description   addcontent(*variantObj*, *contents*) adds the data stored in the variable *contents* to the variant object (*variantObj*).

addcontent(*variantObj1*, *variantObj2*) appends the data in the Content property of the variant object *variantObj2* to the Content property of variant object *variantObj1*.

Note Remember to use the addcontent method instead of using the set method on the Content property because the set method replaces the data in the Content property, whereas addcontent appends the data.

Examples   **1** Create a model containing one species.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
speciesObj = addspecies(compObj, 'A');
```

2 Add a variant object that varies the InitialAmount property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

See Also   addvariant, rmcontent, sbiovariant

addevent (model)

Purpose   Add event object to model object

Syntax   *eventObj* = addevent(*modelObj*, '*TriggerValue*',
                    '*EventFcnsValue*')
                 eventObj = addevent(...'*PropertyName*', *PropertyValue*...)

Arguments

modelObj	Model object.
TriggerValue	Required property to specify a trigger condition. Must be a MATLAB expression that evaluates to a logical value. Use the keyword 'time' to specify that an event occurs at a specific time during the simulation. See Trigger for more information.
EventFcnsValue	A string or a cell array of strings, each of which specifies an assignment of the form '*objectname = expression*', where *objectname* is the name of a valid object. Defines what occurs when the event is triggered. See EventFcns for more information.
PropertyName	Property name for an event object from "Property Summary" on page 4-17.
PropertyValue	Property value. For more information on property values, see the property reference for each property listed in "Property Summary" on page 4-17.

Description *eventObj* = addevent(*modelObj*, '*TriggerValue*', '*EventFcnsValue*')
                 creates an event object (*eventObj*) and adds the event to the
                 model (*modelObj*). In the event object, this method assigns a value
                 (*TriggerValue*) to the property TriggerCondition, assigns a value
                 (*EventFcnsValue*) to the property EventFcns, and assigns the model
                 object (*modelObj*) to the property Parent. In the model object, this
                 method appends the event object to the property Events.

When the trigger expression in the property `Trigger` changes from false to true, the assignments in `EventFcns` are executed during simulation.

For details on how events are handled during a simulation, see "Changing Model Component Values Using Events" in the SimBiology User's Guide documentation.

`eventObj = addevent(...'PropertyName', PropertyValue...)` defines optional properties. The property name and property value pairs can be any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

You can view additional object properties with the `get` command. You can modify additional object properties with the `set` command. To view events of a model object (`modelObj`), use the command `get(modelObj, 'Events')`.

Method Summary		
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	set (any object)	Set object properties

Property Summary		
	Active	Indicate object in use during simulation
	Annotation	Store link to URL or file
	EventFcns	Event expression
	Name	Specify name of object
	Notes	HTML text describing SimBiology object

Parent	Indicate parent object
Tag	Specify label for SimBiology object
Trigger	Event trigger
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples

1 Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator')
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

2 Get a list of properties for an event object.

```
get(modelObj.Events(1));
```

Or

```
get(eventObj)
```

MATLAB displays a list of event properties.

```
        Active: 1
    Annotation: ''
      EventFcns: {'OpC = 200'}
          Name: ''
         Notes: ''
        Parent: [1x1 SimBiology.Model]
           Tag: ''
       Trigger: 'time >= 5'
          Type: 'event'
      UserData: []
```

See Also   Event object

addkineticlaw (reaction)

Purpose   Create kinetic law object and add to reaction object

Syntax
kineticlawObj = addkineticlaw(*reactionObj*,
    '*KineticLawNameValue*')
kineticlawObj= addkineticlaw(..., '*PropertyName*',
*PropertyValue*, ...)

Arguments

reactionObj	Reaction object. Enter a variable name for a reaction object.
KineticLawNameValue	Property to select the type of kinetic law object to create. For built-in kinetic law, valid values are:

'Unknown', 'MassAction',
'Henri-Michaelis-Menten',
'Henri-Michaelis-Menten-Reversible',
'Hill-Kinetics', 'Iso-Uni-Uni',
'Ordered-Bi-Bi', 'Ping-Pong-Bi-Bi',
'Competitive-Inhibition',
'NonCompetitive-Inhibition', and
'UnCompetitive-Inhibition'.

Find valid *KineticLawNameValue*s by querying the SimBiology root object with the commands get(sbioroot, 'BuiltInKineticLaws'), and get(sbioroot, 'UserDefinedKineticLaws').

sbiowhos -kineticlaw lists BuiltInKineticLaws and UserDefinedKineticLaws in the SimBiology root. The root contains all BuiltInKineticLaws and all UserDefinedKineticLaws that are added using sbioaddtolibrary.

Description   *kineticlawObj* = addkineticlaw(*reactionObj*, '*KineticLawNameValue*')
creates a kinetic law object and returns the kinetic law object
(*kineticlawObj*).

In the kinetic law object, this method assigns a name
(*KineticLawNameValue*) to the property KineticLawName and assigns
the reaction object to the property Parent. In the reaction object, this
method assigns the kinetic law object to the property KineticLaw.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'a -> b');
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
parameterObj = addparameter(kineticlawObj, 'K1_forward', 0.1);
set(kineticlawObj, ParameterVariableName, 'K1_forward');
```
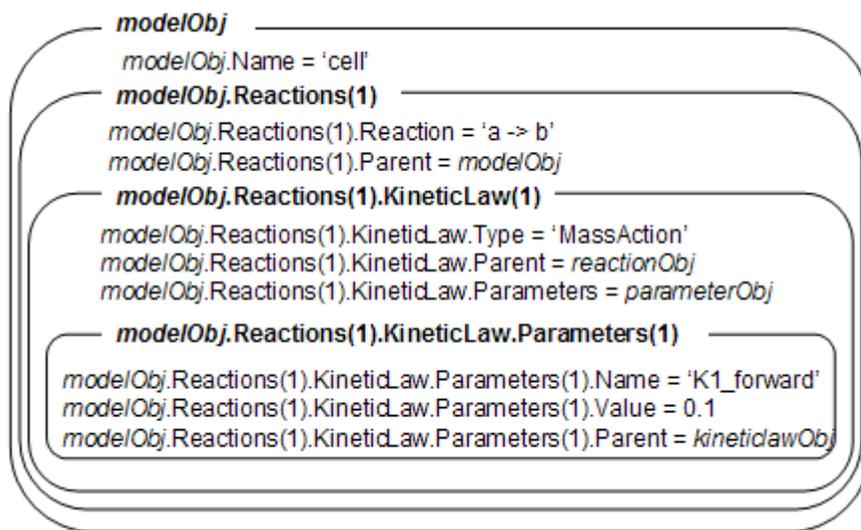

KineticLawNameValue is any valid abstract kinetic law. See "Abstract
Kinetic Law" on page 6-56 for a definition of abstract kinetic laws and
more information about how they are used to get the reaction rate
expression.

addkineticlaw (reaction)

kineticlawObj= addkineticlaw(..., '*PropertyName*', *PropertyValue*, ...) constructs a kinetic law object, *kineticlawObj*, and configures *kineticlawObj* with property value pairs. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The `kineticlawObj` properties are listed in "Property Summary" on page 4-23.

You can view additional kinetic law object properties with the `get` command. You can modify additional kinetic law object properties with the `set` command. The kinetic law used to determine the `ReactionRate` of the `Reaction` can be viewed with `get(`*reactionObj*`, 'KineticLaw')`. Remove a SimBiology kinetic law object from a SimBiology reaction object with the `delete` command.

Method Summary

Methods for kinetic law objects

addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getparameters (kineticlaw)	Get specific parameters in kinetic law object
getspecies (kineticlaw)	Get specific species in kinetic law object
set (any object)	Set object properties

| setparameter (kineticlaw) | Specify specific parameters in kinetic law object |
| setspecies (kineticlaw) | Specify species in kinetic law object |

Property Summary

Properties for kinetic law objects

Annotation	Store link to URL or file
Expression	Expression to determine reaction rate equation
KineticLawName	Name of kinetic law applied to reaction
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parameters	Array of parameter objects
ParameterVariableNames	Cell array of reaction rate parameters
ParameterVariables	Parameters in abstract kinetic law
Parent	Indicate parent object
SpeciesVariableNames	Cell array of species used in reaction rate equation
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object

Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples

Example 1

This example uses the built-in kinetic law `Henri-Michaelis-Menten`.

1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('Cell');
reactionObj = addreaction (modelObj, 'Substrate -> Product');
```

2 Define an abstract kinetic law for the reaction object and view the parameters to be set.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
get (kineticlawObj, 'Expression')

ans =
    Vm*S/(Km + S)
```

The `addkineticlaw` method adds an abstract kinetic law expression to the reaction object (*reactionObj*).

The `Henri-Michaelis-Menten` kinetic law has two parameters (`Vm` and `Km`) and one species (`S`). You need to enter values for these parameters by first creating parameter objects, and then adding the parameter objects to the kinetic law object.

3 Add parameter objects to a kinetic law object. For example, create a parameter object `parameterObj1` named `Vm_d`, another paramter `parameterObj2)` named `Km_d`, and add them to a kinetic law object (`kineticlawObj`).

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d', 'Value', 6.0);
parameterObj2 = addparameter(kineticlawObj, 'Km_d', 'Value', 1.25);
```

The `addparameter` method creates two parameter objects with concrete values that will be associated with the abstract kinetic law parameters.

4 Associate concrete kinetic law parameters with the abstract kinetic law parameters.

```
set(kineticlawObj,'ParameterVariableNames', {'Vm_d' 'Km_d'});
set(kineticlawObj,'SpeciesVariableNames', {'Substrate'});
```

This method associates the concrete parameters in the property `ParameterVariableNames` with the abstract parameters in the property `ParameterVariables` using a one-to-one mapping in the order given.

5 Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
get (reactionObj, 'ReactionRate')

ans =
    Vm_d*Substrate/(Km_d+Substrate)
```

6 Enter an initial value for the substrate and simulate.

```
modelObj.Species(1).InitialAmount = 8;
[T, X] = sbiosimulate(modelObj);
plot(T,X)
```
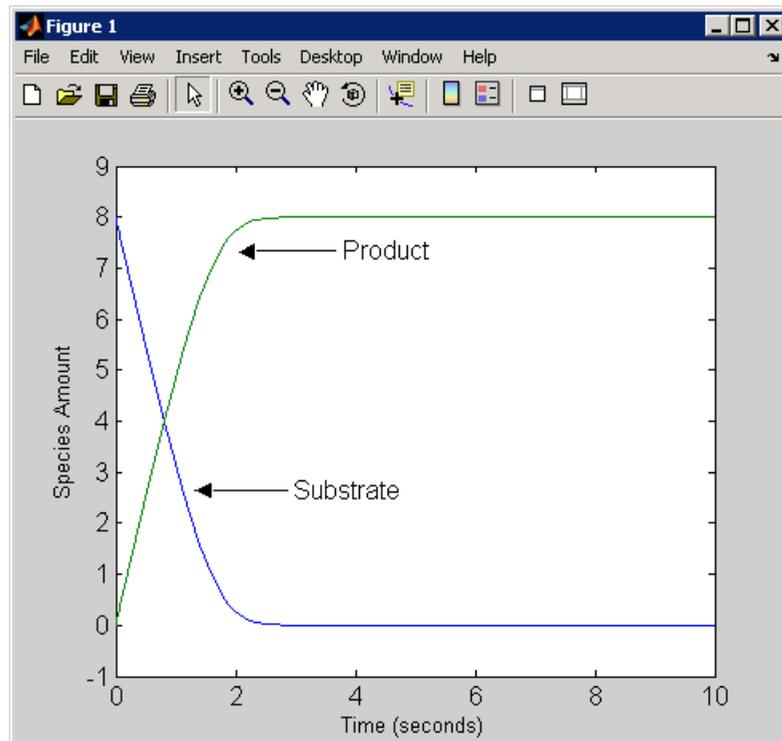
Example 2

This example uses the built-in kinetic law MassAction.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('Cell');
reactionObj = addreaction (modelObj, 'a -> b');
```

2 Define an abstract kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
get(kineticlawObj, 'Expression')
ans =
```

```
MassAction
```

Notice, the property `Expression` for an abstract kinetic law with property `Type` set to `MassAction` does not show the parameters and species in the reaction rate.

3 Assign the rate constant for the reaction.

```
parameterObj = addparameter(kineticlawObj, 'k_forward');
set (kineticlawObj, 'ParameterVariablenames', 'k_forward');
get (reactionObj, 'ReactionRate')

ans =
    k_forward*a
```

4 Enter an initial value for the substrate and simulate.

```
modelObj.Species(1).InitialAmount = 100;
[T, X] = sbiosimulate(modelObj);plot(T,X)
```

The value used for `k_forward` is the default value = `1.0`.

addkineticlaw (reaction)
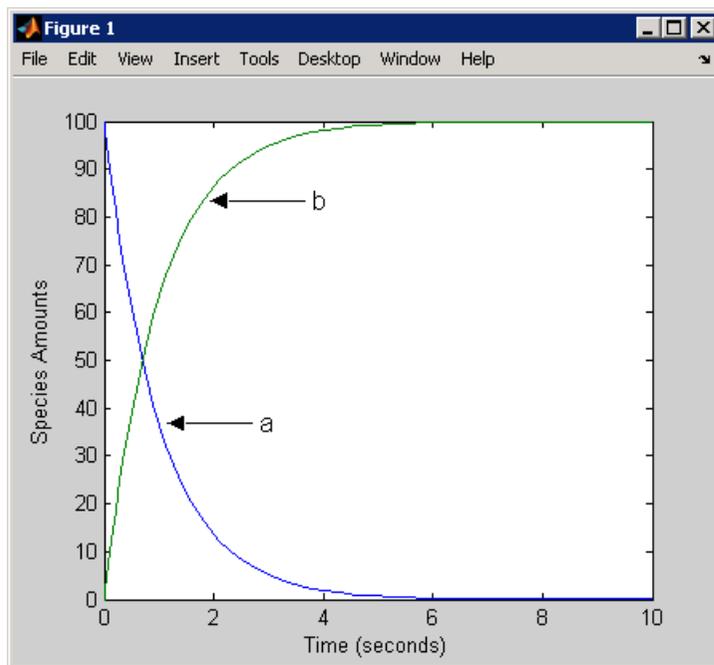
See Also   addreaction, setparameter

Purpose   Add submodel object to model object

Note addmodel produces a warning and will be removed in a future version. Submodels will not be supported in future releases. Use the function sbioupdate to convert submodels into models.

Syntax   *submodelObj* = addmodel(*modelObj*, '*NameValue*')
                  submodelObj = addmodel(...'*PropertyName*', *PropertyValue*...)

Arguments

modelObj	Model object. Enter a name for a model object.
NameValue	Descriptive name for a model object. Enter a unique character string. A model object can be referenced by other objects using this property.
submodelObj	Model object to be added as a submodel.

Description   *submodelObj* = addmodel(*modelObj*, '*NameValue*') creates a submodel object and returns to *submodelObj*. In the submodel object, this method assigns a value (*NameValue*) to the property Name, and assigns the model object (*modelObj*) to the property Parent. In the model object, this method assigns the submodel object to the property Models.

```
modelObj = sbiomodel('cell')
submodelObj = addmodel('nucleus')
```

addmodel (model)
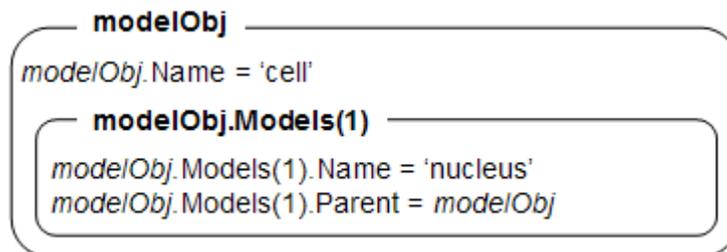
A model object must have a unique name at the level it is created. For example, if you create a model with the name `cell`, you cannot create another model object named `cell`. However, a model object can contain a submodel object named cell which can contain a submodel object named `cell`.

modelObj does not have access to *submodelObj* parameters. However, *submodelObj* does have access and can use *modelObj* parameters.

submodelObj = addmodel(...*'PropertyName'*, *PropertyValue*...) defines optional property values. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

You can view additional model object properties with the function `get`. You can change additional model object properties with the function `set`. You can view the submodel objects of modelObj with the command `get(modelObj, 'Models')`.

See Also   sbiomodel, sbioupdate

addparameter (model, kineticlaw)

Purpose   Create parameter object and add to model or kinetic law object

Syntax   parameterObj = addparameter(*Obj*, '*NameValue*')
                 parameterObj = addparameter(Obj, '*NameValue*', *ValueValue*)
                 parameterObj = addparameter(...'*PropertyName*', *PropertyValue*...)

Arguments

Obj	Model or kinetic law object. Enter a variable name for the object.
NameValue	Property for a parameter object. Enter a unique character string. Since objects can use this property to reference a parameter, a parameter object must have a unique name at the level it is created. For example, a kinetic law object cannot contain two parameter objects named kappa. However, the model object that contains the kinetic law object can contain a parameter object named kappa along with the kinetic law object.
	For information on naming parameters, see Name.
ValueValue	Property for a parameter object. Enter a number.

Description   parameterObj = addparameter(*Obj*, '*NameValue*') creates a parameter
object and returns the object (*parameterObj*). In the parameter object,
this method assigns a value (*NameValue*) to the property Name, assigns
a value 1 to the property Value, and assigns the model or kinetic law
object to the property Parent. In the model or kinetic law object, (*Obj*),
this method assigns the parameter object to the property Parameters.

A parameter object defines an assignment that a model or a kinetic
law can use. The scope of the parameter is defined by the parameter
parent. If a parameter is defined with a kinetic law object, then only the
kinetic law object and objects within the kinetic law object can use the

addparameter (model, kineticlaw)

parameter. If a parameter object is defined with a model object as its parent, then all objects within the model (including all rules, events and kinetic laws) can use the parameter.

```
modelObj = sbiomodel('cell')
parameterObj = addparameter(modelObj, 'TF1', 0.01)
```
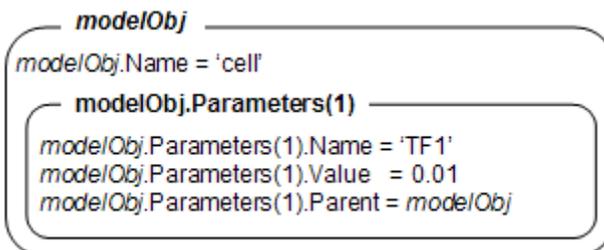


```
modelObj = sbiomodel('cell')
reactionObj = addreaction(modelObj, 'a -> b')
kineticlawObj = addkineticlaw (reactionObj, 'MassAction')
parameterObj = addparameter(kineticlawObj, 'K1_forward', 0.1)
```
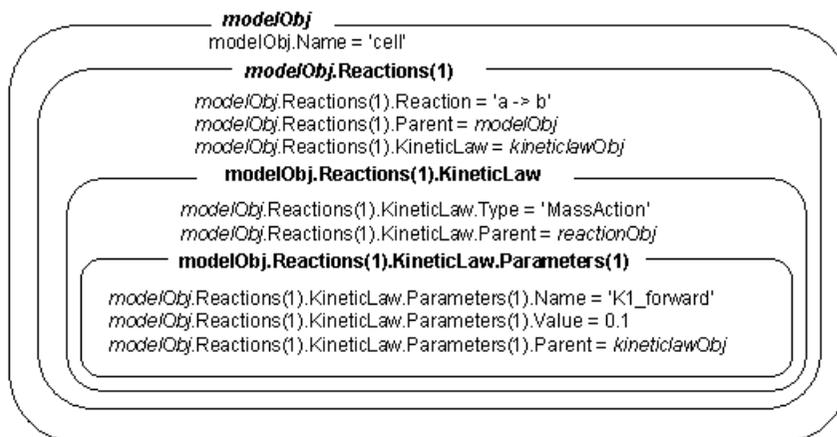
parameterObj = addparameter(Obj, '*NameValue*', *ValueValue*) creates a parameter object, assigns a value (*NameValue*) to the property Name, assigns the value (*ValueValue*) to the property Value, and assigns the model object or the kinetic law object to the property Parent. In the model or kinetic law object (Obj), this method assigns the parameter object to the property Parameters, and returns the parameter object to a variable (parameterObj).

parameterObj = addparameter(...'*PropertyName*', *PropertyValue*...) defines optional property values. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

Scope of a parameter — A parameter can be *scoped* to either a model or a kinetic law.

- When a kinetic law searches for a parameter in its expression, it first looks in the parameter list of the kinetic law. If the parameter isn't found there, it moves to the model that the kinetic law object is in and looks in the model parameter list. If the parameter isn't found there, it moves to the model parent.

- When a rule searches for a parameter in its expression, it looks in the parameter list for the model. If the parameter isn't found there, it moves to the model parent. A rule cannot use a parameter that is scoped to a kinetic law. So for a parameter to be used in both a reaction rate equation and a rule, the parameter should be *scoped* to a model.

Additional parameter object properties can be viewed with the get command. Additional parameter object properties can be modified with the set command. The parameters of Obj can be viewed with get(*Obj*, 'Parameters').

A SimBiology parameter object can be copied to a SimBiology model or kinetic law object with copyobj. A SimBiology parameter object can be removed from a SimBiology model or kinetic law object with delete.

addparameter (model, kineticlaw)

Method Summary	Methods for parameter objects	
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	rename (compartment, parameter, species)	Rename object and update expressions
	set (any object)	Set object properties

Property Summary	Properties for parameter objects	
	Annotation	Store link to URL or file
	ConstantValue	Specify variable or constant parameter value
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	Parent	Indicate parent object
	Tag	Specify label for SimBiology object
	Type	Display top-level SimBiology object type
	UserData	Specify data to associate with object
	Value	Assign value to parameter object
	ValueUnits	Parameter value units

addparameter (model, kineticlaw)

Example

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

3 Add a parameter and assign it to the kinetic law object
(kineticlawObj); add another parameter and assign to the model
object (modelObj).

```
% Add parameter to kinetic law object
parameterObj1 = addparameter (kineticlawObj, 'K1');

get (kineticlawObj, 'Parameters')
```

MATLAB returns:

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1        K1        1

% Add parameter with value 0.9 to model object
parameterObj1 = addparameter (modelObj, 'K2', 0.9);

get (modelObj, 'Parameters')
```

MATLAB returns:

```
SimBiology Parameter Array

 Index:    Name:    Value:    ValueUnits:
   1       K2        1
```

See Also   addreaction

addproduct (reaction)

Purpose   Add product species object to reaction object

Syntax

speciesObj = addproduct(*reactionObj*, '*NameValue*')
speciesObj = addproduct(*reactionObj*, *speciesObj*)
speciesObj = addproduct(*reactionObj*, '*NameValue*',
   Stoichcoefficient)
speciesObj = addproduct(*reactionObj*, *speciesObj*,
   Stoichcoefficient)

Arguments

reactionObj	Reaction object. Enter a name for the reaction object.
NameValue	Property of a species object that names the object (not the reaction object). Enter a unique character string. For example, `'fructose 6-phosphate'`. A species object can be referenced by other objects using this property. You can use the function `sbioselect` to find an object with a specific `NameValue`.
speciesObj	Species object.
Stoichcoeffieient	Stoichiometric coefficients for products, length of array equal to length of `NameValue`, or length of *speciesObj*.

Description   *speciesObj* = addproduct(*reactionObj*, '*NameValue*') creates
a species object and returns the species object (*speciesObj*). In
the species object, this method assigns the value (*NameValue*) to
the property Name. In the reaction object, this method assigns the
species object to the property Products, modifies the reaction equation
in the property Reaction to include the new species, and adds the
stoichiometric coefficient 1 to the property Stoichiometry.

When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.

- If only one compartment object (compObj) exists in the model, the method creates a species object in that compartment.

- If there is more than one compartment object (compObj) in the model, you must qualify the species name with the compartment name.

  For example, cell.glucose denotes that you want to put the species named glucose into a compartment named cell. Additionally, if the compartment named cell does not exist, the process of adding the reaction creates the compartment and names it cell.

Create and add a species object to a compartment object with the method addspecies.

speciesObj = addproduct(*reactionObj*, *speciesObj*), in the species object (*speciesObj*), assigns the parent object of the *reactionObj* to the species property Parent. In the reaction object (reactionObj), it assigns the species object to the property Products, modifies the reaction equation in the property Reaction to include the new species, and adds the stoichiometric coefficient 1 to the property Stoichiometry.

speciesObj = addproduct(*reactionObj*, '*NameValue*', *Stoichcoefficient*), in addition to the description above, adds the stoichiometric coefficient (Stoichcoefficient) to the property Stoichiometry. If NameValue is a cell array of species names, then Stoichcoefficient must be a vector of doubles with the same length as NameValue.

speciesObj = addproduct(*reactionObj*, *speciesObj*, *Stoichcoefficient*), in addition to the description above, adds the stoichiometric coefficient (Stoichcoefficient) to the property Stoichiometry.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the Name of a species the reaction also

addproduct (reaction)

uses the new name. You must however configure all other applicable
elements such as rules that use the species, and the kinetic law object.

Examples   **1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'A + C -> U');
```

2 Modify the reaction of the reactionObj from A + C -> U to A + C
-> U + 2 H.

```
speciesObj = addproduct(reactionObj, 'H', 2);
```

See Also   addspecies, sbiospecies

Purpose   Add species object as reactant to reaction object

Syntax
speciesObj = addreactant(*reactionObj*, '*NameValue*')
addreactant(*reactionObj*, *speciesObj*, *Stoichcoeffieient*)
addreactant(*reactionObj*, '*NameValue*', *Stoichcoeffieient*)

Arguments

reactionObj	Reaction object.
NameValue	Name property of a species object. Enter a unique character string, for example, 'glucose'. A species object can be referenced by other objects using this property. You can use the function sbioselect to find an object with a specific Name property value.
speciesObj	Species object or cell array of species objects.
Stoichcoeffieient	Stoichiometric coefficients for reactants, length of array equal to length of NameValue or length of *speciesObj*.

Description   *speciesObj* = addreactant(*reactionObj*, '*NameValue*') creates a species object and returns the species object (*speciesObj*). In the species object, this method assigns the value (*NameValue*) to the property Name. In the reaction object, this method assigns the species object to the property Reactants, modifies the reaction equation in the property Reaction to include the new species, and adds the stoichiometric coefficient -1 to the property Stoichiometry.

When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.

- If only one compartment object (compObj) exists in the model, the method creates a species object in that compartment.

addreactant (reaction)

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

  For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Create and add a species object to a compartment object with the method `addspecies`.

`addreactant(`*reactionObj*`, `*speciesObj*`, `*Stoichcoeffieient*`)`, in the species object (`speciesObj`), assigns the parent object to the `speciesObj` property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `-1` to the property `Stoichiometry`. If `speciesObj` is a cell array of species objects, then *Stoichcoeffieient* must be a vector of doubles with the same length as `speciesObj`.

`addreactant(`*reactionObj*`, `*'NameValue'*`, `*Stoichcoeffieient*`)`, in addition to the description above, adds the stoichiometric coefficient (*Stoichcoeffieient*) to the property `Stoichiometry`. If *NameValue* is a cell array of species names, then *Coefficient* must be a vector of doubles with the same length as *NameValue*.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species the reaction also uses the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object.

See for more information on species names.

Example

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'A -> U');
```

2 Modify the `reaction` of the `reactionObj` from A -> U to be A + 3 C -> U.

```
speciesObj = addreactant(reactionObj, 'C', 3);
```

See Also   addspecies, sbiospecies

addreaction (model)

Purpose   Create reaction object and add to model object

Syntax

reactionObj = addreaction(*modelObj*,'*ReactionValue*')
reactionObj = addreaction(*modelObj*, '*ReactantsValue*',
   '*ProductsValue*')
reactionObj = addreaction(*modelObj*, '*ReactantsValue*',
   RStoichCoefficients, '*ProductsValue*',
   PStoichCoefficients)
reactionObj = addreaction(...'*PropertyName*', *PropertyValue*...)

Arguments

modelObj	SimBiology model object.
ReactionValue	Specify the reaction equation. Enter a character string. A hyphen preceded by a space and followed by a right angle bracket (->) indicates reactants going forward to products. A hyphen with left and right angle brackets (<->) indicates a reversible reaction. Coefficients before reactant or product names must be followed by a space.
	Examples are 'A -> B', 'A + B -> C', '2 A + B -> 2 C', and 'A <-> B'. Enter reactions with spaces between the species.
	If there are multiple compartments, or to specify the compartment name, use *compartmentName.speciesName*.
	Examples are 'cytoplasm.A -> cytoplasm.B', 'cytoplasm.A -> nucleus.A', and 'cytoplasm.A + cytoplasm.B -> nucleus.AB'.

ReactantsValue	A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments.
ProductsValue	A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments.
RStoichCoefficients	Stoichiometric coefficients for reactants, length of array equal to length of *ReactantsValue*.
PStoichCoefficients	Stoichiometric coefficients for products, length of array equal to length of *ProductsValue*.

Description   *reactionObj* = addreaction(*modelObj*,'*ReactionValue*') creates a reaction object, assigns a value (*ReactionValue*) to the property Reaction, assigns reactant species object(s) to the property Reactants, assigns the product species object(s) to the property Products, and assigns the model object to the property Parent. In the Model object (modelObj), this method assigns the reaction object to the property Reactions, and returns the reaction object (reactionObj).

```
reactionObj = addreaction(modelObj, 'a -> b')
```
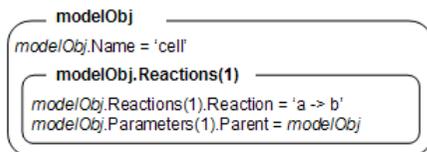

When you define a reaction with a new species:

addreaction (model)

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.

- If only one compartment object (`compObj`) exists in the model, the method creates a species object in that compartment.

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

  For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

You can manually add a species to a compartment object with the method `addspecies`.

You can add species to a reaction object using the methods `addreactant` or `addproduct`. You can remove species from a reaction object with the methods `rmreactant` or `rmproduct`. The property `Reaction` is modified by adding or removing species from the reaction equation.

You can copy a SimBiology reaction object to a model object with the function `copyobj`. You can remove the SimBiology reaction object from a SimBiology model object with the function `delete`.

You can view additional reaction object properties with the `get` command. For example, the reaction equation of `reactionObj` can be viewed with the command `get(`*reactionObj*`, 'Reaction')`. You can modify additional reaction object properties with the command `set`.

reactionObj = addreaction(*modelObj*, *'ReactantsValue'*, *'ProductsValue'*) creates a reaction object, assigns a value to the property `Reaction` using the reactant (*ReactantsValue*) and product (*ProductsValue*) names, assigns the species objects to the properties `Reactants` and `Products`, and assigns the model object to the property `Parent`. In the model object (`modelObj`), this method assigns the reaction object to the property `Reactions`, and returns the reaction object (`reactionObj`). The stoichiometric values are assumed to be 1.

reactionObj = addreaction(*modelObj*, '*ReactantsValue*',
RStoichCoefficients, '*ProductsValue*', *PStoichCoefficients*) adds
stoichiometric coefficients (*RStoichCoefficients*) for reactant species,
and stoichiometric coefficients (PStoichCoefficients) for product
species to the property Stoichiometry. The length of Reactants
and RCoefficients must be equal, and the length of Products and
PCoefficients must be equal.

reactionObj = addreaction(...'*PropertyName*', *PropertyValue*...)
defines optional properties. The property name/property value pairs
can be in any format supported by the function set (for example,
name-value string pairs, structures, and name-value cell array pairs).

**Method
Summary**

Methods for reaction objects

addkineticlaw (reaction)	Create kinetic law object and add to reaction object
addproduct (reaction)	Add product species object to reaction object
addreactant (reaction)	Add species object as reactant to reaction object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmproduct (reaction)	Remove species object from reaction object products
rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

addreaction (model)

Property Summary
Properties for reaction objects

`Active`	Indicate object in use during simulation
`Annotation`	Store link to URL or file
`KineticLaw`	Show kinetic law used for `ReactionRate`
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Products`	Array of reaction products
`Reactants`	Array of reaction reactants
`Reaction`	Reaction object reaction
`ReactionRate`	Reaction rate equation in reaction object
`Reversible`	Specify whether reaction is reversible or irreversible
`Stoichiometry`	Species coefficients in reaction
`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

Examples
Create a model, add a reaction object, and assign the expression for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj KineticLaw property is configured to kineticlawObj.

3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that should to be set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm_d, and Km_d, and assign the objects Parent property value to the kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

4 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Vm_d' 'Km_d'});
set(kineticlawObj,'SpeciesVariableNames', {'a'});
```

5 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Vm_d*a/(Km_d+a)
```

See Also   addkineticlaw, addproduct, addreactant, rmproduct, rmreactant

addrule (model)

Purpose   Create rule object and add to model object

Syntax
ruleObj = addrule(*modelObj*, '*RuleValue*')
ruleObj = addrule(*modelObj*, '*RuleValue*', '*RuleTypeValue*')
ruleObj = addrule(..., '*PropertyName*', *PropertyValue*,...)

Arguments

modelObj	Model object to which to add the rule.
RuleValue	Enter a character string within quotation marks. For example, enter the algebraic rule 'Va*Ea + Vi*Ei - K2'.
RuleTypeValue	Enter 'algebraic', 'initialassignment', 'repeatedAssignment', or 'rate'. See RuleType for more information.

Description   A rule is a mathematical expression that changes the amount of a species or the value of a parameter. It also defines how species and parameters interact with one another.

ruleObj = addrule(*modelObj*, '*RuleValue*') creates a rule object and returns the rule object (*ruleObj*). In the rule object, this method assigns a value ('*RuleValue*') to the property Rule, assigns the value 'algebraic' to the property RuleType, and assigns the model object (*modelObj*) to the property Parent. In the model object (modelObj), this method assigns the rule object to the property Rules.

ruleObj = addrule(*modelObj*, '*RuleValue*', '*RuleTypeValue*') in addition to the assignments above, assigns a value (*RuleTypeValue*) to the property RuleType. For more information on the different types of rules, see RuleType.

ruleObj = addrule(..., '*PropertyName*', *PropertyValue*,...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

View additional rule properties with the function get, and modify rule properties with the function set. Copy a rule object to a model with the function copyobj, or delete a rule object from a model with the function delete.

Method Summary   Methods for rule objects

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary   Properties for rule objects

Active	Indicate object in use during simulation
Annotation	Store link to URL or file
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Rule	Specify species and parameter interactions
RuleType	Specify type of rule for rule object
Tag	Specify label for SimBiology object

Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples   Add a rule with the default `RuleType`.

1 Create a model object, and then add a rule object.

```
modelObj = sbiomodel('cell');
ruleObj = addrule(modelObj, 'O.1*B-A')
```

2 Get a list of properties for a rule object.

```
get(modelObj.Rules(1)) or get(ruleObj)
```

MATLAB displays a list of rule properties.

```
     Active: 1
 Annotation: ''
       Name: ''
      Notes: ''
     Parent: [1x1 SimBiology.Model]
       Rule: 'O.1*B-A'
   RuleType: 'algebraic'
        Tag: ''
       Type: 'rule'
   UserData: []
```

Add a rule with the `RuleType` property set to `rate`.

1 Create model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a -> b');
```

2 Add a rule which defines that the quantity of a species c. In the rule expression, k is the rate constant for a -> b.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

3 Change the RuleType from default ('algebraic') to 'rate', and verify using the get command.

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
     Active: 1
 Annotation: ''
       Name: ''
      Notes: ''
     Parent: [1x1 SimBiology.Model]
       Rule: 'c = k*(a+b)'
   RuleType: 'rate'
        Tag: ''
       Type: 'rule'
   UserData: []
```

See Also   copyobj, delete, sbiomodel

addspecies (compartment)

Purpose   Create species object and add to compartment object

Syntax
speciesObj = addspecies(*compObj*, '*NameValue*')
speciesObj = addspecies(*compObj*, '*NameValue*',
   InitialAmountValue)
speciesObj = addspecies(...'*PropertyName*', *PropertyValue*...)

Arguments

compObj	Compartment object.
NameValue	Name for a species object. Enter a character string unique within *compObj*. Species objects are identified by name within Event, ReactionRate, and Rule property strings. For information on naming species, see Name.
	You can use the function sbioselect to find an object with a specific Name property value.
InitialAmountValue	Initial amount value for the species object. Enter double. Positive real number, default = 0.
PropertyName	Enter the name of a valid property. Valid property names are listed in "Property Summary" on page 4-54.
PropertyValue	Enter the value for the property specified in *PropertyName*. Valid property values are listed on each property reference page.

Description   *speciesObj* = addspecies(*compObj*, '*NameValue*') creates a species object and returns the species object (speciesObj). In the species object, this method assigns a value (*NameValue*) to the property Name, and assigns the compartment object (*compObj*) to the property Parent. In the compartment object, this method assigns the species object to the property Species.

speciesObj = addspecies(*compObj*, '*NameValue*',
InitialAmountValue), in addition to the above, assigns an initial
amount *(InitialAmountValue)* for the species.

You can also add a species to a reaction using the methods addreactant
and addproduct.

A species object must have a unique name at the level at which it is
created. For example, a compartment object cannot contain two species
objects named H2O. However, another compartment can have a species
named H2O.

View properties for a species object with the get command, and modify
properties for a species object with the set command. You can view
a summary table of species objects in a compartment (compObj) with
get(compObj, 'Species') or the properties of the first species with
get(compObj.Species(1)).

speciesObj = addspecies(...'*PropertyName*', *PropertyValue*...)
defines optional properties. The property name/property value pairs
can be in any format supported by the function set (for example,
name-value string pairs, structures, and name-value cell array pairs).
The property summary on this page shows the list of properties.

If there is more than one compartment object (compObj) in the model,
you must qualify the species name with the compartment name. For
example, cell.glucose denotes that you want to put the species
named glucose into a compartment named cell. Additionally, if the
compartment named cell does not exist, the process of adding the
reaction creates the compartment and names it cell.

If you change the name of a species you must configure all applicable
elements, such as events and rules that use the species, any
user-specified ReactionRate, or the kinetic law object property
SpeciesVariableNames. Use the method setspecies to configure
SpeciesVariableNames.

To update species names in the SimBiology graphical user interface,
access each appropriate pane through the **Project Explorer**. You can
also use the **Find** feature to locate the names that you want to update.

addspecies (compartment)

The **Output** pane opens with the results of **Find**. Double-click a result row to go to the location of the model component.

Species names are automatically updated for reactions that use MassAction kinetic law.

Method Summary	Methods for species objects	
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	rename (compartment, parameter, species)	Rename object and update expressions
	set (any object)	Set object properties

Property Summary	Properties for species objects	
	Annotation	Store link to URL or file
	BoundaryCondition	Indicate species boundary condition
	ConstantAmount	Specify variable or constant species amount
	InitialAmount	Species initial amount
	InitialAmountUnits	Species initial amount units
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	Parent	Indicate parent object

Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples   Add two species to a model, where one is a reactant and the other is the enzyme catalyzing the reaction.

1 Create a model object named my_model and add a compartment object.

```
modelObj = sbiomodel ('my_model');
compObj = addcompartment(modelObj, 'comp1');
```

2 Add two species objects named glucose_6_phosphate and glucose_6_phosphate_dehydrogenase.

```
speciesObj1 = addspecies (compObj, 'glucose_6_phosphate');
speciesObj2 = addspecies (compObj, ...
                          'glucose_6_phosphate_dehydrogenase');
```

3 Set the initial amount of glucose_6_phosphate to 100 and verify.

```
set (speciesObj1, 'InitialAmount',100);
get (speciesObj1, 'InitialAmount')
```

MATLAB returns:

```
ans =

    100
```

4 Use get to note that modelObj contains the species object array.

```
get(compObj, 'Species')
```

MATLAB returns:

```
SimBiology Species Array

Index: Name:                                  InitialAmount: InitialAmountUnits:
 1     glucose_6_phosphate                     100
 2     glucose_6_phosphate_dehydrogenase  0
```

5 Retrieve information about the first species in the array.

```
get(compObj.Species(1))
            Annotation: ''
     BoundaryCondition: 0
         ConstantAmount: 0
          InitialAmount: 100
     InitialAmountUnits: ''
                   Name: 'glucose_6_phosphate'
                  Notes: ''
                 Parent: [1x1 SimBiology.Compartment]
                    Tag: ''
                   Type: 'species'
               UserData: []
```

See Also   addcompartment, addproduct, addreactant, addreaction, get, set

addvariant (model)

Purpose   Add variant to model

Syntax   *variantObj* = addvariant(*modelObj*, '*NameValue*')
variantObj2 = addvariant(*modelObj*, *variantObj*)

Arguments

modelObj	Specify the model object to which you want add a variant.
variantObj	Variant object to create and add to the model object.
NameValue	Name of the variant object. *NameValue* is assigned to the Name property of the variant object.

Description   *variantObj* = addvariant(*modelObj*, '*NameValue*') creates a SimBiology variant object (*variantObj*) with the name *NameValue* and adds the variant object to the SimBiology model object *modelObj*. The variant object Parent property is assigned the value of *modelObj*.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see Variant object.

variantObj2 = addvariant(*modelObj*, *variantObj*) adds a SimBiology variant object (*variantObj*) to the SimBiology model object and returns another variant object *variantObj2*. The variant object *variantObj2* Parent property is assigned the value of *modelObj*.

View properties for a variant object with the get command, and modify properties for a variant object with the set command.

Note Remember to use the addcontent method instead of using the set method on the Content property, because the set method replaces the data in the Content property, whereas addcontent appends the data.

---

addvariant (model)

To view the variants stored on a model object, use the `getvariant` method. To copy a variant object to another model, use `copyobj`. To remove a variant object from a SimBiology model, use the `delete` method.

Examples

1 Create a model containing one species.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
speciesObj = addspecies(compObj, 'A');
```

2 Add a variant object that varies the `InitialAmount` property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

See Also   `addcontent`, `commit`, `copyobj`, `delete`, `getvariant`

Purpose	Commit variant contents to model

Syntax   commit(*variantObj*, *modelObj*)

Arguments

modelObj	Specify the model object to which you want to commit a variant.
variantObj	Variant object to commit to the model object.

Description   commit(*variantObj*, *modelObj*) commits the Contents property of a SimBiology variant object (*variantObj*) to the model object *modelObj*. The property values stored in the variant object replace the values stored in the model.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see Variant object.

The Contents are set on the model object in order of occurrence, with duplicate entries overwriting. If the commit method finds an incorrectly specified entry, an error occurs and the remaining properties defined in the Contents property are not set.

Examples   **1** Create a model containing one species.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
speciesObj = addspecies(compObj, 'A', 10);
```

2 Add a variant object that varies the InitialAmount property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

3 Commit the contents of the variant (*variantObj*).

commit (variant)

```
commit (variantObj, modelObj);
```

See Also   addvariant, Variant object

Purpose   Options for compartments

Description   The SimBiology compartment object represents a container for species in a model. Compartment size can vary or remain constant during a simulation. All models must have at least one compartment and all species in a model must be assigned to a compartment. Compartment names must be unique within a model.

Compartments allow you to define the size (`Capacity`) of physically isolated regions that may affect simulation, and associate pools of species within those regions. You can specify or change `Capacity` using rules, events, and variants, similar to species amounts or parameter values.

The model object stores compartments as a flat list. Each compartment stores information on its own organization; in other words a compartment has information on which compartment it lives within (`Owner`) and who it contains (`Compartments`).

The flat list of compartments in the model object lets you vary the way compartments are organized in your model without invalidating any expressions.

To add species that participate in reactions, add the reaction to the model using the `addreaction` method. When you define a reaction with a new species:

- If no compartment objects exist in the model, the `addreaction` method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.

- If only one compartment object exists in the model, the method creates a species object in that compartment.

- If there is more than one compartment object in the model, you must qualify the species name with the compartment name.

  For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the

compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Alternatively, create and add a species object to a compartment object, using the `addspecies` method at the command line.

The SimBiology desktop adds a default compartment (*unnamed*) for you and you can add a species in the **Species** pane. In the **Project Explorer**, expand **Compartment** and click **Species** to open the **Species** pane.

You can specify reactions that cross compartments using the syntax *compartment1Name.species1Name –> compartment2Name.species2Name*. If you add a reaction that contains species from different compartments, and the reaction rate dimensions are concentration/time, all reactants should be from the same compartment.

In addition, if the reaction is reversible then there are two cases:

• If the kinetic law is `MassAction`, and the reaction rate reaction rate dimensions are concentration/time, then the products must be from the same compartment.

• If the kinetic law is not `MassAction`, then both reactants and products must be in the same compartment.

See "Property Summary" on page 4-63 for links to compartment property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addcompartment (model, compartment)	Create compartment object

Method Summary

Methods for compartment objects

addcompartment (model, compartment)	Create compartment object
addspecies (compartment)	Create species object and add to compartment object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties

Property Summary

Properties for compartment objects

Annotation	Store link to URL or file
Capacity	Compartment capacity
CapacityUnits	Compartment capacity units
Compartments	Array of compartments in model or compartment
ConstantCapacity	Specify variable or constant compartment capacity
Name	Specify name of object
Notes	HTML text describing SimBiology object

Compartment object

Owner	Owning compartment
Parent	Indicate parent object
Species	Array of species in compartment object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

See Also   AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object

Purpose   Solver settings information for model simulation

Description   The SimBiology configset object, also known as the configuration set object, contains the options that the solver uses during simulation of the model object. The configuration set object contains the following options for you to choose:

- Type of solver

- Stop time for the simulation

- Solver error tolerances, and for ode solvers — the maximum time step the solver should take

- Whether to perform sensitivity analysis during simulation

- Whether to perform dimensional analysis and unit conversion during simulation

- Species and parameter input factors for sensitivity analysis

A SimBiology model can contain multiple configsets with one being active at any given time. The active configset contains the settings that are used during the simulation. Use the method setactiveconfigset to define the active configset. Use the method getconfigset to return a list of configsets contained by a model. Use the method addconfigset to add a new configset to a model.

See "Property Summary" on page 4-66 for links to configset object property reference pages.

Properties define the characteristics of an object. Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

Constructor Summary

addconfigset (model)	Create configuration set object and add to model object

Configset object

Method Summary	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	set (any object)	Set object properties

Property Summary	Active	Indicate object in use during simulation
	CompileOptions	Dimensional analysis and unit conversion options
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	RuntimeOptions	Options for logged species
	SensitivityAnalysisOptions	Specify sensitivity analysis options
	SolverOptions	Specify model solver options
	SolverType	Select solver type for simulation
	StopTime	Set stop time for simulation
	StopTimeType	Specify type of stop time for simulation
	TimeUnits	Show stop time units for simulation
	Type	Display top-level SimBiology object type

See Also   AbstractKineticLaw object, KineticLaw object, Model object,
                 Parameter object, Reaction object, Root object, Rule object,
                 Species object

construct (PKModelDesign)

Purpose	Construct SimBiology model from PKModelDesign object
Syntax	[*modelObj*, *pkModelMapObject*] = construct(pkModelDesignObject)

Arguments

modelObj	SimBiology model object specifying a pharmacokinetic model.
pkModelMapObject	Defines the roles of the components in *modelObj*. See PKModelMap object for more information.

Description [*modelObj*, *pkModelMapObject*] = construct(pkModelDesignObject) constructs a SimBiology model object containing the model components (such as compartments, species, reactions, and rules) required to represent a pharmacokinetic model specified in pkModelDesignObject. The newly constructed model object is named 'Generated Model' (which you can change).

See Also "Creating PK Models at the Command Line" in the SimBiology User's Guide, PKModelDesign object

Purpose	Copy SimBiology object and its children
Syntax	*copiedObj* = copyobj(*Obj*, *parentObj*) *copiedObj* = copyobj(*modelObj*)

Arguments

Obj	Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, reaction, rule, species, or variant object.
parentObj	

If *copiedObj* is...	*parentObj* must be...
configuration set, event, reaction, rule, or variant object	model object
compartment object	compartment or model object
species object	compartment object
parameter object	model or kinetic law object
kinetic law object	reaction object
model object or abstract kinetic law object	sbioroot

modelObj	Model object to be copied.
copiedObj	Output returned by the copyobj method with the parent set as specified in input argument (*parentObj*).

Description *copiedObj* = copyobj(*Obj*, *parentObj*) makes a copy of a SimBiology object (*Obj*) and returns a pointer to the copy (*copiedObj*). In the copied object (*copiedObj*), this method assigns a value (*parentObj*) to the property Parent.

copyobj (any object)

copiedObj = copyobj(*modelObj*) makes a copy of a model object (*modelObj*) and returns the copy (*copiedObj*). In the copied model object (*copiedObj*), this method assigns the root object to the property Parent.

Examples

Create a reaction object separate from a model object, and then add it to a model.

1 Create a model object and add a reaction object.

```
modelObj1 = sbiomodel('cell');
reactionObj = addreaction(modelObj1, 'a -> b');
```

2 Create a copy of the reaction object and assign it to another model object.

```
modelObj2 = sbiomodel('cell2');
reactionObjCopy = copyobj(reactionObj, modelObj2);
modelObj2.Reactions

SimBiology Reaction Array

Index:    Reaction:
 1          a -> b
```

See Also

sbiomodel, sbioroot

delete (any object)

Purpose   Delete SimBiology object

Syntax   delete(*Obj*)

Arguments

Obj	SimBiology object: abstract kinetic law, configuration set, kinetic law, model, parameter, reaction, rule, or species.

Description   delete(*Obj*) removes an object (*Obj*) from its parent.

- If *Obj* is a species object that is being used by a reaction object, this method returns an error and the species object is not deleted. You need to delete the reaction or remove the species from the reaction before you can delete the species object.

- If *Obj* is a parameter object being used by a kinetic law object, there is no warning when the object is deleted. However, when you try to simulate your model, a error occurs because the parameter cannot be found.

- If *Obj* is a reaction object, this method deletes the object, but the species objects that were being used by the reaction object are not deleted.

- If *Obj* is an abstract kinetic law object and there is a kinetic law object referencing it, this method returns an error.

- If *Obj* is a SimBiology configuration set object, and it is the active configuration set object, this method, after deleting the object, makes the default configuration set object active. Note that you cannot delete the default configuration set.

- You cannot delete the SimBiology root.

You can also delete all model objects from the root with one call to the sbioreset function.

delete (any object)

Examples **Example 1**

Delete a reaction from a model. Notice the species objects are not deleted with the reaction object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'a -> b');
delete(reactionObj)
```

Example 2

Delete a single model from the root object.

```
modelObj1 = sbiomodel('cell');
modelObj2 = sbiomodel('virus');
delete(ModelObj2)
```

See Also sbiomodel, sbioreset, sbioroot

display (any object)

Purpose   Display summary of SimBiology object

Syntax   display(Obj)

Arguments

    Obj   SimBiology object: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, rule, species, or unit.

Description Display the SimBiology object array. display(Obj) is called for the SimBiology object, *Obj* when the semicolon is not used to terminate a statement. The display of *Obj* gives a brief summary of the *Obj* configuration. You can view a complete list of *Obj* properties with the command get. You can modify all *Obj* properties that can be changed, with the command set.

Examples
```
modelObj = sbiomodel('cell')
reactionObj = addreaction(modelObj, 'A + B -> C')
```

Event object

Purpose   Store event information

Description   Events are used to describe sudden changes in model behavior. An event lets you specify discrete transitions in model component values that occur when a user-specified condition become true. You can specify that the event occurs at a particular time, or specify a time-independent condition.

For details on how events are handled during a simulation, see "Changing Model Component Values Using Events" in the SimBiology User's Guide documentation.

See "Property Summary" on page 4-75 for links to event property reference pages.

Properties define the characteristics of an object. For example, an event object includes properties that allow you to specify the conditions to trigger an event (`Trigger`), and what to do after the event is triggered (`EventFcn`). Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

Constructor Summary

addevent (model)	Add event object to model object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

Active	Indicate object in use during simulation
Annotation	Store link to URL or file
EventFcns	Event expression
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Trigger	Event trigger
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

See Also   AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object

get (any object)

Purpose	Get object properties

Syntax

PropertyValue = get(*Obj*, *'PropertyName'*)
objProperties = get(*Obj*)

Arguments

PropertyValue	Value defined for '*PropertyName*'
Obj	Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, PKCompartment, PKData, PKModelDesign PKModelMap, reaction, rule, SimData, species, or variant object.
'*PropertyName*'	Name of the property to get.
objProperties	Struct containing properties and values for the object, *Obj*.

Description

PropertyValue = get(*Obj*, *'PropertyName'*) gets the value *'PropertyValue'* of the object, *Obj*'s *PropertyName* property.

objProperties = get(*Obj*) gets the properties for the object, *Obj*, and returns it to *objProperties*.

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

3 Set the `ConstantValue` property of the parameter object to `false` and verify.

MATLAB returns `1` for `true` and `0` for `false`.

```
set (parameterObj, 'ConstantValue', false);
get(parameterObj, 'ConstantValue')
```

MATLAB returns

```
ans =

     0
```

See Also   `getadjacencymatrix`, `getconfigset`, `getdata`, `getparameters`, `getsensmatrix`, `getspecies`, `getstoichmatrix`, `set`

getadjacencymatrix (model)

Purpose	Get adjacency matrix from model object

Syntax

M = getadjacencymatrix(*modelObj*)
M = getadjacencymatrix(*modelObj*,'flat')
[*M*, *Headings*] = getadjacencymatrix(*modelObj*)
[*M*, *Headings*, *Mask*] = getadjacencymatrix(*modelObj*)

Arguments

M	Adjacency matrix for *modelObj*.
modelObj	Specify the model object.
'flat'	Return adjacency matrix for only specified *modelObj*, not for objects contained in the *modelObj*.
Headings	Return row and column headings. If species are in multiple compartments, species names are qualified with the compartment name in the form compartmentName.speciesName. For example, nucleus.DNA, cytoplasm.mRNA.
Mask	Return 1 for the species object and 0 for the reaction object to *Mask*.

Description getadjacencymatrix returns the adjacency matrix for a model object.

M = getadjacencymatrix(*modelObj*) returns an adjacency matrix for the model object (*modelOBJ*) to *M*.

An adjacency matrix is defined by listing all species contained by *modelObj* and all reactions contained by *modelObj* column-wise and row-wise in a matrix. The reactants of the reactions are represented in the matrix with a 1 at the location of [row of species, column of reaction]. The products of the reactions are represented in the matrix with a 1 at the location of [row of reaction, column of species]. All other locations in the matrix are 0.

M = getadjacencymatrix(*modelObj*,'flat') returns the adjacency matrix to *M* and defines the adjacency matrix for only *modelObj*. *M* is the adjacency matrix for the reactions and species contained by *modelObj*.

[*M*, *Headings*] = getadjacencymatrix(*modelObj*) returns the adjacency matrix to M and the row and column headings to *Headings*. *Headings* is defined by listing all Name property values of species contained by *modelObj* and all Name property values of reactions contained by *modelObj*.

[*M*, *Headings*, *Mask*] = getadjacencymatrix(*modelObj*) returns an array of 1s and 0s to *Mask*, where a 1 represents a species object and a 0 represents a reaction object.

Examples **1** Read in a model using sbmlimport.

```
modelObj = sbmlimport('lotka.xml');
```

2 Get the adjacency matrix for the modelObj.

```
[M, Headings] = getadjacencymatrix(modelObj)
```

See Also getstoichmatrix

getconfigset (model)

Purpose	Get configuration set object from model object
Syntax	*configsetObj* = getconfigset(*modelObj*, '*NameValue*') *configsetObj* = getconfigset(*modelObj*) *configsetObj* = getconfigset(*modelObj*,'active')

Arguments

modelObj	Model object. Enter a variable name for a model object.
NameValue	Name of the configset object.
configsetObj	Object holding the simulation-specific information.

Description

configsetObj = getconfigset(*modelObj*, '*NameValue*') returns the configuration set attached to modelObj that is named *NameValue*, to *configsetObj*.

configsetObj = getconfigset(*modelObj*) returns a vector of all attached configuration sets, to *configsetObj*.

configsetObj = getconfigset(*modelObj*,'active') retrieves the active configuration set.

A configuration set object stores simulation-specific information. A SimBiology model can contain multiple configsets with one being active at any given time. The active configuration set contains the settings that are used during the simulation.

Use the setactiveconfigset function to define the active configset. *modelObj* always contains at least one configset object with the name configured to 'default'. Additional configset objects can be added to *modelObj* with the method addconfigset.

Examples

1 Retrieve the default configset object from the modelObj.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
```

```
Configuration Settings - default (active)
    SolverType:         ode15s
    StopTime:           10.000000

  SolverOptions:
    AbsoluteTolerance:  1.000000e-006
    RelativeTolerance:  1.000000e-003

  RuntimeOptions:
    StatesToLog:        all

  CompileOptions:
    UnitConversion:     true
    DimensionalAnalysis: true
```

2 Configure the SolverType to ssa.

```
set(configsetObj, 'SolverType', 'ssa')
get(configsetObj)


          Active: 1
   CompileOptions: [1x1 SimBiology.CompileOptions]
            Name: 'default'
           Notes: ''
   RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
    SolverOptions: [1x1 SimBiology.SSASolverOptions]
       SolverType: 'ssa'
         StopTime: 10
     StopTimeType: 'simulationTime'
        TimeUnits: 'second'
             Type: 'configset'
```

See Also   addconfigset, removeconfigset, setactiveconfigset

getdata (SimData)

Purpose	Get data from SimData object array
Syntax	`[t, x, names] = getdata(simDataObj)` `[Out] = getdata(simDataObj, 'FormatValue')`

Arguments   **Output Arguments**

t	An n-by-1 vector of time points.
x	An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
names	An m-by-1 cell array of names.
Metadata	When used with the 'nummetadata' input argument, *Metadata* contains a cell array of metadata structures. The elements of *Metadata* label the columns of *x*.
Out	Data returned in the format specified in '*FormatValue*', shown in "Input Arguments" on page 4-82. Depending on the specified '*FormatValue*', *Out* contains one of the following: • Structure array • SimData object • Time series object • Combined time series object from an array of SimData objects

Input Arguments

simDataObj	SimData object. Enter a variable name for a SimData object.
FormatValue	Choose a format from the following table.

FormatValue	Description
`'num'`	Specifies the format that lets you return data in numeric arrays. This is the default when `getdata` is called with multiple output arguments.
`'nummetadata'`	Specifies the format that lets you return a cell array of metadata structures in *metadata* instead of names. The elements of *metadata* label the columns of *x*.
`'numqualnames'`	Specifies the format that lets you return qualified names in *names* to resolve ambiguities.
`'struct'`	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
`'simdata'`	Specifies the format that lets you return data in a new SimData object. This format is more useful for SimData methods other than `getdata`.

getdata (SimData)

FormatValue	Description
'ts'	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in simDataObj.
'tslumped'	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

Description   [*t, x, names*] = getdata(*simDataObj*) gets simulation time and state data from the SimData object simDataObj. When simDataObj contains more than one element, the outputs *t, x, names* are cell arrays in which each cell contains data for the corresponding element of simDataObj.

[*Out*] = getdata(*simDataObj*, '*FormatValue*') returns the data in the specified format. Valid formats are listed in "Input Arguments" on page 4-82.

Examples   **Simulating and Retrieving Data**

1 The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace and simulate the model.

```
sbioloadproject('radiodecay');
simDataObj = sbiosimulate(m1);
```

2 Get all the simulation data from the SimData object.

```
[t x names] = getdata(simDataObj);
```

Retrieving Data for Ensemble Runs

1 The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject('radiodecay');
```

2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj  = getconfigset(m1);
set(csObj, 'SolverType', 'ssa');
simDataObj = sbioensemblerun(m1, 10);
```

3 Get all the simulation data from the SimData object.

```
tsObjs = getdata(simDataObj(1:5), 'ts');
```

See Also   display, get, resample, selectselectbyname, setactiveconfigset

MATLAB function struct

getparameters (kineticlaw)

Purpose　　　Get specific parameters in kinetic law object

Syntax　　　*parameterObj* = getparameters(*kineticlawObj*)
parameterObj = getparameters(*kineticlawObj*,
　　'*ParameterVariablesValue*')

Arguments

kineticlawObj	Retrieve parameters used by the kinetic law object.
ParameterVariablesValue	Retrieve parameters used by the kinetic law object corresponding to the specified parameter in the ParameterVariables property of the kinetic law object.

Description　　*parameterObj* = getparameters(*kineticlawObj*) returns the parameters used by the kinetic law object kineticlawObj to parameterObj.

parameterObj = getparameters(*kineticlawObj*,
'*ParameterVariablesValue*') returns the parameter in the ParameterVariableNames property that corresponds to the parameter specified in the ParameterVariables property of kineticlawObj, to parameterObj. ParameterVariablesValue is the name of the parameter as it appears in the ParameterVariables property of kineticlawObj. ParameterVariablesValue can be a cell array of strings.

If you change the name of a parameter, you must configure all applicable elements such as rules that use the parameter, any user-specified ReactionRate, or the kinetic law object property ParameterVariableNames. Use the method setparameter to configure ParameterVariableNames.

Examples　　　Create a model, add a reaction, and assign the ParameterVariableNames for the reaction rate equation.

1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type
'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Add two parameter objects.

```
parameterObj1 = addparameter(kineticlawObj,'Va');
parameterObj2 = addparameter(kineticlawObj,'Ka');
```

4 The 'Henri-Michaelis-Menten' kinetic law has two parameter
variables (Vm and Km) that should to be set. To set these variables:

```
setparameter(kineticlawObj,'Vm', 'Va');
setparameter(kineticlawObj,'Km', 'Ka');
```

5 To retrieve a parameter variable:

```
parameterObj3 = getparameters(kineticlawObj, 'Vm')
```

MATLAB returns:

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
  1        Va        1


parameterObj4 = getparameters (kineticlawObj, 'Km')
```

See Also   addparameter, getspecies, setparameter

getsensmatrix (SimData)

Purpose   Get 3-D sensitivity matrix from SimData array

Syntax   [*T, R, Outputs, InputFactors*] = getsensmatrix(*simDataObj*)
[*T, R, Outputs, InputFactors*] = getsensmatrix(*simDataObj,
    OutputNames,InputFactorNames*)

Arguments

T	*T* is an m-by-1 array specifying time points for the sensitivity data in R.
R	*R* is an m-by-n-by-p array of sensitivity data with times, outputs, and input factors corresponding to its first, second, and third dimensions respectively. R(:,i,j) is the time course for the sensitivity of state *Outputs{i}* to the input factor *InputFactors{j}*.
Outputs	Name of the output factors, where output factors are the names of the states for which you want to calculate sensitivity.
InputFactors	Name of the input factors, where input factors are the names of the states with respect to which you want to calculate sensitivity.

Description   [*T, R, Outputs, InputFactors*] = getsensmatrix(*simDataObj*) gets time and sensitivity data from the SimData object (*simDataObj*).

When *simDataObj* contains more than one element, the output arguments are cell arrays in which each cell contains data for the corresponding element of *simDataObj*.

The getsensmatrix method can only return sensitivity data that is contained in the SimData object. The sensitivity data that is logged in a SimData object is set at simulation time by the configuration set used during the simulation. This is typically the model's active configuration set. See "Sensitivity Analysis" in the SimBiology User's Guide documentation for an explanation of how to set up a sensitivity calculation using the configuration set. Note in particular that the

sensitivity data R returned by getsensmatrix may be normalized, as specified at simulation time.

[*T, R, Outputs, InputFactors*] =
getsensmatrix(*simDataObj,OutputNames,InputFactorNames*) gets sensitivity data for the outputs specified by *OutputNames* and the input factors specified by *InputFactorNames*.

OutputNames and *InputFactorNames* can both be any one of the following:

- Empty array
- Single name
- Cell array of names

Pass an empty array for *OutputNames* or *InputFactorNames* to ask for sensitivity data on all output factors or input factors contained in *simDataObj*, respectively. You can also use qualified names such as '*CompartmentName.SpeciesName*' or '*ReactionName.ParameterName*' to resolve ambiguities.

Examples   This example shows how to retrieve sensitivity data from a SimData object.

1 Set up the simulation:

   a Import the radio decay model from SimBiology demos.

      modelObj = sbmlimport('radiodecay');

   b Retrieve the configset object from the modelObj.

      configsetObj = getconfigset(modelObj);

   c Specify the species for which you want sensitivity data in the SpeciesOutputs property. All model species are selected in this example.

Use the `sbioselect` function to retrieve the species objects from the model.

```
set (configsetObj.SensitivityAnalysisOptions, 'SpeciesOutputs', ...
 sbioselect(modelObj, 'Type', 'species'));
```

d Specify parameters and species with respect to which you want to calculate the sensitivities in the `ParameterInputFactors` and the `SpeciesInputFactors` properties respectively.

```
set(configsetObj.SensitivityAnalysisOptions,'ParameterInputFactors', ...
    sbioselect(modelObj, 'Type', 'parameter', 'Name', 'c'));

set(configsetObj.SensitivityAnalysisOptions,'SpeciesInputFactors', ...
    sbioselect(modelObj,'Type', 'species', 'Name', 'z'));
```

e Enable `SensitivityAnalysis`.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true)
get(configsetObj.SolverOptions, 'SensitivityAnalysis')

ans =

    1
```

f Simulate and return the results in a SimData object.

```
simDataObj = sbiosimulate(modelObj)
```

2 Extract and plot sensitivity data from the `SimData` object.

a Use `getsensmatrix` to retrieve sensitivity data.

```
[t R outs ifacs] = getsensmatrix(simDataObj);
```

b Plot sensitivity values.

```
plot(t, R(:,:,2));
legend(outs);
```

```
title(['Sensitivities of species relative to ' ifacs{2}]);
```

See Also   display, get, getdata, resample, selectbyname

MATLAB function struct

getspecies (kineticlaw)

Purpose	Get specific species in kinetic law object

Syntax

speciesObj = getspecies(*kineticlawObj*)
speciesObj = getspecies(*kineticlawObj*,
    '*SpeciesVariablesValue*')

Arguments

kineticlawObj	Retrieve species used by the kinetic law object.
SpeciesVariablesValue	Retrieve species used by the kinetic law object corresponding to the specified species in the SpeciesVariables property of the kinetic law object.

Description   *speciesObj* = getspecies(*kineticlawObj*) returns the species used by the kinetic law object kineticlawObj to speciesObj.

speciesObj = getspecies(*kineticlawObj*, '*SpeciesVariablesValue*') returns the species in the SpeciesVariableNames property to speciesObj.

SpeciesVariablesValue is the name of the species as it appears in the SpeciesVariables property of kineticlawObj. SpeciesVariablesValue can be a cell array of strings.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the name of a species, the reaction updates to use the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object SpeciesVariableNames. Use the method setspecies to configure SpeciesVariableNames.

Examples   Create a model, add a reaction, and then assign the SpeciesVariableNames for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type
  'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

  reactionObj KineticLaw property is configured to kineticlawObj.

3 The 'Henri-Michaelis-Menten' kinetic law has one species variable
  (S) that should to be set. To set this variable:

```
setspecies(kineticlawObj,'S', 'a');
```

4 Retrieve the species variable using getspecies.

```
speciesObj = getspecies (kineticlawObj, 'S')
```

  MATLAB returns:

```
SimBiology Species Array

Index:  Compartment: Name: InitialAmount: InitialAmountUnits:
   1       unnamed     a          O
```

See Also   addspecies, getparameters, setparameter, setspecies

getstoichmatrix (model)

Purpose	Get stoichiometry matrix from model object
Syntax	*M* = getstoichmatrix(*modelObj*) *M* = getstoichmatrix(*modelObj*, 'flat') [*M*,*objSpecies*] = getstoichmatrix(*modelObj*) [*M*,*objSpecies*,*objReactions*] = getstoichmatrix(*modelObj*)

Arguments

M	Adjacency matrix for *modelObj*.
modelObj	Specify the model object *modelObj*.
'flat'	Return the stoichiometry matrix for only the specified *modelObj*, not for objects contained in the *Obj*.
objSpecies	Return the list of *modelObj* species by Name property of the species. If the species are in multiple compartments, species names are qualified with the compartment name in the form compartmentName.speciesName. For example, nucleus.DNA, cytoplasm.mRNA.
objReactions	Return the list of *modelObj* reactions by the Name property of reactions.

Description getstoichmatrix returns a stoichiometry matrix for a model object.

M = getstoichmatrix(*modelObj*) returns a stoichiometry matrix for a SimBiology model object (*modelObj*) to *M*.

A stoichiometry matrix is defined by listing all reactions contained by *modelObj* column-wise and all species contained by *modelObj* row-wise in a matrix. The species of the reaction are represented in the matrix with the stoichiometric value at the location of [row of species, column

of reaction]. Reactants have negative values. Products have positive values. All other locations in the matrix are 0.

For example, if *modelObj* is a model object with two reactions with names R1 and R2 and Reaction values of 2 A + B -> 3 C and B + 3 D -> 4 A, the stoichiometry matrix would be defined as:

```
        A     B     C     D
  R1    -2    -1    3     0
  R2    4     -1    0     -3
```

M = getstoichmatrix(*modelObj*, 'flat') defines the stoichiometry matrix for only *modelObj*. *M* is the stoichiometry matrix for the reactions and species contained by *modelObj*.

[*M*,*objSpecies*] = getstoichmatrix(*modelObj*) returns the stoichiometry matrix to *M* and the species to *objSpecies*. *objSpecies* is defined by listing all Name property values of species contained by *Obj*. In the above example, *objSpecies* would be {'A', 'B', 'C', 'D'};.

[*M*,*objSpecies*,*objReactions*] = getstoichmatrix(*modelObj*) returns the stoichiometry matrix to *M* and the reactions to *objReactions*. *objReactions* is defined by listing all Name property values of reactions contained by *modelObj*. In the above example, *objReactions* would be {'R1', 'R2'}.

Examples

1 Read in a model using sbmlimport.

```
modelObj = sbmlimport('lotka.xml');
```

2 Get the stoichiometry matrix for the modelObj.

```
[M,objSpecies,objReactions] = getstoichmatrix(modelObj)
```

See Also   getadjacencymatrix

getvariant (model)

Purpose	Get variant from model
Syntax	*variantObj* = getvariant(*modelObj*) *variantObj* = getvariant(*modelObj*, '*NameValue*')

Arguments

variantObj	Variant object returned by the getvariant method.
modelObj	Model object from which to get the variant.
'*NameValue*'	Name of the variant to get from the model object *modelObj*.

Description *variantObj* = getvariant(*modelObj*) returns SimBiology variant objects contained by the SimBiology model object *modelObj* to *variantObj*.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see Variant object.

variantObj = getvariant(*modelObj*, '*NameValue*') returns the SimBiology variant object with the name *NameValue*, contained by the SimBiology model object, *modelObj*.

View properties for a variant object with the get command, and modify properties for a variant object with the set command.

Note Remember to use the addcontent method instead of using the set method on the Content property, because the set method replaces the data in the Content property whereas addcontent appends the data.

To copy a variant object to another model, use copyobj. To remove a variant object from a SimBiology model, use the delete method.

Examples   **1** Create a model containing several variants.

```
modelObj = sbiomodel('mymodel');
variantObj1 = addvariant(modelObj, 'v1');
variantObj2 = addvariant(modelObj, 'v2');
```

2 Get all variants in the model.

```
vObjs = getvariant(modelObj)

SimBiology Variant Array

   Index:  Name:              Active:
   1       v1                 false
   2       v2                 false
```

3 Get the variant object named 'v2' from the model.

```
vObjv2 = getvariant(modelObj, 'v2');
```

See Also   addvariant, removevariant

KineticLaw object

Purpose   Kinetic law information for reaction

Description   The kinetic law object holds information about the abstract kinetic law
applied to a reaction and provides a template for the reaction rate. In
the model, the SimBiology software uses the information you provide
in a fully defined kinetic law object to determine the `ReactionRate`
property in the reaction object.

When you first create a kinetic law object, you must specify the name
of the abstract kinetic law to use. The SimBiology software fills in the
`KineticLawName` property and the `Expression` property in the kinetic
law object with the name of the abstract kinetic law you specified and
the mathematical expression respectively. The software also fills in the
`ParameterVariables` property and the `SpeciesVariables` property
of the kinetic law object with the values found in the corresponding
properties of the abstract kinetic law object.

To obtain the reaction rate, you must fully define the kinetic law object:

1 In the `ParameterVariableNames` property, specify the parameters
from the model that you want to substitute in the expression
(`Expression` property).

2 In the `SpeciesVariableNames` property, specify the species from the
model that you want to substitute in the expression.

The SimBiology software substitutes in the expression, the names
of parameter variables and species variables in the order specified
in the `ParameterVariables` and `SpeciesVariables` properties
respectively.

The software then shows the substituted expression as the reaction
rate in the `ReactionRate` property of the reaction object. If the
kinetic law object is not fully defined, the `ReactionRate` property
remains ' ' (empty).

For links to kinetic law object property reference pages, see "Property
Summary" on page 4-103.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

For an explanation of how relevant properties relate to one another, see "Command Line" on page 4-99.

The following sections use a kinetic law example to show how you can fully define your kinetic law object to obtain the reaction rate in the SimBiology desktop and at the command line.

The Henri-Michaelis-Menten kinetic law is expressed as follows:

$$V_m * S / (K_m + S)$$

In the SimBiology software `Henri-Michaelis-Menten` is a built-in abstract kinetic law, where V_m and K_m are defined in the `ParameterVariables` property of the abstract kinetic law object, and `S` is defined in the `SpeciesVariables` property of the abstract kinetic law object.

SimBiology Desktop

To fully define kinetic law, in the SimBiology desktop, define the names of the species variables and parameter variables that participate in the reaction rate in the **Project Settings-Reactions** pane on the **Kinetic Law** tab. To add a reaction and set the reaction rate in the SimBiology desktop, see "Adding Reactions to a Model" in the SimBiology Getting Started Guide documentation.

Command Line

To fully define the kinetic law object at the command line, define the names of the parameters in the `ParameterVariableNames` property of the kinetic law object, and define the species names in the `SpeciesVariableNames` property of the kinetic law object. For example, to apply the `Henri-Michaelis-Menten` abstract kinetic law to a reaction

```
A -> B
where Vm = Va, Km = Ka
```
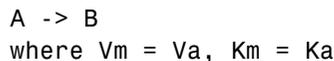
KineticLaw object

```
       and S = A
```

Define Va and Ka in the `ParameterVariableNames` property to
substitute the variables that are in the `ParameterVariables` property
(Vm and Km). Define A in the `SpeciesVariableName` property to be
used to substitute the species variable in the `SpeciesVariables`
property (S). Specify the order of the model parameters to be used for
substitution in the same order that the parameter variables are listed
in the `ParameterVariables` property. Similarly, specify species order if
more than one species variable is represented.

```
% Find the order of the parameter variables
% in the kinetic law expression.

get(kineticlawObj, 'ParameterVariables')

ans =

    'Vm'    'Km'

% Find the species variable in the
% kinetic law expression

get(kineticlawObj, 'SpeciesVariables')
ans =

    'S'

% Specify the parameters and species variables
% to be used in the substitution.
% Remember to specify order, for example Vm = Va
% Vm is listed first in 'ParameterVariables',
% therefore list Va first in 'ParameterVariableNames'.

set(kineticlawObj,'ParameterVariableNames', {'Va' 'Ka'});
set(kineticlawObj,'SpeciesVariableNames', {'A'});
```

The rate equation is assigned in the reaction object as follows:

Va*A/(Ka + A)

For a detailed procedure, see "Examples" on page 4-104.

The following table summarizes the relationships between the properties in the abstract kinetic law object and the kinetic law object in the context of the above example.

Property	Property Purpose	Abstract Kinetic Law Object	Kinetic Law Object
Name (abstract kinetic law object) KineticLawName (kinetic law object)	Name of abstract kinetic law applied to a reaction. For example: Henri-Michaelis -Menten	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only
Expression	Mathematical expression used to determine the reaction rate equation. For example: $V_{\mathrm{m}} * S / (K_m + S)$	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only; depends on abstract kinetic law applied to reaction.

KineticLaw object

Property	Property Purpose	Abstract Kinetic Law Object	Kinetic Law Object
`ParameterVariables`	Variables in `Expression` that are parameters. For example: `Vm and Km`	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only; depends on abstract kinetic law applied to reaction.
`SpeciesVariables`	Variables in `Expression` that are species. For example: `S`	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only; depends on abstract kinetic law applied to reaction.
`ParameterVariableNames`	Variables in `ReactionRate` that are parameters. For example: `Va and Ka`	Not applicable	Define these variables corresponding to `ParameterVariables`.
`SpeciesVariablesNames`	Variables in `ReactionRate` that are species. For example: `A`	Not applicable	Define these variables corresponding to `SpeciesVariables`.

Constructor Summary   addkineticlaw (reaction)   Create kinetic law object and add to reaction object

Method Summary	addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	getparameters (kineticlaw)	Get specific parameters in kinetic law object
	getspecies (kineticlaw)	Get specific species in kinetic law object
	set (any object)	Set object properties
	setparameter (kineticlaw)	Specify specific parameters in kinetic law object
	setspecies (kineticlaw)	Specify species in kinetic law object

Property Summary	Annotation	Store link to URL or file
	Expression	Expression to determine reaction rate equation
	KineticLawName	Name of kinetic law applied to reaction
	Name	Specify name of object
	Notes	HTML text describing SimBiology object
	Parameters	Array of parameter objects

KineticLaw object

ParameterVariableNames	Cell array of reaction rate parameters
ParameterVariables	Parameters in abstract kinetic law
Parent	Indicate parent object
SpeciesVariableNames	Cell array of species used in reaction rate equation
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Examples This example shows how to define the reaction rate for a reaction.

1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'A -> B');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Query the parameters and species variables defined in the kinetic law.

```
get(kineticlawObj, 'ParameterVariables')

ans =

    'Vm'    'Km'
```

```
get(kineticlawObj, 'SpeciesVariables')
ans =

    'S'
```

4 Define Va and Ka as ParameterVariableNames, which correspond
to the ParameterVariables Vm and Km. To set these variables,
first create the parameter variables as parameter objects
(parameterObj1, parameterObj2) with the names Va and Ka, and
then add them to kineticlawObj. The species object with Name A is
created when reactionObj is created and need not be redefined.

```
parameterObj1 = addparameter(kineticlawObj, 'Va');
parameterObj2 = addparameter(kineticlawObj, 'Ka');
```

5 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Va' 'Ka'});
set(kineticlawObj,'SpeciesVariableNames', {'A'});
```

6 Verify that the reaction rate is expressed correctly in the reaction
object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Va*A/(Ka+A)
```

See Also   AbstractKineticLaw object, Configset object, Model object,
Parameter object, Reaction object, Root object, Rule object,
Species object

SimBiology property Expression

Model object

Purpose	Model and component information
Description	The SimBiology model object represents a *model*, which is a collection of interrelated reactions and rules that transform, transport, and bind species. The model includes model components such as compartments, reactions, parameters, rules, and events. Each of the components is represented as a property of the model object. A model object also has a default configuration set object to define simulation settings. You can also add more configuration set objects to a model object.

See "Property Summary" on page 4-108 for links to model property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

You can retrieve top-level SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object.

Constructor Summary	`sbiomodel`	Construct model object

Method Summary	addcompartment (model, compartment)	Create compartment object
	addconfigset (model)	Create configuration set object and add to model object
	addevent (model)	Add event object to model object
	addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
	addreaction (model)	Create reaction object and add to model object

addrule (model)	Create rule object and add to model object
addvariant (model)	Add variant to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getadjacencymatrix (model)	Get adjacency matrix from model object
getconfigset (model)	Get configuration set object from model object
getstoichmatrix (model)	Get stoichiometry matrix from model object
getvariant (model)	Get variant from model
removeconfigset (model)	Remove configuration set from model
removevariant (model)	Remove variant from model
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties
setactiveconfigset (model)	Set active configuration set for model object
verify (model, variant)	Validate and verify SimBiology model

Model object

Property Summary

Annotation	Store link to URL or file
Compartments	Array of compartments in model or compartment
Events	Contain all event objects
Models	Contain all model objects
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parameters	Array of parameter objects
Parent	Indicate parent object
Reactions	Array of reaction objects
Rules	Array of rules in model object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

See Also   AbstractKineticLaw object, Configset object, KineticLaw object, Parameter object, Reaction object, Root object, Rule object, Species object

Purpose   Parameter and scope information

Description   The parameter object represents a *parameter*, which is a quantity that
                    can change or can be constant. SimBiology parameters are generally
                    used to define rate constants. You can add parameter objects to a
                    model object or a kinetic law object. The scope of a parameter depends
                    on where you add the parameter object: If you add the parameter
                    object to a model object, the parameter is available to all reactions
                    in the model and the Parent property of the parameter object is
                    SimBiology.Model. If you add the parameter object to a kinetic law
                    object, the parameter is available only to the reaction for which you are
                    using the kinetic law object and the Parent property of the parameter
                    object is SimBiology.KineticLaw.

                    See "Property Summary" on page 4-110 for links to parameter object
                    property reference pages.

                    Properties define the characteristics of an object. Use the get and set
                    commands to list object properties and change their values at the
                    command line. You can graphically change object properties in the
                    graphical user interface.

Constructor   addparameter (model, kineticlaw)   Create parameter object and add
Summary   to model or kinetic law object

Method   copyobj (any object)   Copy SimBiology object and its
Summary   children

                    delete (any object)   Delete SimBiology object

                    display (any object)   Display summary of SimBiology
                                                        object

                    get (any object)   Get object properties

Parameter object

rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

Property Summary

Annotation	Store link to URL or file
ConstantValue	Specify variable or constant parameter value
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object
Value	Assign value to parameter object
ValueUnits	Parameter value units

See Also   AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Reaction object, Root object, Rule object, Species object

Purpose   Used by PKModelDesign to create SimBiology model

Description   The PKCompartment object is used by the PKModelDesign object
                     to construct a SimBiology model for pharmacokinetic modeling.
                     PKCompartment holds the following information:

- Name of the compartment

- Dosing type

- Elimination type

- Whether the drug concentration in this compartment is reported

The PKCompartment class is a subclass of the hgsetget class which is
a subclass of the handle class. For more information on the inherited
methods, see hgsetget, and handle.

Construction   addCompartment   Add compartment to
                    (PKModelDesign)   PKModelDesign object

Method Summary get (any object)   Get object properties
                    set (any object)   Set object properties

Property Summary DosingType   Drug dosing type in compartment
                      EliminationType   Drug elimination type from
                                              compartment
                      HasResponseVariable   Compartment drug concentration
                                              reported
                      Name   Specify name of object

PKCompartment object

See Also   "Creating PK Models at the Command Line" in the SimBiology User's Guide, PKModelDesign object

Purpose   Define roles of data set columns

Description   The properties of the PKData object specify what each column in the
                   data represents. The PKData object specifies which columns in the data
                   set represent the following:

- The grouping variable

- The independent and dependent variables

- The dose

- The rate (only if infusion is the dosing type)

- The covariates

This information is used by the fitting functions, sbionlmefit and
sbionlinfit.

To create the PKData object specify:

    pkDataObject = PKData(data);

Where data is the imported data set.

The PKData class is a subclass of the hgsetget class, which is a subclass
of the handle class. For more information on the inherited methods,
see hgsetget and handle.

Construction   PKData   Create PKData object

**Method
Summary**   get (any object)   Get object properties

                   set (any object)   Set object properties

PKData object

Property Summary	CovariateLabels	Identify covariate columns in data set
	DataSet	Dataset object containing imported data
	DependentVarLabel	Identify dependent variable column in data set
	DoseLabel	Identify dose column in data set
	GroupID	Integer identifying each group in data set
	GroupLabel	Identify group column in data set
	GroupNames	Unique values from GroupLabel in data set
	IndependentVarLabel	Identify independent variable column in data set
	RateLabel	Identify rate of infusion column in data set

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, PKModelDesign object

Purpose   Helper object to construct pharmacokinetic model

Description   Use the PKModelDesign object to construct a SimBiology model for PK
                    modeling. The PKModelDesign object lets you specify the number of
                    compartments, the type of dosing, and method of elimination which you
                    then use to construct the SimBiology model object with the necessary
                    compartments, species, reactions, rules, and events.

```
pkm = PKModelDesign;
```

Use the addCompartment method to add a compartment with
a specified dosing and elimination. addCompartment adds each
subsequent compartment and connects it to the previous compartment
using a reversible reaction. This reaction models the flux between
compartments in a PK model.

The construct method uses the PKModelDesign object to create a
SimBiology model object.

The PKModelDesign class is a subclass of the hgsetget class, which is
a subclass of the handle class. For more information on the inherited
methods see hgsetget and handle.

Construction   PKModelDesign   Create PKModelDesign object

Method   addCompartment   Add compartment to
Summary   (PKModelDesign)   PKModelDesign object

                     construct (PKModelDesign)   Construct SimBiology model from
                                                       PKModelDesign object

                     get (any object)   Get object properties

                     set (any object)   Set object properties

PKModelDesign object

Property Summary

PKCompartments	Hold compartments in PK model

See Also   "Creating PK Models at the Command Line" in the SimBiology User's Guide, PKCompartment object

Purpose	Define SimBiology model components roles

Description   The PKModelMap object holds information about the dosing type, and defines which components of a SimBiology model represent the observed response, the dose, and the estimated parameters.

The PKModelMap class is a subclass of the hgsetget class which is a subclass of the handle class. For more information on the inherited methods see, hgsetget, and handle.

Construction   PKModelMap   Create PKModelMap object

Method Summary

get (any object)   Get object properties

set (any object)   Set object properties

Property Summary

Dosed   Dosed object name

DosingType   Drug dosing type in compartment

Estimated   Names of parameters to estimate

Observed   Measured response object name

See Also   "Defining Model Components for Observed Response, Dose, Dosing Type, and Parameters to be Estimated" in the SimBiology User's Guide, PKModelDesign object

Reaction object

Purpose   Options for model reactions

Description   The reaction object represents a *reaction*, which describes a
                     transformation, transport, or binding process that changes one or more
                     species. Typically, the change is the amount of a species. For example:

```
Creatine + ATP <-> ADP + phophocreatine

glucose + 2 ADP + 2 Pi -> 2 lactic acid + 2 ATP + 2 H2O
```

Spaces are required before and after species names and stoichiometric
values.

See "Property Summary" on page 4-119 for links to reaction object
property reference pages.

Properties define the characteristics of an object. Use the `get` and
`set` commands to list object properties and change their values at the
command line. You can graphically change object properties in the
graphical user interface.

**Constructor
Summary**

addreaction (model)	Create reaction object and add to model object

**Method
Summary**

addkineticlaw (reaction)	Create kinetic law object and add to reaction object
addproduct (reaction)	Add product species object to reaction object
addreactant (reaction)	Add species object as reactant to reaction object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object

display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmproduct (reaction)	Remove species object from reaction object products
rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

Property Summary

Active	Indicate object in use during simulation
Annotation	Store link to URL or file
KineticLaw	Show kinetic law used for ReactionRate
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Products	Array of reaction products
Reactants	Array of reaction reactants
Reaction	Reaction object reaction
ReactionRate	Reaction rate equation in reaction object
Reversible	Specify whether reaction is reversible or irreversible
Stoichiometry	Species coefficients in reaction

Reaction object

Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

See Also   AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Root object, Rule object, Species object

removeconfigset (model)

Purpose   Remove configuration set from model

Syntax

```
removeconfigset(modelObj, 'NameValue')
removeconfigset(modelObj, configsetObj)
```

Arguments

modelObj	Model object from which to remove the configuration set.
NameValue	Name of the configuration set.
configsetObj	Configuration set object that is to be removed from the model object.

Description   `removeconfigset(modelObj, 'NameValue')` removes the configset object with the name *NameValue* from the SimBiology model object *modelObj*. A configuration set object stores simulation-specific information. A SimBiology model can contain multiple configuration sets with one being active at any given time. The active configuration set contains the settings that are used during the simulation. *modelObj* always contains at least one configuration set object with name configured to `'default'`. You cannot remove the default configuration set from *modelObj*. If the active configuration set is removed from *modelObj*, then the default configuration set will be made active.

`removeconfigset(modelObj, configsetObj)` removes the configuration set object, *configsetObj*, from the SimBiology model, *modelObj*. The configuration set is not deleted; if you want to delete configsetObj, use the `delete` method.

If however, there is no MATLAB variable holding the configset, `removeconfigset(modelObj, 'NameValue')` removes the configset from the model and deletes it.

Examples   **1** Create a model object by importing the file `oscillator.xml` and add a configset.

```
modelObj = sbmlimport('oscillator');
```

removeconfigset (model)

```
configsetObj = addconfigset(modelObj, 'myset');
```

2 Remove the configset from `modelObj` by name or alternatively by indexing.

```
% Remove the configset with name 'myset'.
removeconfigset(modelObj, 'myset');
```

```
% Get all configset objects and remove the second.
configsetObj = getconfigset(modelObj);
removeconfigset(modelObj, configsetObj(2));
```

See Also   addconfigset, getconfigset, setactiveconfigset

removevariant (model)

Purpose   Remove variant from model

Syntax   *variantObj* = removevariant(*modelObj*, '*NameValue*')
                  variantObj = removevariant(*modelObj*, *variantObj*)

Arguments

modelObj	Specify the model object from which you want to remove the variant.
variantObj	Specify the variant object to return from the model object.

Description   *variantObj* = removevariant(*modelObj*, '*NameValue*') removes a
                  SimBiology variant object with the name *NameValue* from the model
                  object *modelObj* and returns the variant object to *variantObj*. The
                  variant object Parent property is assigned [] (empty).

                  A SimBiology variant object stores alternate values for properties on
                  a SimBiology model. For more information on variants, see Variant
                  object.

                  variantObj = removevariant(*modelObj*, *variantObj*) removes a
                  SimBiology variant object (*variantObj*) and returns the variant object
                  variantObj.

                  To view the variants stored on a model object, use the getvariant
                  method. To copy a variant object to another model, use copyobj. To add
                  a variant object to a SimBiology model, use the addvariant method.

Examples   **1** Create a model containing several variants.

```
modelObj = sbiomodel('mymodel');
variantObj1 = addvariant(modelObj, 'v1');
variantObj2 = addvariant(modelObj, 'v2');
variantObj3 = addvariant(modelObj, 'v3');
```

                  2 Remove a variant object using its name.

removevariant (model)

```
removevariant(modelObj, 'v1');
```

3 Remove a variant object using its index number.

a Get the index number of the variant in the model.

```
vObjs = getvariant(modelObj)

SimBiology Variant Array

    Index:  Name:              Active:
    1       v2                 false
    2       v3                 false
```

b Remove the variant object.

```
removevariant(modelObj, vObjs(2));
```

See Also   addvariant, getvariant

rename (compartment, parameter, species)

Purpose   Rename object and update expressions

Syntax   rename(*Obj*, '*NewNameValue*')

Arguments

Obj	Compartment, parameter, or species object.
'*NewNameValue*'	Specify the new name.

Description   rename(*Obj*, '*NewNameValue*'), changes the Name property of the object, *Obj* to *NewNameValue* and updates any expressions in the model (such as Rule or ReactionRate) to use the new name.

If the new name is already being used by another model component, the new name will be qualified to ensure that it is unique. For example if you change a species named A to K, and a parameter with the name K exists, the species will be qualified as *CompartmentName*.K to indicate that the reference is to the species. If you are referring to an object by it's qualified name, for example *CompartmentName*.A and you change the species name, the reference will contain the qualified name in it's updated form, for example, *CompartmentName*.K

When you want to change the name of a compartment, parameter, or species object, use this method instead of set. The set method only changes the Name property of the object, except for species objects where the species object's Name property and any reaction strings which refer to species are updated to use the new name.

Examples   **1** Create a model object that contains a species A in a rule.

```
m = sbiomodel('cell');
s = addspecies(m, 'A');
r = addrule(m, 'A = 4');
```

2 Rename the species to Y

```
rename(s, 'Y');
```

3 See that the rule expression is now updated.

```
r

SimBiology Rule Array

Index:     RuleType:              Rule:
1          initialAssignment      Y = 4
```

See Also   set

Purpose	Reorder component lists

Syntax	*modelObj* = reorder(*Obj*, *NewOrder*)

Arguments

Obj	Model object or compartment. Enter a variable name.
NewOrder	Object vector in the new order. If *Obj* is a model object, *NewOrder* can be an array of compartments, events, parameters, reactions or rules objects. If *Obj* is a compartment object, *NewOrder* must be an array of species objects.

Description

modelObj = reorder(*Obj*, *NewOrder*) reorders the component vector *NewOrder*, to be in the order specified.

You can use this method to reorder any of the component vectors, such as events, parameters, rules, and species. The vector of components, when reordered, must contain the same objects as the original list of objects but they can be in a different order.

Examples

1 Import a model.

```
modelObj = sbmlimport('lotka');
```

2 Display the order of the reactions in the model.

```
get(modelObj.Reactions);
```

```
SimBiology Reaction Array

Index:     Reaction:
1          x + y1 -> 2 y1 + x
2          y1 + y2 -> 2 y2
3          y2 -> z
```

3 Reverse the order of the reactions in the model.

```
reorder(modelObj, modelObj.Reactions([3 2 1]))
```

Purpose   Resample SimData object array onto new time vector

Syntax   *newSimDataObj* = resample(*simDataObj*)
newSimDataObj = resample(*simDataObj*, *timevector*)
newSimDataObj = resample(*simDataObj*, *timevector*, *method*)

Arguments

newSimDataObj	Resampled SimData object array.
simDataObj	SimData object array that you want to resample.
timevector	Real numeric array of time points onto which you want to resample the data.
method	Method to use during resampling. Can be one of the following:

- 'interp1q' — Uses the MATLAB function interp1q.

- — To use the MATLAB function interp1, specify one of the following methods:

  - 'nearest'

  - 'linear'

  - 'spline'

  - 'pchip'

  - 'cubic'

  - 'v5cubic'

- 'zoh' — specifies zero-order hold.

Description   *newSimDataObj* = resample(*simDataObj*) resamples the simulation data contained in every element of the SimData object array *simDataObj* onto a common time vector, producing a new SimData array *newSimDataObj*. By default, the common time vector is taken from the element of *simDataObj* with the earliest stopping time.

resample (SimData)

newSimDataObj = resample(*simDataObj*, *timevector*) resamples the SimData array *simDataObj* onto the time vector *timevector*. *timevector* must either be a real numeric array or the empty array [ ]. If you use an empty array, resample uses the default time vector as described above.

newSimDataObj = resample(*simDataObj*, *timevector*, *method*) uses the interpolation method specified in *method*.

If the specified *timevector* includes time points outside the time interval encompassed by one or more SimData objects in *simDataObj*, the resampling will involve extrapolation and you will see a warning. See the help for the MATLAB function corresponding to the interpolation method in use for information on how the function performs the extrapolation.

Examples

Simulating and Resampling Data

1 The project file, radiodecay.sbproj contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject('radiodecay');
simDataObj = sbiosimulate(m1);
```

2 Resample the data.

```
newSimDataObj = resample(simDataObj, [1:5], 'linear');
```

Resampling Data for Ensemble Runs

1 The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject('radiodecay');
```

2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj  = getconfigset(m1);
```

```
set(csObj, 'SolverType', 'ssa');
simDataObj = sbioensemblerun(m1, 10);
```

3 Interpolate the time steps.

```
newSimDataObj = resample(simDataObj, [1:10], 'linear');
```

4 View the time steps in the SimData object arrays.

```
newSimDataObj(1).Time
simDataObj(1).Time
```

See Also   sbioensemblerun, sbioensemblestats, sbiosimulate, SimData
object

MATLAB functions interp1, interp1q

reset (root)

Purpose   Delete all model objects from root object

Syntax   reset(sbioroot)

Description   reset(sbioroot) deletes all SimBiology model objects contained by the SimBiology root. The SimBiology root object is returned with the method sbioroot. This call is equivalent to sbioreset.

The SimBiology root object contains a list of SimBiology model objects, available units, unit prefixes, and abstract kinetic law objects. A SimBiology model object has its Parent property set to the SimBiology root object.

To add an abstract kinetic law to the SimBiology root user-defined library, use the sbioaddtolibrary function. To add a unit to the SimBiology root user-defined library, use the function sbioregisterunit. To add a unit prefix to the SimBiology root user-defined library, use the function sbioregisterunitprefix.

Examples   **1** Query sbioroot, which has two model objects.

```
sbioroot

    SimBiology Root Contains:

    Models:                         2
    Builtin Abstract Kinetic Laws:  3
    User Abstract Kinetic Laws:     1
    Builtin Units:                  54
    User Units:                     0
    Builtin Unit Prefixes:          13
    User Unit Prefixes:             0
```

2 Call reset.

```
sbioroot

    SimBiology Root Contains:
```

```
Models:                        0
Builtin Abstract Kinetic Laws: 3
User Abstract Kinetic Laws:    1
Builtin Units:                 54
User Units:                    0
Builtin Unit Prefixes:         13
User Unit Prefixes:            0
```

See Also   sbioaddtolibrary, sbiohelp, sbioregisterunit, sbioregisterunitprefix, sbioreset, sbioroot

rmcontent (variant)

Purpose   Remove contents from variant object

Syntax   rmcontent(*variantObj*, *contents*)
rmcontent(*variantObj*, *idx*)

Arguments

variantObj	Specify the variant object from which you want to remove data. The Content property is modified to remove the new data.
contents	Specify the data you want to remove from a variant object. Contents can either be a cell array or an array of cell arrays. A valid cell array should have the form {'*Type*', '*Name*', '*PropertyName*', *PropertyValue*}, where *PropertyValue* is the new value to be applied for the *PropertyName*. Valid *Type*, *Name*, and *PropertyName* values are as follows.

'Type'	*'Name'*	*'PropertyName'*
'species'	Name of the species. If there are multiple species in the model with the same name, specify the species as [compartmentName.speciesName], where compartmentName is the name of the compartment containing the species.	'InitialAmount'
'parameter'	If the parameter scope is a model, specify the parameter name. If the parameter scope is a kinetic law, specify [reactionName.parameterName].	'Value'
'compartment'	Name of the compartment.	'Capacity'

idx	Specify the ContentIndex or indices of the data to be removed. To display the ContentIndex, enter the object name and press **Enter**.

rmcontent (variant)

Description   rmcontent(*variantObj*, *contents*) removes the data stored in the variable *contents* from the variant object (*variantObj*).

rmcontent(*variantObj*, *idx*) removes the data specified by the indices *idx* (also called ContentIndex) from the Content property of the variant object.

Examples   **1** Create a model containing three species in one compartment.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
A = addspecies(compObj, 'A');
B = addspecies(compObj, 'B');
C = addspecies(compObj, 'C');
```

2 Add a variant object that varies the species' InitialAmount property.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {{'species','A', 'InitialAmount', 5}, ...
{'species', 'B', 'InitialAmount', 10}, ...
{ 'species', 'C', 'InitialAmount', 15}});% Display the variant
variantObj
```

```
SimBiology Variant - v1 (inactive)

   ContentIndex:    Type:      Name:      Property:        Value:
   1                species    A          InitialAmount    5
   2                species    B          InitialAmount    10
   3                species    C          InitialAmount    15
```

3 Use the ContentIndex number to remove a species from the Content property of the variant object.

```
rmcontent(variantObj, 2);
variantObj
```

```
SimBiology Variant - v1 (inactive)
```

rmcontent (variant)

```
ContentIndex:     Type:        Name:       Property:          Value:
1                 species      A           InitialAmount      5
2                 species      C           InitialAmount      15
```

4 (Alternatively) Remove a species from the contents of the variant object using detailed reference to the species.

```
rmcontent(variantObj, {'species','A', 'InitialAmount', 5});
% Display variant object
variantObj
SimBiology Variant - v1 (inactive)

ContentIndex:     Type:        Name:       Property:          Value:
1                 species      C           InitialAmount      15
```

See Also   addvariant, rmcontent, sbiovariant

Purpose   Remove species object from reaction object products

Syntax   rmproduct(*reactionObj*, *SpeciesName*)
rmproduct(*reactionObj*, *speciesObj*)

Arguments

reactionObj	Reaction object.
SpeciesName	Name for a model object. Enter a species name or a cell array of species names.
speciesObj	Species object. Enter a species object or an array of species objects.

Description   rmproduct(*reactionObj*, *SpeciesName*), in a reaction object
(reactionObj), removes a species object with a specified name
(SpeciesName) from the property Products, removes the species name
from the property Reaction, and updates the property Stoichiometry
to exclude the species coefficient.

rmproduct(*reactionObj*, *speciesObj*) removes a species object as
described above using a MATLAB variable for a species object.

The species object is not removed from the parent model property
Species. If the species object is no longer used by any reaction, you can
use the function delete to remove it from the parent object.

If one of the species specified does not exist as a product, a warning
is returned.

Examples   **Example 1**

This example shows how to remove a product that was previously added
to a reaction. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP -> creatine + ATP + Pi');
rmproduct(reactionObj, 'Pi')
```

rmproduct (reaction)

```
SimBiology Reaction Array

Index:  Reaction:
  1      Phosphocreatine + ADP -> creatine + ATP
```

Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A -> B + C');
reactionObj.Reaction
 ans =
    A -> B + C

rmproduct(reactionObj, modelObj.Species(2));
reactionObj.Reaction
  ans =
    A -> C
```

See Also   rmreactant

Purpose	Remove species object from reaction object reactants

Syntax   rmreactant(*reactionObj*, *SpeciesName*)
                 rmreactant(*reactionObj*, *speciesObj*)

Arguments

reactionObj	Reaction object.
SpeciesName	Name for a species object. Enter a species name or a cell array of species names.
speciesObj	Species object. Enter a species object or an array of species objects.

Description   rmreactant(*reactionObj*, *SpeciesName*), in a reaction object
                 (reactionObj), removes a species object with a specified name
                 (SpeciesName) from the property Reactants, removes the species name
                 from the property Reaction, and updates the property Stoichiometry
                 to exclude the species coefficient.

                 rmreactant(*reactionObj*, *speciesObj*) removes a species object as
                 described above using a MATLAB variable for a species object, or a
                 model index for a species object.

                 The species object is not removed from the parent model property
                 Species. If the species object is no longer used by any reaction, you can
                 use the method delete to remove it from the parent object.

                 If one of the species specified does not exist as a reactant, a warning
                 is returned.

Examples   **Example 1**

               This example shows how to remove a reactant that was added to a
               reaction by mistake. You can remove the species object using the
               species name.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP + Pi -> creatine + ATP');
```

```
rmreactant(reactionObj, 'Pi')

SimBiology Reaction Array

 Index:     Reaction:
   1        Phosphocreatine + ADP -> creatine + ATP
```

Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A -> B + C');

reactionObj.Reaction
ans =
    A + B -> C

rmreactant(reactionObj, modelObj.Species(1));
reactionObj.Reaction

ans =
    A -> C
```

See Also   delete, rmproduct

Purpose   Hold models, unit libraries, and abstract kinetic law libraries

Description   The SimBiology root object contains a list of the top-level SimBiology model objects and SimBiology libraries. The components that the libraries contain are: all available units, unit prefixes, and available abstract kinetic law objects. There are two types of libraries: one contains components that are built in (`BuiltinLibrary`), and the other contains components that are user defined (`UserdefinedLibrary`).

You can retrieve top-level SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object.

See "Property Summary" on page 4-142 for links to root object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

Constructor Summary

sbioroot	Return SimBiology root object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
get (any object)	Get object properties
reset (root)	Delete all model objects from root object
set (any object)	Set object properties

Root object

Property Summary

`BuiltInLibrary`	Library of built-in components
`Models`	Contain all model objects
`Type`	Display top-level SimBiology object type
`UserDefinedLibrary`	Library of user-defined components

See Also   `AbstractKineticLaw object`, `Configset object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Rule object`, `Species object`

Purpose	Hold rule for species and parameters

Description The SimBiology rule object represents a *rule*, which is a mathematical expression that modifies a species amount or a parameter value. For a description of the types of SimBiology rules, see RuleType.

See "Property Summary" on page 4-143 for links to rule property reference pages.

Properties define the characteristics of an object. Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addrule (model)	Create rule object and add to model object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

Active	Indicate object in use during simulation
Annotation	Store link to URL or file
Name	Specify name of object

Rule object

Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Rule	Specify species and parameter interactions
RuleType	Specify type of rule for rule object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

See Also AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Species object

select (SimData)

Purpose	Select data from SimData object

Syntax
```
[t,x,names] = select(simDataObj, Query)
[Out] = select(simDataObj, Query, 'Format', 'FormatValue')
```

Arguments

Output Arguments

t	An n-by-1 vector of time points.
x	An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
names	An m-by-1 cell array of names.
Out	Data returned in the format specified in '*FormatValue*', shown in "Input Arguments" on page 4-145. Depending on the specified '*FormatValue*', *Out* contains one of the following:

- Structure array
- SimData object
- Time series object
- Combined time series object from an array of SimData objects

Input Arguments

simDataObj	SimData object array. Enter a variable name for a SimData object.

select (SimData)

Query	A cell array of arguments consisting of some combination of property name/property value pairs and/or `'Where'` clauses. For a more complete description of the query syntax, including `'Where'` clauses and their supported condition types, see `sbioselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a SimData object in a query. These include `'Type'`, `'Name'`, `'Units'`, `'Compartment'` (species only), or `'Reaction'` (parameter only).
FormatValue	Choose a format from the following table.

FormatValue	Description
`'num'`	Specifies the format that lets you return data in numeric arrays. This is the default when `select` is called with multiple output arguments.
`'nummetadata'`	Specifies the format that lets you return a cell array of metadata structures in *metadata* instead of names. The elements of *metadata* label the columns of *x*.
`'numqualnames'`	Specifies the format that lets you return qualified names in *names* to resolve ambiguities.
`'struct'`	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
`'simdata'`	Specifies the format that lets you return data in a new SimData object. This is the default format when `select` is called with zero or one output argument.
`'ts'`	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in `simDataObj`.
`'tslumped'`	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

Description   [*t*,*x*,*names*] = select(*simDataObj*, *Query*) returns simulation time and state data from the SimData object (simDataObj) that matches the query argument *Query*.

In a SimData object *simDataObj*, the columns of the data matrix simDataObj.Data are labeled by the cell array of metadata structures given by simDataObj.DataInfo. The select method enables you to pick out columns of the data matrix based on their metadata labels. For example, to extract data for all parameters logged in a SimData object *simDataObj*, use the syntax [*t, x, names*] = select (*simDataObj*, {'Type', 'parameter'}].

[*Out*] = select(*simDataObj*, *Query*, 'Format', '*FormatValue*') returns the data in the specified format. Valid formats are listed in "Input Arguments" on page 4-145.

Examples   This example shows how to extract data of interest from your simulation data with the select method.

1 The project file radiodecay.sbproj contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject gprotein_norules m1
```

2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj  = getconfigset(m1);
set(csObj, 'SolverType', 'ssa');
simDataObj = sbioensemblerun(m1, 10);
```

3 Select all species data logged in the SimData array sdarray.

```
[t x n] = select(simDataObj, {'Type','species'});
```

4 Select data for the parameters with name 'Kd' and return the results in a new SimData object array.

```
 newsd = select(simDataObj, {'Type','parameter','name', 'Kd'});
```

5 This selects all data from simDataObj with a name that matches the pattern 'G' and returns time series objects.

```
ts = select(simDataObj, {'Where','Name','regexp','G'}, ...
                        'Format','ts');
```

See Also   getdata, sbioselect, sbiosimulate, selectbyname, Simdata object

selectbyname (SimData)

Purpose	Select data by name from SimData object array

Syntax

```
[t,x,n] = selectbyname(simDataObj, 'NameValue')
Out = selectbyname(simDataObj, NameValue, 'Format', Format)
```

Arguments

Output Arguments

t	An n-by-1 vector of time points.
x	An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
n	An m-by-1 cell array of names.
Out	Data returned in the format as specified in '*FormatValue*', shown in "Input Arguments" on page 4-149. Depending on the specified '*FormatValue*', *Out* contains one of the following:

- Structure array
- SimData object
- Time series object
- Combined time series object from an array of SimData objects

Input Arguments

simDataObj	SimData object array. Enter a variable name for a SimData object.
NameValue	Names of the states for which you want to select data from *simDataObj*. Must be either a string or a cell array of strings.

Query	A cell array of arguments consisting of some combination of property name/property value pairs and/or `'Where'` clauses. For a more complete description of the query syntax, including `'Where'` clauses and their supported condition types, see `sbioselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a SimData object. These include `'Type'`, `'Name'`, `'Units'`, `'Compartment'` (species only), or `'Reaction'` (parameter only).
FormatValue	Choose a format from the following table.

FormatValue	Description
`'num'`	Specifies the format that lets you return data in numeric arrays. This is the default when `select` is called with multiple output arguments.
`'nummetadata'`	Specifies the format that lets you return a cell array of metadata structures in *metadata* instead of names. The elements of *metadata* label the columns of *x*.
`'numqualnames'`	Specifies the format that lets you return qualified names in *names* to resolve ambiguities.
`'struct'`	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
`'simdata'`	Specifies the format that lets you return data in a new SimData object. This is the default format when `select` is called with zero or one output argument.

FormatValue	Description
'ts'	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in simDataObj.
'tslumped'	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

Description

The selectbyname method allows you to select data from a SimData object array by name. [*t,x,n*] = selectbyname(*simDataObj*, '*NameValue*') returns time and state data from the SimData object *simDataObj* for states with names '*NameValue*'.

In a SimData object *simDataObj*, the names labeling the columns of the data matrix *simDataObj*.Data are given by *simDataObj*.DataNames. A name specified in '*NameValue*' can match more than one data column, for example, when *simDataObj* contains data for a species and parameter both named 'k'. To resolve ambiguities, use qualified names in '*NameValue*', such as '*CompartmentName.SpeciesName*' or '*ReactionName.ParameterName*'. selectbyname returns qualified names in the output argument *names* when there are ambiguities.

Out = selectbyname(*simDataObj*, *NameValue*, 'Format', *Format*) returns the data in the specified format. Valid formats are listed in "Input Arguments" on page 4-149.

Examples

```
% Get data for the species 'glucose' from the simdata array sdarray.
[t x n] = selectbyname(sdarray,'glucose');

% Get data for multiple states and return the results in a struct array.
s = selectbyname(sdarray,{'RexGFP';'nuc.GFP';'cytosol.GFP'},...
                         'Format','struct');
```

See Also getdata, sbioselect, sbiosimulate

set (any object)

Purpose   Set object properties

Syntax   set(*Obj*, *'PropertyName'*, *PropertyValue*)
                  set(*Obj*, *'PropertyName1'*, *PropertyValue1*, *'PropertyName2'*,
                     PropertyValue2...)

Arguments

Obj	Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, PKCompartment, PKData, PKModelDesign PKModelMap, reaction, rule, SimData, species, or variant object.
'PropertyName'	Name of the property to set.
PropertyValue	Specify the value to set. Property values depend on the property being set. See the reference page for an object property for values that can be specified.

Description   set(*Obj*, *'PropertyName'*, *PropertyValue*) sets the property
                  'PropertyName' of the object *Obj*, to *PropertyValue*.

                  set(*Obj*, *'PropertyName1'*, *PropertyValue1*, *'PropertyName2'*,
                  PropertyValue2...) sets the properties *'PropertyName1'* and
                  'PropertyName2' to *PropertyValue1* and *PropertyValue2*
                  respectively, and so on in sequence. You can specify multiple
                  PropertyName, *PropertyValue* pairs.

                  When you want to change the name of a compartment, parameter,
                  or species object, use the rename method instead of set. The rename
                  method allows you to change the name and update the expressions in
                  which these components are used.

Examples   **1** Create a model object.

                       modelObj = sbiomodel ('my_model');

                  2 Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

3 Set the `ConstantValue` property of the parameter object to `false`
and verify.

MATLAB returns 1 for `true` and 0 for `false`.

```
set (parameterObj, 'ConstantValue', false);
get(parameterObj, 'ConstantValue')
```

MATLAB returns

```
ans =

     0
```

See Also   get , rename, setactiveconfigset

setactiveconfigset (model)

Purpose   Set active configuration set for model object

Syntax   *configsetObj* = setactiveconfigset(*modelObj*, '*NameValue*')
configsetObj2 = setactiveconfigset(*modelObj*, *configsetObj1*)

Description   *configsetObj* = setactiveconfigset(*modelObj*, '*NameValue*') sets the configuration set *NameValue* to be the active configuration set for the model *modelObj* and returns to *configsetObj*.

configsetObj2 = setactiveconfigset(*modelObj*, *configsetObj1*) sets the configset *configsetObj1* to be the active configset for *modelObj* and returns to *configsetObj2*. Any change in one of these two configset objects *configsetObj1* and *configsetObj2* is reflected in the other. To copy over a configset object from one model object to another, use the copyobj method.

The active configuration set contains the settings that are be used during a simulation. A default configuration set is attached to any new model.

Examples   **1** Create a model object by importing the file oscillator.xml and add a configset that simulates for 3000 seconds.

```
modelObj  = sbmlimport('oscillator');
configsetObj = addconfigset(modelObj, 'myset');
```

2 Configure the configsetObj StopTime to 3000.

```
set(configsetObj, 'StopTime', 3000)
get(configsetObj)


            Active: 0
     CompileOptions: [1x1 SimBiology.CompileOptions]
              Name: 'myset'
             Notes: ''
     RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
      SolverOptions: [1x1 SimBiology.ODESolverOptions]
```

```
               SolverType: 'ode15s'
                 StopTime: 3000
             StopTimeType: 'simulationTime'
                TimeUnits: 'second'
                     Type: 'configset'
```

3 Set the new configset to be active, simulate the model using the new configset, and plot the result.

```
setactiveconfigset(modelObj, configsetObj);
[t,x] = sbiosimulate(modelObj);
plot (t,x)
```

See Also   addconfigset, getconfigset, removeconfigset

setparameter (kineticlaw)

Purpose	Specify specific parameters in kinetic law object
Syntax	setparameter(*kineticlawObj*, 'ParameterVariablesValue', 'ParameterVariableNamesValue')

Arguments

ParameterVariableValue	Specify the value of the parameter variable in the kinetic law object.
ParameterVariableNamesValue	Specify the parameter name with which to configure the parameter variable in the kinetic law object. Determines parameters in the ReactionRate equation.

Description Configure ParameterVariableNames in the kinetic law object.

setparameter(*kineticlawObj*, 'ParameterVariablesValue', 'ParameterVariableNamesValue') configures the ParameterVariableNames property of the kinetic law object (kineticlawObj). ParameterVariableValue corresponds to one of the strings in kineticlawObj ParameterVariables property. The corresponding element in the kineticlawObjParameterVariableNames property is configured to ParameterVariableNamesValue. For example, if ParameterVariables is {'Vm', 'Km'} and ParameterVariablesValue is specified as Vm, then the first element of the ParameterVariableNames cell array is configured to ParameterVariableNamesValue.

Examples Create a model, add a reaction, and then assign the ParameterVariableNames for the reaction rate equation.

1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
```

```
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of the type
`'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

`reactionObj` KineticLaw property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has two parameter
variables (`Vm` and `Km`) that should be set. To set these variables:

```
setparameter(kineticlawObj,'Vm', 'Va');
setparameter(kineticlawObj,'Km', 'Ka');
```

4 Verify that the parameter variables are correct.

```
get (kineticlawObj, 'ParameterVariableNames')
```

MATLAB returns:

```
ans =

    'Va'     'Ka'
```

See Also   addparameter, getspecies, setspecies

setspecies (kineticlaw)

Purpose	Specify species in kinetic law object

Syntax

```
setspecies(kineticlawObj, 'SpeciesVariablesValue',
    'SpeciesVariableNamesValue')
```

Arguments

SpeciesVariablesValue	Specify the species variable in the kinetic law object.
SpeciesVariableNamesValue	Specify the species name with which to configure the species variable in the kinetic law object. Determines the species in the ReactionRate equation.

Description

setspecies configures the kinetic law object SpeciesVariableNames property.

setspecies(kineticlawObj, 'SpeciesVariablesValue', 'SpeciesVariableNamesValue') configures the SpeciesVariableNames property of the kinetic law object, kineticlawObj. SpeciesVariablesValue corresponds to one of the strings in the SpeciesVariables property of kineticlawObj. The corresponding element in kineticlawObj SpeciesVariableNames property is configured to SpeciesVariableNamesValue.

For example, if SpeciesVariables are {'S', 'S1'} and SpeciesVariablesValue is specified as S1, the first element of the SpeciesVariableNames cell array is configured to SpeciesVariableNamesValue.

Examples

Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj KineticLaw property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has one species variable (S) that should be set. To set this variable:

```
setspecies(kineticlawObj,'S', 'a');
```

4 Verify that the species variable is correct.

```
get (kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =

'a'
```

See Also   `addparameter`, `getspecies`, `setparameter`

SimData object

Purpose

Simulation data storage

Description

The SimBiology `SimData` object contains simulation data. The output from the `sbiosimulate` function, is stored in the `SimData` object which holds time and state data as well as metadata, such as the types and names for the logged states or the configuration set used during simulation.

You can also store data from multiple simulation runs as an array of SimData objects. Thus, the output of `sbioensemblerun` is an array of SimData objects. You can use any SimData method on an array of SimData objects.

You can access the time, data, and metadata stored in the `SimData` object through the properties in "Property Summary" on page 4-161. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line.

Methods you can use to query the SimData object are listed in "Method Summary" on page 4-160.

Constructor Summary

sbioensemblerun	Multiple stochastic ensemble runs of SimBiology model
sbiosimulate	Simulate model object

Method Summary

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getdata (SimData)	Get data from `SimData` object array

getsensmatrix (SimData)	Get 3-D sensitivity matrix from SimData array
resample (SimData)	Resample SimData object array onto new time vector
select (SimData)	Select data from SimData object
selectbyname (SimData)	Select data by name from SimData object array
set (any object)	Set object properties

Property Summary

Data	Store simulation data
DataCount	Numbers of species, parameters, sensitivities
DataInfo	Metadata labels for simulation data
DataNames	Show names in SimData object
ModelName	Name of model simulated
Name	Specify name of object
Notes	HTML text describing SimBiology object
RunInfo	Information about simulation
Time	Show simulation time steps
TimeUnits	Show stop time units for simulation
UserData	Specify data to associate with object

SimData object

See Also   AbstractKineticLaw object, KineticLaw object, Model object,
Parameter object, Reaction object, Root object, Rule object,
Species object

Purpose   Options for compartment species

Description   The SimBiology species object represents a *species*, which is a chemical or entity that participates in reactions, for example, DNA, ATP, `Pi`, `creatine`, `G-Protein`, or `Mitogen-Activated Protein Kinase` `(MAPK)`. Species amounts can vary or remain constant during a simulation.

To add species that participate in reactions, add the reaction to the model. The process of adding the reaction to the model creates a compartment object (*unnamed*) and the necessary species objects.

Alternatively, create and add a species object to a compartment object, using the addspecies method at the command line. The SimBiology desktop adds a default compartment (*unnamed*) for you and you can add a species in the **Species** pane. In the **Project Explorer**, expand **Compartment** and click **Species** to open the **Species** pane.

See "Property Summary" on page 4-164 for links to species property reference pages. Properties define the characteristics of an object. Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addspecies (compartment)	Create species object and add to compartment object

Method Summary   Methods for species objects

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties

Species object

rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

Property Summary

Properties for species objects

Annotation	Store link to URL or file
BoundaryCondition	Indicate species boundary condition
ConstantAmount	Specify variable or constant species amount
InitialAmount	Species initial amount
InitialAmountUnits	Species initial amount units
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

See Also   Compartment object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object

Unit object

Notes	HTML text describing SimBiology object
Offset	Unit composition modifier
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

See Also   AbstractKineticLaw object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object, UnitPrefix object

Purpose   Hold information about user-defined unit prefix

Description   The SimBiology unit prefix object holds information about user-defined unit prefixes. To create a unit prefix, create the unit prefix object and add the unit prefix to the library using the sbioaddtolibrary function.

Use the unit prefix object property Exponent, to specify the exponent of your unit prefix. See "Property Summary" on page 4-167 for links to unit prefix object property reference pages.

Properties define the characteristics of an object. Use the get and set commands to list object properties and change their values at the command line. You can graphically change unit prefix object properties using the **Unit Manager** in the SimBiology desktop.

Constructor Summary

sbiounitprefix	Create user-defined unit prefix

Method Summary

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

Annotation	Store link to URL or file
Exponent	Exponent value of unit prefix
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object

UnitPrefix object

`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

See Also   `AbstractKineticLaw object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`, `Species object`, `Unit object`

Purpose   Store alternate component values

Description The SimBiology variant object stores the names and values of model
components and allows you to use the values stored in a variant object
as the alternate value to be applied during a simulation. You can store
values for species `InitialAmount`, parameter `Value`, and compartment
`Capacity` in a variant object. Simulating using a variant does not
alter the model component values. The values specified in the variant
temporarily apply during simulation.

Using one or more variant objects associated with a model allows you to
evaluate model behavior during simulation, with different values for
the various model components without having to search and replace
these values, or having to create additional models with these values. If
you determine that the values in a variant object accurately define your
model, you can permanently replace the values in your model with the
values stored in the variant object, using the `commit` method.

To use a variant in a simulation you must add the variant object to the
model object and set the `Active` property of the variant to true. Set the
`Active` property to true if you always want the variant to be applied
before simulating the model. You can also enter the variant object as an
argument to `sbiosimulate`; this applies the variant only for the current
simulation and supersedes any active variant objects on the model.

When there are multiple active variant objects on a model, if there are
duplicate specifications for a property's value, the last occurrence for
the property value in the array of variants, is used during simulation.
You can find out which variant is applied last by looking at the indices
of the variant objects stored on the model. Similarly, in the `Content`
property, if there are duplicate specifications for a property's value,
the last occurrence for the property in the `Content` property, is used
during simulation.

Use the `addcontent` method to append contents to a variant object.

See "Property Summary" on page 4-170 for links to species property
reference pages. Properties define the characteristics of an object.
Use the `get` and `set` commands to list object properties and change

Variant object

their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary	sbiovariant	Construct variant object

Method Summary	Methods for variant objects	
	addcontent (variant)	Append content to variant object
	commit (variant)	Commit variant contents to model
	copyobj (any object)	Copy SimBiology object and its children
	delete (any object)	Delete SimBiology object
	display (any object)	Display summary of SimBiology object
	get (any object)	Get object properties
	rmcontent (variant)	Remove contents from variant object
	set (any object)	Set object properties
	verify (model, variant)	Validate and verify SimBiology model

Property Summary	Properties for variant objects	
	Active	Indicate object in use during simulation
	Annotation	Store link to URL or file
	Content	Contents of variant object
	Name	Specify name of object

`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

See Also   `Compartment object`, `Configset object`, `Model object`, `Parameter object`, `Species object`

`sbiosimulate`

verify (model, variant)

Purpose	Validate and verify SimBiology model
Syntax	verify(*modelObj*) verify(*modelObj*, *configsetObj*) verify(*modelObj*, *variantObj*) verify(*modelObj*, *configsetObj*, *variantObj*)

Description   verify(*modelObj*) performs checks on a model object (*modelObj*) to verify that you can simulate the model. This method generates stacked errors and warnings if any problems are found. To see the entire list of errors and warnings, use sbiolasterror and sbiolastwarning. The verify method uses the active configuration set for verification.

verify(*modelObj*, *configsetObj*) performs checks on the specified configuration set object (*configsetObj*) in conjunction with the model object (*modelObj*) to verify that you can simulate the model.

verify(*modelObj*, *variantObj*) performs checks on the variant object (*variantObj*) in conjunction with the model object (*modelObj*) to verify that you can simulate the model. The model object is required for the verification of the variant object.

verify(*modelObj*, *configsetObj*, *variantObj*) performs checks on the configuration set object *configsetObj*, and the variant object *variantObj* in conjunction with the model object (*modelObj*) to verify that you can simulate the model.

Verification in the SimBiology GUI

While you are building your model in the SimBiology desktop, you can click Verify at any time to generate a list of any errors and warnings in the model. The errors and warnings appear in the **Errors and Warnings** pane.

Examples
```
modelObj = sbmlimport('radiodecay.xml');
verify(modelObj);
```

See Also   sbiolasterror, sbiolastwarning

5

Property Reference

Abstract Kinetic Law

`Annotation`	Store link to URL or file
`Expression`	Expression to determine reaction rate equation
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`ParameterVariables`	Parameters in abstract kinetic law
`Parent`	Indicate parent object
`SpeciesVariables`	Species in abstract kinetic law
`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

Compartments

Annotation	Store link to URL or file
Capacity	Compartment capacity
CapacityUnits	Compartment capacity units
Compartments	Array of compartments in model or compartment
ConstantCapacity	Specify variable or constant compartment capacity
Name	Specify name of object
Notes	HTML text describing SimBiology object
Owner	Owning compartment
Parent	Indicate parent object
Species	Array of species in compartment object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Configuration Sets

`Active`	Indicate object in use during simulation
`CompileOptions`	Dimensional analysis and unit conversion options
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`RuntimeOptions`	Options for logged species
`SensitivityAnalysisOptions`	Specify sensitivity analysis options
`SolverOptions`	Specify model solver options
`SolverType`	Select solver type for simulation
`StopTime`	Set stop time for simulation
`StopTimeType`	Specify type of stop time for simulation
`TimeUnits`	Show stop time units for simulation
`Type`	Display top-level SimBiology object type

Events

Properties for event objects

`Active`	Indicate object in use during simulation
`Annotation`	Store link to URL or file
`EventFcns`	Event expression
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Tag`	Specify label for SimBiology object
`Trigger`	Event trigger
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

Kinetic Laws

`Annotation`	Store link to URL or file
`Expression`	Expression to determine reaction rate equation
`KineticLawName`	Name of kinetic law applied to reaction
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parameters`	Array of parameter objects
`ParameterVariableNames`	Cell array of reaction rate parameters
`ParameterVariables`	Parameters in abstract kinetic law
`Parent`	Indicate parent object
`SpeciesVariableNames`	Cell array of species used in reaction rate equation
`SpeciesVariables`	Species in abstract kinetic law
`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

Models

`Annotation`	Store link to URL or file
`Compartments`	Array of compartments in model or compartment
`Events`	Contain all event objects
`Models`	Contain all model objects
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parameters`	Array of parameter objects
`Parent`	Indicate parent object
`Reactions`	Array of reaction objects
`Rules`	Array of rules in model object
`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

Parameters

Annotation	Store link to URL or file
ConstantValue	Specify variable or constant parameter value
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object
Value	Assign value to parameter object
ValueUnits	Parameter value units

PKCompartment

DosingType	Drug dosing type in compartment
EliminationType	Drug elimination type from compartment
HasResponseVariable	Compartment drug concentration reported
Name	Specify name of object

PKData

CovariateLabels	Identify covariate columns in data set
DataSet	`Dataset` object containing imported data
DependentVarLabel	Identify dependent variable column in data set
DoseLabel	Identify dose column in data set
GroupID	Integer identifying each group in data set
GroupLabel	Identify group column in data set
GroupNames	Unique values from `GroupLabel` in data set
IndependentVarLabel	Identify independent variable column in data set
RateLabel	Identify rate of infusion column in data set

PKModelDesign

PKCompartments   Hold compartments in PK model

PKModelMap

`Dosed`	Dosed object name
`DosingType`	Drug dosing type in compartment
`Estimated`	Names of parameters to estimate
`Observed`	Measured response object name

Reactions

`Active`	Indicate object in use during simulation
`Annotation`	Store link to URL or file
`KineticLaw`	Show kinetic law used for `ReactionRate`
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Products`	Array of reaction products
`Reactants`	Array of reaction reactants
`Reaction`	Reaction object reaction
`ReactionRate`	Reaction rate equation in reaction object
`Reversible`	Specify whether reaction is reversible or irreversible
`Stoichiometry`	Species coefficients in reaction
`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

Root

`BuiltInLibrary`	Library of built-in components
`Models`	Contain all model objects
`Type`	Display top-level SimBiology object type
`UserDefinedLibrary`	Library of user-defined components

Rules

`Active`	Indicate object in use during simulation
`Annotation`	Store link to URL or file
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Rule`	Specify species and parameter interactions
`RuleType`	Specify type of rule for rule object
`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

SimData

`Data`	Store simulation data
`DataCount`	Numbers of species, parameters, sensitivities
`DataInfo`	Metadata labels for simulation data
`DataNames`	Show names in SimData object
`ModelName`	Name of model simulated
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`RunInfo`	Information about simulation
`Time`	Show simulation time steps
`TimeUnits`	Show stop time units for simulation
`UserData`	Specify data to associate with object

Species

Annotation	Store link to URL or file
BoundaryCondition	Indicate species boundary condition
ConstantAmount	Specify variable or constant species amount
InitialAmount	Species initial amount
InitialAmountUnits	Species initial amount units
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Unit

Annotation	Store link to URL or file
Composition	Unit composition
Multiplier	Relationship between defined unit and base unit
Name	Specify name of object
Notes	HTML text describing SimBiology object
Offset	Unit composition modifier
Parent	Indicate parent object
Tag	Specify label for SimBiology object

Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Unit Prefix

Annotation	Store link to URL or file
Exponent	Exponent value of unit prefix
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Variant

Active	Indicate object in use during simulation
Annotation	Store link to URL or file
Content	Contents of variant object
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object

`Tag`	Specify label for SimBiology object
`Type`	Display top-level SimBiology object type
`UserData`	Specify data to associate with object

Using Object Properties

Command-line syntax for entering and retrieving property values.

Entering Property Values

Enter or change a single property value using dot notation.

```
ObjectName.PropertyName = PropertyValue
```

Enter or change one or more property values using the MATLAB function set.

```
set(ObjectName, 'Propertyname', PropertyValue, ...)
```

Retrieving Property Values

Retrieve a single property value using dot notation.

```
PropertyValue = ObjectName.PropertyName
```

Retrieve one or more property values using the MATLAB function `get`.

```
PropertyValue(s) = get(ObjectName, 'PropertyName', ...)
```

Retrieve one or more property values using the object method `get`.

```
PropertyValue(s) = ObjectName.get('PropertyName', ...)
```

List or retrieve all property values using one of the following commands:

```
get(ObjectName)
AllPropertyValues = get(ObjectName)
```

```
ObjectName.get
```

Help for Objects, Methods, and Properties

Display information for SimBiology object methods and properties in the
MATLAB Command Window.

`help sbio`	Display a list of functions and methods.
`help FunctionName`	Display function information.
`sbiohelp('MethodName')`	Display method information.
`sbiohelp('PropertyName')`	Display property information.

Properties — Alphabetical List

AbsoluteTolerance

Purpose　　　Specify largest allowable absolute error

Description　　The `AbsoluteTolerance` property specifies the largest allowable absolute error at any step in simulation. It is a property of `SolverOptions` object. `SolverOptions` is a property of the `configset` object. `AbsoluteTolerance` is available for the ode solvers (`'ode45'`, `'ode23'`, `'ode113'`, `'ode15s'`, `'ode23s'`, `'ode23t'`, and `'ode23tb'`).

At each simulation step, the solver estimates the local error e_i in the ith state vector y. Simulation converges at that time step if e_i satisfies the following equation:

`|e`$_i$`|≤max(RelativeTolerance*|y`$_i$`|,AbsoluteTolerance)`

Thus at higher state values, convergence is determined by `RelativeTolerance`. As the state values approach zero, convergence is controlled by `AbsoluteTolerance`. The choice of values for `RelativeTolerance` and `AbsoluteTolerance` will vary depending on the problem. The default values should work for first trials of the simulation; however if you want to optimize the solution, consider that there is a trade-off between speed and accuracy. If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy. If the results appear to be inaccurate, you can decrease the tolerance values but this will slow down the solver. If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

This may be important for reactions where species values tend to zero. Even if you are not interested in the value of a state `y(i)` when it is small, you may have to specify `AbsoluteTolerance` small enough to get some correct digits in `y(i)` so that you can accurately compute more interesting state values.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`

Data values	>0, <1. Default is 1e-6. For generated pharmacokinetic models default is 1e-15.
Access	Read/write

Examples

This example shows how to change AbsoluteTolerance.

1 Retrieve the configset object from the modelObj.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
```

2 Change the AbsoluteTolerance to 1e-8.

```
set(configsetObj.SolverOptions, 'AbsoluteTolerance', 1.0e-8);
get(configsetObj.SolverOptions, 'AbsoluteTolerance')

ans =

  1.0000e-008
```

See Also

RelativeTolerance

Active

Purpose	Indicate object in use during simulation

Description The Active property indicates whether a simulation is using a SimBiology object. A SimBiology model is organized into a hierarchical group of objects. Use the Active property to include or exclude objects during a simulation.

- **Configuration set** — For the configset object, use the method setactiveconfigset to set the object Active property to true.

- **Event, Reaction, or Rule** — When an event, a reaction, or rule object Active property is set to false, the simulation does not include the event, reaction, or rule. This is a convenient way to test a model with and without a reaction or rule.

- **Variant** — Set the Active property to true if you always want the variant to be applied before simulating the model. You can also pass the variant object as an argument to sbiosimulate; this applies the variant only for the current simulation. For more information on using the Active property for variants, see Variant object.

Characteristics

Applies to	Objects: configset, event, reaction, rule, or variant
Data type	boolean
Data values	true or false. The default value for events, reactions, and rules is true. For the configset object, default is true. For added configset object, the default is false. For variants, the default is false.
Access	Read/write

Examples **1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a reaction object and verify that the Active property setting is 'true' or 1.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
get (reactionObj, 'Active')
```

MATLAB returns:

```
ans =

1
```

3 Set the Active property to 'false' and verify.

```
set (reactionObj, 'Active', false);
get (reactionObj, 'Active')
```

MATLAB returns:

```
ans =

0
```

See Also   addconfigset, addreaction, addrule, Event object, Reaction object, Rule object, setactiveconfigset, Variant object

Annotation

Purpose   Store link to URL or file

Description   The `Annotation` property stores the URL or file name linking to information about a model.

Characteristics

Applies to	SimBiology objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, rule, species, or unit
Data type	`char string`, `URL`
Data values	Character string with a directory path and filename or a URL
Access	Read/write

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Set the annotation for a model object.

```
set (modelObj, 'annotation', 'www.reactome.org')
```

3 Verify the assignment.

```
get (modelObj, 'annotation')
```

MATLAB returns:

```
ans =

www.reactome.org
```

See Also   addkineticlaw, addparameter, addreaction, addrule, addspecies, sbiomodel, sbioroot, sbiounit, sbiounitprefix

Purpose   Indicate species boundary condition

Description   The BoundaryCondition property indicates whether a species object
has a boundary condition. If BoundaryCondition is true, the species
quantity is determined by InitialAmount and/or a rule object, and not
by the reaction rate equation. All SimBiology species are state variables
regardless of the BoundaryCondition or ConstantAmount property.

By default, BoundaryCondition is false and the reaction rate
equations determine the rate of change of a species quantity in the
model. Boundary condition is used when a species is modeled as a
participant of reactions but the species quantity is not determined by
a reaction rate equation.

More Information

Consider the following two use cases of boundary conditions:

- Modeling receptor-ligand interactions that affect the rate of change
  of the receptor but not the ligand. For example, in response to
  hormone, steroid receptors such as the glucocorticoid receptor (GR)
  translocate from the cytoplasm (cyt) to the nucleus (nuc). The hsp90/
  hsp70 chaperone complex directs this nuclear translocation [Pratt
  2004]. The natural ligand for GR is cortisol; the synthetic hormone
  dexamethasone (dex) is used in place of cortisol in experimental
  systems. In this system dexamethasone participates in the reaction
  but the quantity of dexamethasone in the cell is regulated using a
  rule. To simply model translocation of GR you could use the following
  reactions:

  Formation of the chaperone-receptor complex,

  ```
  Hsp90_complex + GR_cyt -> Hsp90_complex:GR_cyt
  ```

  In response to the synthetic hormone dexamethasone (dex), GR
  moves from the cytoplasm to the nucleus.

  ```
  Hsp90_complex:GR_cyt + dex -> Hsp90_complex + GR_nuc + dex
  ```

BoundaryCondition

For `dex`,

```
BoundaryCondition = true; ConstantAmount = false
```

In this example `dex` is modeled as a boundary condition with a rule to regulate the rate of change of `dex` in the system. Here, the quantity of `dex` is not determined by the rate of the second reaction but by a rate rule such as

```
ddex/dt = 0.001
```

which is specified in the SimBiology software as

```
dex = 0.001
```

- Modeling the role of nucleotides (for example, GTP, ATP, cAMP) and cofactors (for example, Ca^{++}, NAD^+, coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.

```
Ras-GDP + GTP  -> Ras-GTP + GDP

For GTP, BoundaryCondition = true; ConstantAmount = true
```

Model GTP and GDP with boundary conditions, thus making them *boundary species*. In addition, you can set the `ConstantAmount` property of these species to `true` to indicate that their quantity does not vary during a simulation.

Characteristics

Applies to	Object: species
Data type	`boolean`
Data values	`true` or `false`. The default value is `false`.
Access	Read/write

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a species object and verify that the boundary condition property setting is `'false'` or `0`.

```
speciesObj = addspecies(modelObj, 'glucose');
get(speciesObj, 'BoundaryCondition')
```

MATLAB returns:

```
ans =

     0
```

3 Set the boundary condition to `'true'` and verify.

```
set(speciesObj, 'BoundaryCondition', true);
get(speciesObj, 'BoundaryCondition')
```

MATLAB returns:

```
ans =

     1
```

References   Pratt, W.B., Galigniana, M.D., Morishima, Y., Murphy, P.J. (2004), Role of molecular chaperones in steroid receptor action, *Essays Biochem*, 40:41-58.

See Also   addrule, addspecies, ConstantAmount, InitialAmount

BuiltInKineticLaws

Purpose   Contain built-in kinetic laws

Note BuiltInKineticLaws has been removed and produces an error.
Use BuiltInLibrary instead.

Description   BuiltInKineticLaws is a SimBiology root object property showing all
abstract kinetic laws that are shipped with the SimBiology software.
Use the command sbiowhos -builtin -kineticlaw to see the list of
built-in kinetic laws. You can use built-in kinetic laws when you use the
command addkineticlaw to create a kinetic law object for a reaction
object. Refer to the kinetic law by name when you create the kinetic law
object, for example:

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

You cannot add, modify, or delete BuiltInKineticLaws.

See "Abstract Kinetic Law" on page 6-56 for a definition and more
information.

Characteristics

Applies to	Object: root
Data type	char string of valid abstract kinetic law name
Data values	Valid kinetic laws
Access	Read-only

See Also   BuiltInLibrary

Purpose   Library of built-in components

Description   BuiltInLibrary is a SimBiology root object property containing all built-in components namely, unit, unit-prefixes, and abstract kinetic laws that are shipped with the SimBiology product. You cannot add, modify, or delete components in the built-in library. The BuiltInLibrary property is an object that contains the following properties:

- Units — contains all units that are shipped with the SimBiology product. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in units either by using the command sbiowhos -builtin -unit, or by accessing the root object.

- UnitPrefixes — contains all unit-prefixes that are shipped with the SimBiology product. You can specify unit—prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in unit-prefixes either by using the command sbiowhos -builtin -unitprefix, or by accessing the root object.

- KineticLaws — contains all abstract kinetic laws that are shipped with the SimBiology product. Use the command sbiowhos -builtin -kineticlaw to see the list of built-in kinetic laws. You can use built-in kinetic laws when you use the command addkineticlaw to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');

See "Abstract Kinetic Law" on page 6-56 for a definition and more information.

Characteristics   BuiltInLibrary

BuiltInLibrary

Applies to	Object: root
Data type	object
Data values	Unit, unit-prefix, and abstract kinetic law objects
Access	Read-only

Characteristics for `BuiltInLibrary` properties:

- `Units`

Applies to	`BuiltInLibrary` property
Data type	unit objects
Data values	units
Access	Read-only

- `UnitPrefixes`

Applies to	`BuiltInLibrary` property
Data type	unit prefix objects
Data values	unit prefixes
Access	Read-only

- `KineticLaws`

Applies to	`BuiltInLibrary` property
Data type	Abstract kinetic law object
Data values	kinetic laws
Access	Read-only

Examples   **Example 1**

This example uses the command sbiowhos to show the current list of built-in components.

```
sbiowhos -builtin -kineticlaw
sbiowhos -builtin -unit
sbiowhos -builtin -unitprefix
```

Example 2

This example shows the current list of built-in components by accessing the root object.

```
rootObj = sbioroot;
get(rootObj.BuiltinLibrary, 'KineticLaws')
get(rootObj.BuiltinLibrary, 'Units')
get(rootObj.BuiltinLibrary, 'UnitPrefixes')
```

See Also   Functions — sbioaddtolibrary, sbioremovefromlibrary sbioroot, sbiounit, sbiounitprefix

Properties — UserDefinedLibrary

BuiltInUnitPrefixes

Purpose   Contain built-in unit prefixes

> **Note** BuiltInUnitPrefixes has been removed and produces an error.
> Use BuiltInLibrary instead.

Description BuiltInUnitPrefixes is a SimBiology root object property showing
all unit prefixes that are shipped with SimBiology. You can specify
units with prefixes for species amounts and parameter values to do
dimensional analysis and unit conversion during simulation. The
valid units and unit prefixes are either built-in or user-defined. You
can display the built-in unit prefixes either by using the command
sbiowhos, or by accessing the root object.

You cannot add, modify, or delete BuiltInUnitsPrefixes.

Characteristics

Applies to	Object: root
Data type	char string
Data values	Valid units
Access	Read-only

See Also   BuiltInLibrary

Purpose   Contain built-in units

Note `BuiltInUnits` has been removed and produces an error. Use `BuiltInLibrary` instead.

Description   `BuiltInUnits` is a SimBiology root object property showing all units that are shipped with SimBiology. You can specify units for species amounts and parameter values to do dimensional analysis and unit conversion during simulation. The valid units are either built-in or user-defined. You can display the built-in units either by using the command `sbiowhos`, or by accessing the root object.

You cannot add, modify, or delete `BuiltInUnits`.

Characteristics

Applies to	Object: root
Data type	char string
Data values	Valid units
Access	Read-only

See Also   `BuiltInLibrary`

Capacity

Purpose   Compartment capacity

Description   The Capacity property indicates the size of the SimBiology compartment object. If the size of the compartment does not vary during simulation, set the property ConstantCapacity to true.

You can vary compartment capacity using rules or events.

Note Remember to set the ConstantCapacity property to false for varying capacity.

Events cannot result in the capacity having a negative value. Rules could result in the capacity having a negative value.

Characteristics

Applies to	Object: compartment
Data type	double
Data values	Positive real number. The default value is 1.
Access	Read/write

Examples   Add a compartment to a model and set the compartment capacity.

1 Create a model object named my_model.

```
modelObj = sbiomodel ('comp_model');
```

2 Add the compartment object named nucleus with a capacity of 0.5.

```
compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);
```

See Also   addcompartment, addspecies, CapacityUnits, ConstantCapacity

Purpose   Compartment capacity units

Description   The CapacityUnits property indicates the unit definition for the Capacity property of a compartment object. CapacityUnits can be any unit from the units library. To get a list of the defined units in the library, use the sbioshowunits function. If CapacityUnits changes from one unit definition to another, the Capacity does not automatically convert to the new units. The sbioconvertunits function does this conversion. To add a user-defined unit to the list, see sbioaddtolibrary.

Characteristics

Applies to	Object: compartment
Data type	char string
Data values	Units from library with dimensions of length, area, or volume. Default = '' (empty).
Access	Read/write

Examples   **1** Create a model object named my_model.

```
modelObj = sbiomodel ('my_model');
```

2 Add a compartment object named cytoplasm with a capacity of 0.5.

```
compObj = addcompartment (modelObj, 'cytoplasm', 0.5);
```

3 Set the CapacityUnits to femtoliter, and verify.

```
set (compObj,'CapacityUnits', 'femtoliter');
get (compObj,'CapacityUnits')
```

MATLAB returns:

```
ans =

femtoliter
```

CapacityUnits

See Also   InitialAmount, sbioaddtolibrary, sbioconvertunits, sbioshowunits

Purpose   Array of compartments in model or compartment

Description   Compartments shows you a read-only array of SimBiology compartment
objects in the model object and the compartment object. In the model
object, the Compartments property indicates all the compartments in a
Model object as a flat list. In the compartment object, the Compartments
property indicates other compartments that are referenced within the
compartment. The two instances of Compartments are illustrated in
"Examples" on page 6-19.

You can add a compartment object using the method addcompartment.

Characteristics

Applies to	Objects: compartment, model
Data type	Array of compartment objects
Data values	Compartment object. Default is [] (empty).
Access	Read-only

Examples   **1** Create a model object named modelObj.

    modelObj = sbiomodel('cell');

2 Add two compartments to the model object.

    compartmentObj1 = addcompartment(modelObj, 'nucleus');
    compartmentObj2 = addcompartment(modelObj, 'mitochondrion');

3 Add a compartment to one of the compartment objects.

    compartmentObj3 = addcompartment(compartmentObj2, 'matrix');

4 Display the Compartments property in the model object.

    get(modelObj, 'Compartments')

    SimBiology Compartment Array

```
Index:     Name:              Capacity:     CapacityUnits:
1          nucleus            1
2          mitochondrion      1
3          matrix             1
```

5 Display the Compartments property in the compartment object.

```
get(compartmentObj2, 'Compartments')

SimBiology Compartment - matrix

  Compartment Components:
    Capacity:          1
    CapacityUnits:
    Compartments:      0
    ConstantCapacity:  true
    Owner:             mitochondrion
    Species:           0
```

See Also   addcompartment, addreaction, addspecies, Compartment object

Purpose   Dimensional analysis and unit conversion options

Description   The SimBiology CompileOptions property is an object that defines
the compile options available for simulation; you can specify whether
dimensional analysis and unit conversion is necessary for simulation.
Compile options are checked during compile time. The compile options
object can be accessed through the CompileOptions property of the
configset object. Retrieve CompileOptions object properties with the
get function and configure the properties with the set function.

**Property
Summary**

DefaultSpeciesDimension	Species dimension
DimensionalAnalysis	Perform dimensional analysis on model
Type	Display top-level SimBiology object type
UnitConversion	Perform unit conversion

Characteristics

Applies to	Object: configset
Data type	Object
Data values	Compile-time options
Access	Read-only

Examples   **1** Retrieve the configset object of modelObj.

```
modelObj = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2 Retrieve the CompileOptions object (optionsObj) from the
configsetObj.

```
optionsObj = get(configsetObj, 'CompileOptions');
```

CompileOptions

```
Compile Settings:

    UnitConversion:      false
    DimensionalAnalysis: true
```

See Also   get, set

Purpose   Unit composition

Description   The Composition property holds the composition of a unit object. The Composition property shows the combination of base and derived units that defines the unit. For example, molarity is the name of the unit and the composition is mole/liter. Base units are the set of units used to define all unit quantity equations. Derived units are defined using base units or mixtures of base and derived units.

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

Characteristics

Applies to	Object: Unit
Data type	char string
Data values	Valid combination of units and prefixes from the library. Default is '' (empty).
Access	Read/write

Examples   This example shows you how to create a user-defined unit, add it to the user-defined library, and query the Composition property.

1 Create a unit for the rate constants of a second-order reaction.

```
unitObj = sbiounit('secondconstant', '1/molarity*second', 1);
```

2 Query the Composition property.

```
get(unitObj, 'Composition')

ans =

1/molarity*second
```

3 Change the Composition property.

Composition

```
set(unitObj, 'Composition', 'liter/mole*second'))

ans =

liter/mole*second
```

4 Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

See Also get, Multiplier, Offset, sbiounit, set

Purpose   Specify variable or constant species amount

Description   The ConstantAmount property indicates whether the quantity of the species object can vary during the simulation. ConstantAmount can be either true or false. If ConstantAmount is true, the quantity of the species cannot vary during the simulation. By default, ConstantAmount is false and the quantity of the species can vary during the simulation. If ConstantAmount is false, the quantity of the species can be determined by reactions and rules.

The property ConstantAmount is for species objects; the property ConstantValue is for parameter objects.

More Information

The following is an example of modeling species as constant amounts:

Modeling the role of nucleotides (GTP, ATP, cAMP) and cofactors (Ca^{++}, NAD^+, coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.

```
Ras-GDP + GTP  -> Ras-GTP + GDP
```

Model GTP and GDP with constant amount set to true. In addition, you can set the BoundaryCondition of these species to true, thus making them *boundary species*.

Characteristics

Applies to	Object: species
Data type	boolean
Data values	true or false. The default value is false.
Access	Read/write

Examples   **1** Create a model object named my_model.

```
modelObj = sbiomodel ('my_model');
```

ConstantAmount

2 Add a species object and verify that the ConstantAmount property setting is `'false'` or O.

```
speciesObj = addspecies (modelObj, 'glucose');
get (speciesObj, 'ConstantAmount')
```

MATLAB returns:

```
            ans =

        0
```

3 Set the constant amount to `'true'` and verify.

```
set (speciesObj, 'ConstantAmount', true);
get (speciesObj, 'ConstantAmount')
```

MATLAB returns:

```
ans =

        1
```

See Also   addspecies, BoundaryCondition

Purpose   Specify variable or constant compartment capacity

Description   The ConstantCapacity property indicates whether the capacity of the compartment object can vary during the simulation. ConstantCapacity can be either true (1) or false (0). If ConstantCapacity is true, the quantity of the compartment cannot vary during the simulation. By default, ConstantCapacity is true and the quantity of the compartment cannot vary during the simulation. If ConstantCapacity is false, the quantity of the compartment can be determined by rules and events.

Characteristics

Applies to	Object: compartment
Data type	boolean
Data values	true or false. The default value is true.
Access	Read/write

Examples   Add a compartment to a model and check the ConstantCapacity property of the compartment.

1 Create a model object named comp_model.

    modelObj = sbiomodel ('comp_model');

2 Add the compartment object named nucleus with a capacity of 0.5.

    compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);

3 Display the ConstantCapacity property.

    get(compartmentObj, 'ConstantCapacity')

    ans =

        1

See Also   addcompartment, ConstantAmount, ConstantValue

ConstantValue

Purpose   Specify variable or constant parameter value

Description   The `ConstantValue` property indicates whether the value of a parameter can change during a simulation. Enter either `true` (value is constant) or `false` (value can change).

You can allow the value of the parameter to change during a simulation by specifying a rule that changes the `Value` property of the parameter object.

The property `ConstantValue` is for parameter objects; the property `ConstantAmount` is for species objects.

More Information

As an example, consider feedback inhibition of an enzyme such as aspartate kinase by threonine. Aspartate kinase has three isozymes that are independently inhibited by the products of downstream reactions (threonine, homoserine, and lysine). Although threonine is made through a series of reactions in the synthesis pathway, for illustration, the reactions are simplified as follows:

$$Aspartic \text{ acid} \xrightarrow{\text{aspartate kinase}} \beta - Aspartylphosphate$$

$$\beta - Aspartylphosphate \longrightarrow Threonine$$
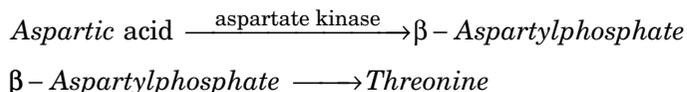
To model inhibition of aspartate kinase by threonine, you could use a rule like the `algebraic` rule below to vary the rate of the above reaction and simulate inhibition. In the rule, the rate constant for the above reaction is denoted by `k_aspartate_kinase` and the quantity of threonine is `threonine`.

```
k_aspartate_kinase -(1/threonine)
```

Characteristics

Applies to	Object: parameter
Data type	`boolean`

Data values	true or false. The default value is 'true'.
Access	Read/write

Examples   **1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

3 Change the ConstantValue property of the parameter object from default (true) to false and verify.

MATLAB returns 1 for true and 0 for false.

```
set (parameterObj, 'ConstantValue', false);
get(parameterObj, 'ConstantValue')
```

MATLAB returns:

```
ans =

     0
```

See Also   addparameter

Content

Purpose   Contents of variant object

Description   The `Content` property contains the data for the variant object. `Content` is a cell array with the structure `{'Type', 'Name', 'PropertyName', 'PropertyValue'}`. You can store values for species `InitialAmount`, parameter `Value`, and compartment `Capacity`, in a variant object.

For more information about variants, see `Variant object`.

Characteristics

Applies to	Object: variant
Data type	`cell array`
Data values	Default value is `[]` (empty).
Access	Read/write

Examples   **1** Create a model containing three species in one compartment.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
A = addspecies(compObj, 'A');
B = addspecies(compObj, 'B');
C = addspecies(compObj, 'C');
```

2 Add a variant object that varies the species' `InitialAmount` property.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {{'species','A', 'InitialAmount', 5}, ...
{'species', 'B', 'InitialAmount', 10}});
% Display the variant
variantObj

SimBiology Variant - v1 (inactive)

   ContentIndex:    Type:       Name:          Property:         Value:
   1                species     A              InitialAmount     5
   2                species     B              InitialAmount     10
```

3 Append data to the Content property.

```
addcontent(variantObj, {'species', 'C', 'InitialAmount', 15});

SimBiology Variant - v1 (inactive)

   ContentIndex:     Type:      Name:        Property:        Value:
   1                 species    A            InitialAmount      5
   2                 species    B            InitialAmount      10
   3                 species    C            InitialAmount      15
```

4 Remove a species from the Content property.

```
rmcontent(variantObj, 3);
```

5 Replace the data in the Content property.

```
set(variantObj, 'Content', {'species', 'C', 'InitialAmount', 15});
```

See Also   addcontent, rmcontent, sbiovariant

CovariateLabels

Purpose	Identify covariate columns in data set
Description	CovariateLabels is a property of the PKData object. It specifies the column in DataSet that contains that contain the covariate data.

Characteristics

Applies to	Object: PKData
Data type	char string or cell array of strings
Data values	Column headers from imported data set
Access	Read/write

See Also   "Specifying and Classifying the Data to Fit" and "Specifying the Covariate Model" in the SimBiology User's Guide, PKData object

Purpose   Store simulation data

Description   The `Data` property contains the simulation data stored in the SimData object.

This property contains all data logged during a simulation, including species amounts, parameter values, and sensitivities. The property is an `m x n` array, where `m` is the number of time steps in the simulation and `n` is the number of quantities logged. The rows of the array are labeled by the time points in the `Time` property, and the columns are labeled by the metadata in the `DataInfo` property.

Characteristics

Applies to	Object: SimData
Data type	`double`
Data values	Default value is `[]` (empty).
Access	Read-only

See Also   `DataInfo`, `ModelName`

DataCount

Purpose Numbers of species, parameters, sensitivities

Description The DataCount property shows how many species, parameters, and sensitivities are logged in a SimData object. It is a MATLAB structure with the fields Species, Parameter, and Sensitivity. The information in this property is redundant with the DataInfo property and is there to give you a convenient means to access the information.

Characteristics

Applies to	Object: SimData
Data type	struct
Data values	Default value for each field is 0.
Access	Read-only

See Also StopTime, StopTimeType

Purpose   Metadata labels for simulation data

Description   The `DataInfo` property contains the metadata that label the columns of the SimData object array. It is an `n x 1` cell array of structures. The `ith` cell contains metadata labeling the `ith` column of the SimData object array.

The possible types of structures are as follows.

Type	Fields

Species	<pre>Type: species
	Name:
	Compartment:
	Units:</pre>

Parameter	<pre>Type: parameter
	Name:
	Reaction: <name of reaction that a parameter is scoped to,
	          or '' if parameter is scoped to model>
	Units:</pre>

Sensitivity	<pre>Type: sensitivity
	Name: <for example: d[x]/d[y]_O>
	OutputType: <The type of the sensitivity output,
	            either 'species' or 'parameter'>
	OutputName: <The name of the sensitivity output>
	OutputQualifier: <The compartment or reaction for
	                 the sensitivity output, for
	                 species or parameters, respectively>
	InputType: <The type of the sensitivity input,
	           either 'species' or 'parameter'>
	InputName: <The name of the sensitivity input>
	InputQualifier: <The compartment or reaction for</pre>

DataInfo

Type	Fields
	the sensitivity input, for species or parameters, respectively> Units:

Characteristics

Applies to	Object: SimData
Data type	n x 1 cell array of structs
Data values	Default value is 0x1 cell array.
Access	Read-only

See Also StopTime, StopTimeType

DataNames

Purpose   Show names in SimData object

Description   The DataNames property holds the names that label the columns of the data matrix in the Data property. The property contains an nx1 array of strings. The software provides this information for your convenience.

Characteristics

Applies to	Object: SimData
Data type	string array
Data values	Default value is 0x1 cell array.
Access	Read-only

See Also   StopTime, StopTimeType

DataSet

Purpose   Dataset object containing imported data

Description   DataSet is a property of the PKData object. It contains the imported
                    data set. The PKData object constructor (PKData) assigns the specified
                    data set to its DataSet property during construction.

Characteristics

Applies to	Object: PKData
Data type	dataset object
Data values	Variable containing dataset object
Access	Read-only

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's
                    Guide, PKData object

DefaultSpeciesDimension

Purpose   Species dimension

Description   The `DefaultSpeciesDimension` property specifies whether the species dimensions are `substance` or `concentration`. If however, you specify the species units in the `InitialAmountUnits` property, these units define the species dimension regardless of the value in `DefaultSpeciesDimension`. Thus, if `DefaultSpeciesDimension` is `concentration` and you specify species units as `molecule`, the species dimensions are evaluated as `substance`.

You can find `DefaultSpeciesDimension` in the `CompileOptions` property.

When `DefaultSpeciesDimension` is set to `substance`, species quantities ignore compartment capacity, unless capacity is explicitly defined in an expression (reaction rate, rule, or event expression).

When `DefaultSpeciesDimension` is set to `concentration`, species quantities are scaled for compartment capacity in reaction rate, rule, or event expressions. `CompartmentCapacity` has a default value of `1`, thus when capacity and capacity unit are not defined, species amount is equivalent to concentration.

For example, consider a reaction `a + b  > c`. Using mass action kinetics, the reaction rate is defined as `a*b*k`, where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are `0.01M` and `0.005M` respectively, then units of `k` are `1/(M*second)`. If you specify `k` with another equivalent unit definition, for example, `1/[(moles/liter)*second]`, `DimensionalAnalysis` checks whether the physical quantities match. If the physical quantities do not match, you see an error and the model is not simulated.

If, in the above example, you specify that initial amounts of `a` and `b` are `0.01` and `0.005` respectively, without specifying units, the compile options check whether `DefaultSpeciesDimension` is `substance` or `concentration`. If the `DefaultSpeciesDimension` is `concentration`, and the reaction rate dimensions resolve to concentration/time, the model is simulated with species amounts scaled for compartment capacity, and the solver returns the species values in concentration.

DefaultSpeciesDimension

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

Characteristics

Applies to	Object: `CompileOptions` (in `configset` object)
Data type	`char string`
Data values	`concentration` or `substance`. Default value is `concentration`.
Access	Read/write

See Also   `CompileOptions`, `DimensionalAnalysis`, `get`, `getconfigset`, `sbiosimulate`, `set`

Purpose   Identify dependent variable column in data set

Description   DependentVarLabel is a property of the PKData object. It specifies the column in DataSet that contains the dependent variable (for example, measured response).

Characteristics

Applies to	Objects: PKData
Data type	char string
Data values	Column header from imported data set
Access	Read/write

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, PKData object

DimensionalAnalysis

Purpose   Perform dimensional analysis on model

Description   The DimensionalAnalysis property specifies whether to perform
                     dimensional analysis on the model before simulation. It is a property of
                     the CompileOptions object. CompileOptions holds the model's compile
                     time options and is the object property of the configset object. When
                     DimensionalAnalysis is set to true, the SimBiology software checks
                     whether the physical quantities of the units involved in reactions and
                     rules, match and are applicable.

                     For example, consider a reaction a + b > c. Using mass action
                     kinetics, the reaction rate is defined as a*b*k, where k is the rate
                     constant of the reaction. If you specify that initial amounts of a and b
                     are 0.01M and 0.005M respectively, then units of k are 1/(M*second).
                     If you specify k with another equivalent unit definition, for example,
                     1/[(moles/liter)*second], DimensionalAnalysis checks whether
                     the physical quantities match. If the physical quantities do not match,
                     you see an error and the model is not simulated.

                     Unit conversion requires dimensional analysis. If DimensionalAnalysis
                     is off, and you turn UnitConversion on, then DimensionalAnalysis
                     is turned on automatically. If UnitConversion is on and you turn
                     off DimensionalAnalysis, then UnitConversion is turned off
                     automatically.

                     If you have MATLAB function calls in your model, dimensional analysis
                     ignores any expressions containing function calls and generates a
                     warning.

                     Valid physical quantities for reaction rates are amount/time, mass/time,
                     or concentration/time.

Characteristics

Applies to	Object: CompileOptions (in configset object)
Data type	boolean

Data values   `true` or `false`. Default value is `true`.

Access   Read/write

Examples   This example shows how to retrieve and set `DimensionalAnalysis` from the default `true` to `false` in the default configuration set in a model object.

1 Import a model.

```
modelObj = sbmlimport('oscillator')


SimBiology Model - Oscillator

   Model Components:
     Models:          0
     Parameters:      0
     Reactions:       42
     Rules:           0
     Species:         23
```

2 Retrieve the `configset` object of the model object.

```
configsetObj = getconfigset(modelObj)

 Configuration Settings - default (active)
    SolverType:          ode15s
    StopTime:            10.000000

    SolverOptions:
      AbsoluteTolerance:  1.000000e-006
      RelativeTolerance:  1.000000e-003

    RuntimeOptions:
      StatesToLog:        all
```

```
        CompileOptions:
          UnitConversion:        true
          DimensionalAnalysis:   true
```

3 Retrieve the CompileOptions object.

```
        optionsObj = get(configsetObj,'CompileOptions')
```

```
        Compile Settings:
```

```
          UnitConversion:        true
          DimensionalAnalysis:   true
```

4 Assign a value of false to DimensionalAnalysis.

```
         set(optionsObj,'DimensionalAnalysis', false)
```

See Also   get, getconfigset, sbiosimulate, set

Purpose   Dosed object name

Description   Dosed is a property of the PKModelMap object. It specifies the name of an object that is receiving an input, such as a drug in a compartment or a ligand. Specify the name of a species, a compartment, or a parameter (scoped to a model).

Characteristics

Applies to	Objects: PKModelMap
Data type	char string or cell array of strings
Data values	Name of an object, or empty
Access	Read/write

See Also   "Defining Model Components for Observed Response, Dose, Dosing Type, and Parameters to be Estimated" in the SimBiology User's Guide, Estimated, Observed, PKModelMap object

DoseLabel

Purpose	Identify dose column in data set
Description	DoseLabel is a property of the PKData object. It specifies the column in DataSet that contains that contains the dosing information.

Characteristics

Applies to	Object: PKData
Data type	char string
Data values	Column header from imported data set
Access	Read/write

See Also "Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, PKData object

Purpose   Drug dosing type in compartment

Description   DosingType is a property of the PKCompartment and PKModelMap objects. It specifies the type of dosing of a drug in a compartment. You can only dose one compartment in the model at any given time. For a description of the types of dosing supported, the model components created for each type of dosing, and the parameters to estimate, see "About Dosing Types" in the SimBiology User's Guide.

Characteristics

Applies to	Objects: PKCompartment, PKModelMap
Data type	char string
Data values	'', 'Bolus', 'Infusion', 'ZeroOrder', 'FirstOrder'
Access	Read/write

See Also   EliminationType, PKCompartment object, PKModelMap object

EliminationType

Purpose Drug elimination type from compartment

Description `EliminationType` is a property of the `PKCompartment` object. It specifies the type of elimination of adrug from a compartment. For a description of the types of elimination supported, the model components created for each type of elimination, and the parameters to estimate, see "About Elimination Types" in the SimBiology User's Guide.

Characteristics

Applies to	Object: PKCompartment
Data type	char string
Data values	'Linear', 'Linear-Clearance', 'Enzymatic', and ''
Access	Read/write

See Also addCompartment, DosingType, PKCompartment object

Purpose   Specify explicit or implicit tau error tolerance

Description   The ErrorTolerance property specifies the error tolerance for the explicit tau and implicit tau stochastic solvers. It is a property of the SolverOptions object. SolverOptions is a property of the configset object. The explicit and implicit tau solvers automatically chooses a time interval (tau) such that the relative change in the propensity function for each reaction is less than the user-specified error tolerance.

A propensity function describes the probability that the reaction will occur in the next smallest time interval, given the conditions and constraints.

If the error tolerance is too large, there may not be a solution to the problem and that could lead to an error. If the error tolerance is small, the solver will take more steps than when the error tolerance is large leading to longer simulation times. The error tolerance should be adjusted depending upon the problem, but a good value for the error tolerance is between 1 % to 5 %.

Characteristics

Applies to	Object: SolverOptions
Data type	double
Data values	>0, <1. The default is 3e-2.
Access	Read/write

Examples   This example shows how to change ErrorTolerance settings.

1 Retrieve the configset object from the modelObj and change the SolverType to expltau.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'expltau')
```

2 Change the ErrorTolerance to 1e-8.

ErrorTolerance

```
set(configsetObj.SolverOptions, 'ErrorTolerance', 5.0e-2);
get(configsetObj.SolverOptions, 'ErrorTolerance')


ans =

  5.000000e-002
```

See Also LogDecimation, RandomState

Purpose   Names of parameters to estimate

Description   Estimated is a property of the `PKModelMap` object. It specifies the name
                   of the objects to estimate. Specify the name of a species, compartment,
                   or parameter.

Characteristics

Applies to	Objects: `PKModelMap`
Data type	`char string` or cell array of strings
Data values	Name of objects
Access	Read/write

See Also   "Defining Model Components for Observed Response, Dose, Dosing
                   Type, and Parameters to be Estimated" in the SimBiology User's Guide,
                   `Dosed`, `Observed`, `PKModelMap object`

EventFcns

Purpose   Event expression

Description   Property of the event object that defines what occurs when the event is triggered. Specify a cell array of strings.

EventFcns can be any MATLAB assignment or expression that defines what is executed when the event is triggered. All EventFcn expressions are assignments of the form '*objectname* = *expression*', where *objectname* is the name of a valid SimBiology object.

For more information about how SimBiology handles events, see "How Events Are Evaluated" in the SimBiology User's Guide documentation. For examples of event functions, see "Specifying Event Functions" in the SimBiology User's Guide documentation.

Characteristics

Applies to	Object: event
Data type	Cell array of strings
Data values	EventFcn strings '' (empty)
Access	Read/write

Examples   **1** Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator');
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

2 Set the EventFcns property of the event object.

```
set(eventObj, 'EventFcns', {'pA = OpA','mA = pol'});
```

3 Get the EventFcns property.

```
get(eventObj, 'EventFcns')
```

See Also   Event object, Trigger

Purpose   Contain all event objects

Description Property to indicate events in a model object. Read-only array of Event objects.

An event defines an action when a defined condition is met. For example, the quantity of a species may double when the quantity of species B is 100. An event is triggered when the conditions specified in the event are met by the model. See "Changing Model Component Values Using Events" in the SimBiology User's Guide documentation for more information.

Add an event to a Model object with the method addevent method and remove an event with the delete method. See Event object for more information.

You can view event object properties with the get command and modify the properties with the set command.

Characteristics

Applies to	Object: model
Data type	Array of event objects
Data values	Event object. The default is [] (empty).
Access	Read-only

Examples **1** Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator')
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

2 Get a list of properties for an event object.

```
get(modelObj.Events(1));
```

Or

```
get(eventObj)
```

MATLAB displays a list of event properties.

```
                 Active: 1
             Annotation: ''
              EventFcns: {'OpC = 200'}
                   Name: ''
                  Notes: ''
                 Parent: [1x1 SimBiology.Model]
                    Tag: ''
                Trigger: 'time >= 5'
            TriggerDelay: 0
        TriggerDelayUnits: 'second'
                   Type: 'event'
               UserData: []
```

See Also   EventFcns, Event object, Model object, Trigger

Purpose   Exponent value of unit prefix

Description   *Exponent* shows the value of 10^Exponent that defines the numerical value of the unit prefix *Name*. You can use the unit prefix in conjunction with any built-in or user-defined units. For example, for the unit mole, specify as picomole to use the Exponent, -12.

Characteristics

Applies to	Object: Unit prefix
Data type	double
Data values	Real number. Default is 0.
Access	Read/write

Examples   This example shows you how to create a user-defined unit prefix, add it to the user-defined library, and query the Exponent property.

1 Create a unit prefix.

```
unitprefixObj1 = sbiounitprefix('peta', 15);
```

2 Add the unit prefix to the user-defined library.

```
sbioaddtolibrary(unitprefixObj1);
```

3 Query the Exponent property.

```
get(unitprefixObj1, 'Exponent')

ans =

    15
```

See Also   get, sbioaddtolibrary, sbiounitprefix, set, UnitPrefix object

Expression

Purpose
Expression to determine reaction rate equation

Description
The `Expression` property indicates the mathematical expression that is used to determine the `ReactionRate` property of the reaction object. `Expression` is a reaction rate expression assigned by the abstract kinetic law used by the kinetic law object. The abstract kinetic law being used is indicated by the property `KineticLawName`. You can configure `Expression` for user-defined abstract kinetic laws, but not for built-in abstract kinetic laws. `Expression` is read only for kinetic law objects.

Abstract Kinetic Law

The abstract kinetic law provides a mechanism for applying a specific rate law to multiple reactions. It acts as a mapping template for the reaction rate. The abstract kinetic law is defined by a reaction rate expression, which is defined in the property `Expression`, and the species and parameter variables used in the expression. The species variables are defined in the `SpeciesVariables` property, and the parameter variables are defined in the `ParameterVariables`property of the kinetic law object.

If a reaction is using an abstract kinetic law, the `ReactionRate` property of the reaction object shows the result of a mapping from an abstract kinetic law. To determine `ReactionRate`, the species variables and parameter variables that participate in the reaction rate should be clearly mapped in the kinetic law for the reaction. In this case, SimBiology software determines the `ReactionRate` by using the `Expression` property of the abstract kinetic law object, and by mapping `SpeciesVariableNames` to `SpeciesVariables` and `ParameterVariableNames` to `ParameterVariables`.

For example, the abstract kinetic law `Henri-Michaelis-Menten` has the `Expression` Vm*S/(Km+S), where `Vm` and `Km` are defined as parameters in the `ParameterVariables` property of the abstract kinetic law object, and `S` is defined as a species in the `SpeciesVariable` property of the abstract kinetic law object.

By applying the abstract kinetic law `Henri-Michaelis-Menten` to a reaction `A -> B` with `Va` mapping to `Vm`, `A` mapping to `S`, and `Ka` mapping to `Km`, the rate equation for the reaction becomes Va*A/(Ka+A).

The exact expression of a reaction using `MassAction` kinetic law varies depending upon the number of reactants. Thus, for mass action kinetics the `Expression` property is set to `MassAction` because in general for mass action kinetics the reaction rate is defined as

$$r = k \prod_{i=1}^{n_r} [Si]^{m_i}$$

where [`Si`] is the concentration of the `ith` reactant, m_i is the stoichiometric coefficient of [`Si`], n_r is the number of reactants, and `k` is the mass action reaction rate constant.

SimBiology software comes with some built-in kinetic laws. Users can also define their own abstract kinetic laws. To find the list of available kinetic laws, use the `sbiowhos -kineticlaw` command (`sbiowhos`). You can create an abstract kinetic law with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

Characteristics

Applies to	Objects: abstract kineticlaw, kineticlaw
Data type	`char string`
Data values	Defined by abstract kinetic law
Access	Read-only in kinetic law object. Read/write in user-defined abstract kinetic law.

Examples

Example 1

Example with `Henri-Michaelis-Menten` kinetics

Expression

1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Verify that the Expression property for the kinetic law object is Henri-Michaelis-Menten.

```
get (kineticlawObj, 'Expression')
```

MATLAB returns:

```
ans =

Vm*S/(Km + S)
```

4 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that you should set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm_d, Km_d, and assign the objects' Parent property value to the kineticlawObj. The species object with Name a is created when reactionObjis created and need not be redefined.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

5 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Vm_d' 'Km_d'});
set(kineticlawObj,'SpeciesVariableNames', {'a'});
```

6 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Vm_d*a/(Km_d+a)
```

Example 2

Example with Mass Action kinetics.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
get(kineticlawObj, 'Expression')
```

MATLAB returns:

```
ans =

MassAction
```

3 Assign the rate constant for the reaction.

```
set (kineticlawObj, 'ParameterVariablenames', 'k');

get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

k*a*b
```

Expression

See Also   KineticLawName, Parameters, ParameterVariableNames,
ParameterVariables, ReactionRate, sbioaddtolibrary, sbiowhos,
SpeciesVariables, SpeciesVariableNames

Purpose   Integer identifying each group in data set

Description GroupID is a property of the PKData object. It is an array of the same length as the DataSet property containing an integer to identify each group. PKData sets this property during construction of the PKData object.

Characteristics

Applies to	Object: PKData
Data type	double
Data values	Index value for each group
Access	Read-only

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, PKData object

GroupLabel

Purpose   Identify group column in data set

Description   `GroupLabel` is a property of the `PKData` object. It specifies the column in DataSet that contains the group identification labels.

Characteristics

Applies to	Object: `PKData`
Data type	`char string`
Data values	Column header string from imported data set
Access	Read/write

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`, `GroupNames`

Purpose   Unique values from GroupLabel in data set

Description   GroupNames is a property of the PKData object. It contains unique values
                     from the data column specified by the GroupLabel property. PKData
                     sets this property during construction of the PKData object.

Characteristics

Applies to	Object: PKData
Data type	char string or cell array of strings
Data values	Unique values in GroupLabel
Access	Read-only

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's
                     Guide, PKData object, GroupLabel

HasResponseVariable

Purpose Compartment drug concentration reported

Description HasResponseVariable is a property of the PKCompartment object. It is a logical indicating if the drug concentration in this compartment is reported.

Characteristics

Applies to	Objects: PKCompartment
Data type	logical
Data values	1 (true) or 0 (false).
Access	Read/write

See Also addCompartment, DosingType, EliminationType, PKCompartment object

Purpose   Identify independent variable column in data set

Description   `IndependentVarLabel` is a property of the `PKData` object. It specifies the column in DataSet that contains the independent variable (for example, time).

Characteristics

Applies to	Object: `PKData`
Data type	`char string`
Data values	Column header from imported data set
Access	Read/write

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`

InitialAmount

Purpose	Species initial amount

Description   The InitialAmount property indicates the initial quantity of the
SimBiology species object. InitialAmount is the quantity of the species
before the simulation starts.

Characteristics

Applies to	Object: species
Data type	double
Data values	Positive real number. Default value is 0.
Access	Read/write

Examples   Add a species to a model and set the initial amount of the species.

1 Create a model object named my_model.

```
modelObj = sbiomodel ('my_model');
```

2 Add the species object named glucose.

```
speciesObj = addspecies (modelObj, 'glucose');
```

3 Set the initial amount to 100 and verify.

```
set (speciesObj, 'InitialAmount',100);
get (speciesObj, 'InitialAmount')
```

MATLAB returns:

```
ans =

    100
```

See Also   addspecies, InitialAmountUnits

Purpose   Species initial amount units

Description   The `InitialAmountUnits` property indicates the unit definition for the
                  `InitialAmount` property of a species object. `InitialAmountUnits` can
                  be one of the built-in units. To get a list of the defined units, use the
                  `sbioshowunits` function. If `InitialAmountUnits` changes from one unit
                  definition to another, `InitialAmount` does not automatically convert to
                  the new units. The `sbioconvertunits` function does this conversion.
                  To add a user-defined unit to the list, see `sbioregisterunit`.

                  See `DefaultSpeciesDimension` for more information on specifying
                  dimensions for species quantities. `InitialAmountUnits` must have
                  corresponding dimensions to `CapacityUnits`. For example, if the
                  `CapacityUnits` are `meter`^{2}, then species must be `amount/meter`^{2} or
                  `amount`.

Characteristics

Applies to	Object: species
Data type	`char string`
Data values	Units from library with dimensions of amount, amount/length, amount/area, or amount/volume. Default is `''` (empty).
Access	Read/write

Examples   **1** Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');
compObj = addcompartment(modelObj, 'cell');
```

                 2 Add a species object named `glucose`.

```
speciesObj = addspecies (compObj, 'glucose');
```

                 3 Set the initial amount to `100`, `InitialAmountUnits` to `molecule`,
                 and verify.

InitialAmountUnits

```
set (speciesObj,'InitialAmountUnits','molecule');
get (speciesObj,'InitialAmountUnits')
```

MATLAB returns:

```
ans =

molecule
```

See Also   DefaultSpeciesDimension, InitialAmount, sbioconvertunits,
sbioregisterunit, sbioshowunits

Purpose   Show kinetic law used for `ReactionRate`

Description   The `KineticLaw` property shows the kinetic law that determines the reaction rate specified in the `ReactionRate` property of the reaction object. This property shows the kinetic law used to define `ReactionRate`.

KineticLaw can be configured with the `addkineticlaw` method. The `addkineticlaw` function configures the `ReactionRate` based on the `KineticLaw` and the species and parameters specified in the kinetic law object properties `SpeciesVariableNames` and `ParameterVariableNames`. `SpeciesVariableNames` are determined automatically for mass action kinetics.

If the reaction is updated, the `ReactionRate` is automatically updated only for mass action kinetics. For all other kinetics, the `SpeciesVariableNames` property of the kinetic law object should be reconfigured.

Characteristics

Applies to	Object: reaction
Data type	Kinetic law object
Data values	Kinetic law object. Default is `[]` (empty).
Access	Read-only

Examples   Example with `Henri-Michaelis-Menten` kinetics

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Verify that the `KineticLaw` property for the reaction object is
`Henri-Michaelis-Menten`.

```
get (reactionObj, 'KineticLaw')
```

MATLAB returns:

```
SimBiology Kinetic Law Array

 Index:    KineticLawName:
    1      Henri-Michaelis-Menten
```

See Also   `KineticLawName`, `Parameters`, `ParameterVariableNames`,
`ReactionRate`, `SpeciesVariableNames`

KineticLawName

Purpose   Name of kinetic law applied to reaction

Description The KineticLawName property of the kinetic law object indicates the name of the abstract kinetic law applied to the reaction. KineticLawName can be any valid name from the built-in or user-defined abstract kinetic law library. See "Abstract Kinetic Law" on page 6-56 for a definition and more information.

You can find the KineticLawName list in the abstract kinetic law library by using the command sbiowhos -kineticlaw (sbiowhos). You can create an abstract kinetic law with the function sbioabstractkineticlaw and add it to the library using sbioaddtolibrary.

Characteristics

Applies to	Object: kineticlaw
Data type	char string
Data values	char string defined by abstract kinetic law
Access	Read-only

Examples **1** Create a model object, add a reaction object, and define a kinetic law for the reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

2 Verify the KineticLawName of kineticlawObj.

```
get (kineticlawObj, 'KineticLawName')
```

MATLAB returns:

```
ans =
```

KineticLawName

```
Henri-Michaelis-Menten
```

See Also

Expression, Parameters, ParameterVariableNames,
ParameterVariables, ReactionRate, sbioaddtolibrary, sbiowhos,
SpeciesVariables, SpeciesVariableNames

Purpose   Specify recorded simulation output frequency

Description   The LogDecimation property defines how often the simulation data is recorded as output. It is a property of the SolverOptions object. SolverOptions is a property of the configset object. LogDecimation is available for ssa, expltau, and inmpltau solvers.

Use LogDecimation to specify how frequently you want to record the output of the simulation. For example, if the LogDecimation is set to 1, for the command (t,x) = sbiosimulate(modelObj), at each simulation step the time will be logged in t and the quantity of each logged species will be logged as a row in x. If LogDecimation is 10, then every 10th simulation step will be logged in t and x.

Characteristics

Applies to	Object: SolverOptions
Data type	int
Data values	>0. Default is 1.
Access	Read/write

Examples   This example shows how to change LogDecimation settings.

1 Retrieve the configset object from the modelObj, and change the SolverType to expltau.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'expltau')
```

2 Change the LogDecimation to 10.

```
set(configsetObj.SolverOptions, 'LogDecimation', 10);
get(configsetObj.SolverOptions, 'LogDecimation')

ans =
```

LogDecimation

10

See Also   `ErrorTolerance, RandomState`

Purpose   Specify nonlinear solver maximum iterations in implicit tau

Description   The MaxIterations property specifies the maximum number of iterations for the nonlinear solver in impltau. It is a property of the SolverOptions object. SolverOptions is a property of the configset object.

The implicit tau solver in SimBiology software internally uses a nonlinear solver to solve a set of algebraic nonlinear equations at every simulation step. Starting with an initial guess at the solution, the nonlinear solver iteratively tries to find the solution to the algebraic equations. The closer the initial guess is to the solution, the fewer the iterations the nonlinear solver will take before it finds a solution. MaxIterations specifies the maximum number of iterations the nonlinear solver should take before it issues a "failed to converge" error. If you get this error during simulation, try increasing MaxIterations. The default value of MaxIterations is 15.

Characteristics

Applies to	Object: SolverOptions
Data type	int
Data values	>0. Default is 15.
Access	Read/write

Examples   This example shows how to change MaxIterations settings.

1 Retrieve the configset object from the modelObj, and change the SolverType to impltau.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'impltau');
```

2 Change the MaxIterations to 25.

```
set(configsetObj.SolverOptions, 'MaxIterations', 25);
```

MaxIterations

```
get(configsetObj.SolverOptions, 'MaxIterations')

ans =

    25
```

See Also   ErrorTolerance, LogDecimation, RandomState

Purpose   Specify upper bound on solver step size

Description   The MaxStep property specifies the size of the bounds on the size of the time steps. If the differential equation has periodic coefficients or solutions, it might be a good idea to set MaxStep to some fraction (such as 1/4) of the period. This guarantees that the solver does not enlarge the time step too much and step over a period of interest. For more information on MaxStep, see odeset in the MATLAB documentation.

Characteristics

Applies to	Object: SolverOptions
Data type	Positive scalar
Data values	{0.1*abs(t0-tf)}. Default is [] (empty).
Access	Read/write

See Also   SimBiology property RelativeTolerance

MATLAB function odeset

ModelName

Purpose	Name of model simulated
Description	The ModelName property shows the name of the model for which the SimData object contains the simulation data.

Characteristics

Applies to	Object: SimData
Data type	string
Data values	Default value is '' (empty).
Access	Read-only

See Also   Data, DataInfo

Purpose   Contain all model objects

Note The Models property will be removed in a future version. Submodels will not be supported in future releases. Use the function sbioupdate to convert submodels into models.

Description   The Models property shows the submodels in a model object or models in the SimBiology root. Read-only array of model objects. SimBiology has a hierarchical organization. A top-level model object has the SimBiology root as its Parent. Model objects with another model object as Parent are submodels. For a model object to access configset, kinetic law, reaction, rule and species objects, you must assign the model object as Parent in these objects. Parameter objects can have a model object or kinetic law object as Parent. You can display all the component objects with *modelObj*.Models or get (*modelObj*, 'Models').

The components of a submodel are contained within the submodel. In addition, a submodel object can reference parameter variables that have been assigned to the model object. For example, a parameter defined within a submodel cannot be used by the parent model or another model object. A submodel object however, can use the parameters assigned to the model object.

You can add a submodel to a model object with the method addmodel and remove it from its parent with the method delete.

Characteristics

Applies to	Objects: model, root
Data type	Array of model objects
Data values	Model object. Default is [] (empty).
Access	Read-only

See Also   sbiomodel, sbioupdate

Multiplier

Purpose   Relationship between defined unit and base unit

Description   The `Multiplier` is the numerical value that defines the relationship between the unit `Name` and the base unit as a product of the `Multiplier` and the base unit. For example, in `Celsius = (5/9)*( Fahrenheit-32);` `Multiplier` is 5/9 and `Offset` is -32. For `1 mole = 6.0221e23*molecule`, the `Multiplier` is `6.0221e23`.

Characteristics

Applies to	Object: Unit
Data type	`double`
Data values	Nonzero real number. Default value is 1.
Access	Read/write

Examples   This example shows how to create a user-defined unit, add it to the user-defined library, and query the library.

1 Create a user-defined unit called `usermole`, whose composition is `molecule` and `Multiplier` property is `6.0221e23`.

```
unitObj = sbiounit('usermole', 'molecule', 6.0221e23);
```

2 Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

3 Query the `Multiplier` property.

```
get(unitObj, 'Multiplier')

ans =

1/molarity*second
```

See Also   Composition, get, Offset, sbiounit, set

Purpose   Specify name of object

Description The Name property identifies a SimBiology object. Compartments, species, parameters, and model objects can be referenced by other objects using the Name property, therefore Name must be unique for these objects. However, species names need only be unique within each compartment. Parameter names must be unique within a model (if at the model level), or within each kinetic law (if at the kinetic law level). This means that you can have nonunique species names if the species are in different compartments, and nonunique parameter names if the parameters are in different kinetic laws or at different levels. Note that having nonunique parameter names can cause the model to have shadowed parameters and that may not be best modeling practice. For more information on levels of parameters, see "Definition of Parameter Scope" in the SimBiology User's Guide documentation.

Use the function sbioselect to find an object with the same Name property value.

In addition, note the following constraints and reserved characters for the Name property in objects:

- Model names cannot be empty.

- Parameter names cannot be empty, or have the name time.

- If you have a parameter, a species, or compartment name that is not a valid MATLAB variable name, when you write an event function, an event trigger, a reaction, reaction rate equation, or a rule you must enclose that name in brackets. For example, enclose [DNA polymerase+] in brackets. In addition, if you have the same species in multiple compartments you must qualify the species with the compartment name, for example, nucleus.[DNA polymerase+], [nuclear complex].[DNA polymerase+].

- Species and compartment names cannot be empty, and note the following reserved words, characters, and constraints:

- The literal words `null` and `time`. Note that you can specify species names with these words contained within the name. For example, `nullaminoacids` or `nullnucleotides`.

- The characters `->`, `< >`, `[`, and `]`.

For more information on valid MATLAB variable names, see `genvarname` and `isvarname`.

Characteristics

Applies to	Objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, rule, species, unit, or variant
Data type	`char string`
Data values	Any `char string` except reserved words and characters
Access	Read/write

Examples

1 Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');
```

2 Add a reaction object to the model object. Note the use of brackets because the names are not valid MATLAB variable names.

```
reactionObj = addreaction(modelObj, '[Aspartic acid] -> [beta-Aspartyl-PO4]')
```

MATLAB returns:

```
SimBiology Reaction Array

Index:    Reaction:
  1       [Aspartic acid] -> [beta-Aspartyl-PO4]
```

3 Set the reaction `Name` and verify.

```
set (reactionObj, 'Name', 'Aspartate kinase reaction');
get (reactionObj, 'Name')
```

MATLAB returns:

```
ans =

   Aspartate kinase reaction
```

See Also   addcompartment, addkineticlaw, addmodel, addparameter, addreaction, addrule, addspecies, sbiomodel, sbiounit, sbiounitprefix

Normalization

Purpose   Specify normalization type for sensitivity analysis

Description   `Normalization` is a property of the `SensitivityAnalysisOptions` object. `SensitivityAnalysisOptions` is a property of the configuration set object. Use `Normalization` to specify the normalization for the computed sensitivities.

The following values let you specify the type of normalization. The examples show you how sensitivities of a species `x` with respect to a parameter `k` are calculated for each normalization type:

- `'None'` specifies no normalization.

$$\frac{dx(t)}{dk}$$

- `'Half'` specifies normalization relative to the numerator (species quantity) only.

$$\left(\frac{1}{x(t)}\right)\left(\frac{dx(t)}{dk}\right)$$

- `'Full'` specifies that the data should be made dimensionless.

$$\left(\frac{k}{x(t)}\right)\left(\frac{dx(t)}{dk}\right)$$

Characteristics

Applies to	Object: `SensitivityAnalysisOptions`
Data type	enum
Data values	`'None'`, `'Half'`, `'Full'`. Default is `'None'`.
Access	Read/write

See Also   `ParameterInputFactors`, `SensitivityAnalysis`, `SensitivityAnalysisOptions`, `SpeciesInputFactors`

Purpose   HTML text describing SimBiology object

Description Use the Notes property of an object to store comments about the object.
                 You can include HTML tagging in the notes to render formatted text
                 in the SimBiology desktop.

Characteristics

Applies to	Objects: compartment, kinetic law, model, parameter, reaction, rule, species, unit, or unit prefix
Data type	char string
Data values	Any char string
Access	Read/write

Examples   **1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Write notes for the model object.

```
set (modelObj, 'notes', '09/01/05 experimental data')
```

3 Verify the assignment.

```
get (modelObj, 'notes')
```

   MATLAB returns:

```
ans =

09/01/05 experimental data
```

See Also   addkineticlaw, addmodel, addparameter, addreaction, addrule,
                addspecies, sbiomodel, sbiounit, sbiounitprefix

Observed

Purpose   Measured response object name

Description `Observed` is a property of the `PKModelMap` object. It specifies the name of an object that represents the measured response (the response variable). Specify the name of a species, compartment, or parameter.

Characteristics

Applies to	Object: `PKModelMap`
Data type	`char string` or cell array of strings
Data values	Names of objects
Access	Read/write

See Also   "Defining Model Components for Observed Response, Dose, Dosing Type, and Parameters to be Estimated" in the SimBiology User's Guide, `Dosed`, `Estimated`, `PKModelMap` object

Purpose   Unit composition modifier

Description

Note The `Offset` property is currently not supported.

The `Offset` is the numerical value by which the unit composition is modified from the base unit. For example, Celsius = (5/9)*(Fahrenheit-32); Multiplier is 5/9 and Offset is -32.

Characteristics

Applies to	Object: Unit
Data type	`double`
Data values	Real number. Default is `0`.
Access	Read/write

Examples   This example shows how to create a user-defined unit, add it to the user-defined library, and query the library.

1 Create a user-defined unit called `celsius2`, whose composition refers to `fahrenheit`, `Multiplier` property is `9/5`, and `Offset` property is `32`.

```
unitObj = sbiounit('celsius2','fahrenheit',9/5,32);
```

2 Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

3 Query the `Offset` property.

```
get(unitObj, 'Offset')

ans =
```

Offset

32

See Also   Composition, get, Multiplier, sbioaddtolibrary, sbioshowunits, sbiounit, set

Purpose   Owning compartment

Description   Owner shows you the SimBiology compartment object that owns
the compartment object. In the compartment object, the Owner
property shows you whether the compartment resides within another
compartment. The Compartments property indicates whether other
compartments reside within the compartment. You can add a
compartment object using the method addcompartment.

Characteristics

Applies to	Object: compartment
Data type	char string
Data values	Name of compartment object. Default is [ ].
Access	Read-only

Examples   **1** Create a model object named modelObj.

    modelObj = sbiomodel('cell');

2 Add two compartments to the model object.

    compartmentObj1 = addcompartment(modelObj, 'nucleus');
    compartmentObj2 = addcompartment(modelObj, 'mitochondrion');

3 Add a compartment to one of the compartment objects.

    compartmentObj3 = addcompartment(compartmentObj2, 'matrix');

4 Display the Owner property in the compartment objects.

    get(compartmentObj3, 'Owner')

The result shows you the owning compartment and its components:

    SimBiology Compartment - mitochondrion

```
Compartment Components:
  Capacity:          1
  CapacityUnits:
  Compartments:      1
  ConstantCapacity:  true
  Owner:
  Species:           0
```

5 Change the owning compartment.

```
set(compartmentObj3, 'Owner', compartmentObj1)
```

See Also   Compartments, Parent

Purpose   Specify parameter input factors for sensitivity analysis

Description   ParameterInputFactors is a property of the
SensitivityAnalysisOptions object. SensitivityAnalysisOptions
is a property of the configuration set object. Use
ParameterInputFactors to specify the parameters with respect to
which you want to compute the sensitivities of the species states in your
model. When you simulate a model with SensitivityAnalysis enabled
in the active configuration set object, sensitivity analysis returns the
computed sensitivities of the species specified in StatesToLog. For
a description of the output, see the SensitivityAnalysisOptions
property description.

Characteristics

Applies to	Object: SensitivityAnalysisOptions
Data type	Parameter object or array of parameter objects
Data values	Parameter object array. Default is [ ] (empty).
Access	Read/write

Examples   This example shows how to set ParameterInputFactors for sensitivity
analysis.

1 Import the radio decay model from the SimBiology demos.

```
modelObj  = sbmlimport('radiodecay');
```

2 Retrieve the configuration set object from modelObj.

```
configsetObj = getconfigset(modelObj);
```

3 Add a parameter to the ParameterInputFactors property and
display. Use the sbioselect function to retrieve the parameter
object from the model.

ParameterInputFactors

```
set(configsetObj.SensitivityAnalysisOptions,'ParameterInputFactors', ...
   sbioselect(modelObj, 'Type', 'parameter', 'Name', 'c'));
get (configsetObj.SensitivityAnalysisOptions, 'ParameterInputFactors')


SimBiology Parameter Array


Index:    Name:     Value:      ValueUnits:
 1         c         0.5         1/second
```

See Also   sbioselect, SensitivityAnalysis, SensitivityAnalysisOptions, SpeciesInputFactors

Purpose   Array of parameter objects

Description   The `Parameters` property indicates the parameters in a `Model` or `KineticLaw` object. Read-only array of `Parameter` objects. Display with `modelObj.Parameters` or `get(modelObj, 'Parameters')`.

The scope of a parameter object is hierarchical and is defined by the parameter's parent. If a parameter is defined with a kinetic law object as its parent, then only the kinetic law object can use the parameter. If a parameter object is defined with a model object as its parent, then components such as rules, events, and kinetic laws (reaction rate equations) can use the parameter.

You can add a parameter to a model object, or kinetic law object with the method `addparameter` and delete it with the method `delete`.

You can view parameter object properties with the `get` command and configure properties with the `set` command.

Characteristics

Applies to	Objects: `model`, `kineticlaw`
Data type	Array of parameter objects
Data values	Parameter objects. Default value is `[]` (empty).
Access	Read-only

Examples   **1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

3 Add a parameter and assign it to the kinetic law object (kineticlawObj);.

Parameters

```
parameterObj1 = addparameter (kineticlawObj, 'K1');
get (kineticlawObj, 'Parameters')

SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1        K1        1
```

4 Add a parameter and assign it to the model object (modelObj).

```
parameterObj1 = addparameter(modelObj, 'K2');
get(modelObj, 'Parameters')
SimBiology Parameter Array

 Index:    Name:    Value:    ValueUnits:
  1        K2        1
```

See Also   addparameter, delete, get, sbioparameter, set

Purpose   Cell array of reaction rate parameters

Description   The ParameterVariableNames property shows the parameters used by the kinetic law object to determine the ReactionRate equation in the reaction object. Use setparameter to assign ParameterVariableNames. When you assign species to ParameterVariableNames, SimBiology software maps these parameter names to ParameterVariables in the kinetic law object.

If the reaction is using a kinetic law, the ReactionRate property of a reaction object shows the result of a mapping from an abstract kinetic law. The ReactionRate is determined by the kinetic law object Expression property by mapping ParameterVariableNames to ParameterVariables and SpeciesVariableNames to SpeciesVariables.

Characteristics

Applies to	Object: kineticlaw
Data type	Cell array of strings
Data values	Cell array of parameters
Access	Read/write

Examples   Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj KineticLaw property is configured to kineticlawObj.

ParameterVariableNames

3 The `'Henri-Michaelis-Menten'` kinetic law has two parameter
variables (Vm and Km) that should to be set. To set these variables:

```
setparameter(kineticlawObj,'Vm', 'Va');
setparameter(kineticlawObj,'Km', 'Ka');
```

4 Verify that the parameter variables are correct.

```
get (kineticlawObj, 'ParameterVariableNames')
```

MATLAB returns:

```
ans =

    'Va'    'Ka'
```

See Also   Expression, ParameterVariables, ReactionRate, setparameter,
SpeciesVariables, SpeciesVariableNames

Purpose   Parameters in abstract kinetic law

Description   The ParameterVariables property shows the parameter variables that
are used in the Expression property of the abstract kinetic law object.
Used to determine the ReactionRate equation in the reaction object.
Use the MATLAB function set to assign ParameterVariables to an
abstract kinetic law. For more information, see abstract kinetic law.

Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Defined by abstract kinetic law
Access	Read/write in abstract kinetic law. Read-only in kinetic law.

Examples   Create a model, add a reaction, and assign the SpeciesVariableNames
for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of the type
'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj KineticLaw property is configured to kineticlawObj.

3 The 'Henri-Michaelis-Menten' kinetic law has two parameter
variables. To set these variables:

```
get (kineticlawObj, 'ParameterVariables')
```

MATLAB returns:

ParameterVariables

```
ans =

    'Vm'    'Km'
```

See Also   Expression, ParameterVariableNames, ReactionRate, set, setparameter, SpeciesVariables, SpeciesVariableNames

Purpose   Indicate parent object

Description   The Parent property indicates the parent object for a SimBiology object (read-only). The Parent property indicates accessibility of the object. The object is accessible to the Parent object and other objects within the Parent object. The value of Parent depends on the type of object and how it was created. All models always have the SimBiology root as the Parent.

More Information

The following table shows you the different objects and the possible Parent value.

Object	Parent
abstract kinetic law	• [] (empty) until added to library • root object upon addition to library
compartment	model object
event	model object or [] (empty)
kinetic law	reaction object
model	root object
parameter	model object, kinetic law object, or [] (empty)
reaction	model object or [] (empty)
rule	model object or [] (empty)
species	compartment

Parent

Object	Parent
variant	model object or [] (empty)
unit and unit prefixes	• [] (empty) until added to library • root object upon addition to library

Characteristics

Applies to	Objects: abstract kinetic law, compartment, event, kinetic law, model, parameter, reaction, rule, species, variant, unit, or unit prefix
Data type	Object
Data values	SimBiology component object or [] (empty)
Access	Read-only

See Also   addkineticlaw, addmodel, addparameter, addreaction, sbiomodel

Purpose   Hold compartments in PK model

Description   PKCompartments is a property of the PKModelDesign object. It is used to specify the compartments in the PKModelDesign object. Each compartment is a PKCompartment object added using the addCompartment method.

Characteristics

Applies to	Objects: PKModelDesign
Data type	object
Data values	PKCompartment object
Access	Read-only

See Also   "Creating PK Models at the Command Line" in the SimBiology User's Guide, addCompartment, PKCompartment object, PKModelDesign object

Products

Purpose	Array of reaction products

Description   The Products property contains an array of SimBiology.Species objects.

Products is a 1-by-n species object array that indicates the species that are changed by the reaction. If the Reaction property is modified to use a different species, the Products property is updated accordingly.

You can add product species to the reaction with addproduct function. You can remove product species from the reaction with rmproduct. You can also update reaction products by setting the Reaction property with the function set.

Characteristics

Applies to	Object: reaction
Data type	Array of objects
Data values	Species objects. Default is [] (empty).
Access	Read-only

Examples   **1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add reaction objects.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

3 Verify the assignment.

```
productsObj = get(reactionObj, 'Products')
```

MATLAB returns:

```
SimBiology Species Array

Index:  Compartment:  Name:  InitialAmount: InitialAmountUnits:
```

```
                          1        unnamed        c        0
                          2        unnamed        d        0
```

See Also   addkineticlaw, addproduct, addspecies, rmproduct

RandomState

Purpose	Set random number generator
Description	The RandomState property sets the random number generator for the stochastic solvers. It is a property of the SolverOptions object. SolverOptions is a property of the configset object.

SimBiology software uses a pseudorandom number generator. The sequence of numbers generated is determined by the state of the generator, which can be specified by the integer RandomState. If RandomState is set to integer J, the random number generator is initialized to its Jth state. The random number generator can generate all the floating-point numbers in the closed interval [2^(-53), 1-2^(-53)]. Theoretically, it can generate over 2^1492 values before repeating itself. But for a given state, the sequence of numbers generated will be the same. To change the sequence, change RandomState. SimBiology software resets the state at startup. The default value of RandomState is [].

Characteristics

Applies to	Objects: SolverOptions for SSA, expltau, impltau
Data type	int
Data values	Default is [] (empty).
Access	Read/write

Examples   This example shows how to change RandomState settings.

1 Retrieve the configset object from the modelObj and change the SolverType to expltau.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'expltau')
```

2 Change the Randomstate to 5.

```
set(configsetObj.SolverOptions, 'RandomState', 5);
get(configsetObj.SolverOptions, 'RandomState'))


ans =

  5
```

See Also   ErrorTolerance, LogDecimation, MaxIterations

RateLabel

Purpose   Identify rate of infusion column in data set

Description   RateLabel is a property of the PKData object. It specifies the column in DataSet that contains the rate of infusion. This applies only when dosing type is infusion. The data set must contain the rate and not an infusion time.

Characteristics

Applies to	Object: PKData
Data type	char string
Data values	Column header string from imported data set
Access	Read/write

See Also   "Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, PKData object, DosingType

Purpose   Array of reaction reactants

Description   The Reactants property is a 1-by-n species object array of reactants
in the reaction. If the Reaction property is modified to use a different
reactant, the Reactants property will be updated accordingly.

You can add reactant species to the reaction with the addreactant
method.

You can remove reactant species from the reaction with the rmreactant
method. You can also update reactants by setting the Reaction property
with the function set.

Characteristics

Applies to	Object: reaction
Data type	Species object or array of species objects
Data values	Species objects. Default is [] (empty).
Access	Read-only

Examples   **1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add reaction objects.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

3 View the reactants for reactionObj.

```
get(reactionObj, 'Reactants')
```

MATLAB returns:

```
SimBiology Species Array

Index: Compartment: Name: InitialAmount: InitialAmountUnits:
 1     unnamed       a         0
```

Reactants

```
        2       unnamed       b       O
```

See Also   addreactant, addreaction, addspecies, rmreactant

Purpose   Reaction object reaction

Description   Property to indicate the reaction represented in the reaction object. Indicates the chemical reaction that can change the amount of one or more species, for example, `'A + B  > C'`. This property is different from the model object property called Reactions.

See addreaction for more information on how the Reaction property is set.

Characteristics

Applies to	Object: reaction
Data type	char string
Data values	Valid reaction string. Default is `''` (empty).
Access	Read/write

Examples   **1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Verify that the reaction property records the input.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =

a + b -> c + d
```

See Also   addreaction, sbioreaction

ReactionRate

Purpose   Reaction rate equation in reaction object

Description   The `ReactionRate` property defines the reaction rate equation. You can define a `ReactionRate` with or without the `KineticLaw` property. `KineticLaw` defines the type of reaction rate. The `addkineticlaw` function configures the `ReactionRate` based on the `KineticLaw` and the species and parameters specified in the kinetic law object properties `SpeciesVariableNames` and `ParameterVariableNames`.

The reaction takes place in the reverse direction if the `Reversible` property is true. This is reflected in `ReactionRate`. The `ReactionRate` includes the forward and reverse rate if reversible.

You can specify `ReactionRate` without `KineticLaw`. Use the `set` function to specify the reaction rate equation. SimBiology software adds species variables while creating `reactionObj` using the `addreaction` method. You must add the parameter variables (to the `modelObj` in this case). See the example below.

After you specify the `ReactionRate` without `KineticLaw` and you later configure the `reactionObj` to use `KineticLaw`, the `ReactionRate` is unset until you specify `SpeciesVariableNames` and `ParameterVariableNames`.

Characteristics

Applies to	Object: reaction
Data type	char string
Data values	Reaction rate string. Default is `''` (empty).
Access	Read/write

Examples   **Example 1**

Create a model, add a reaction, and assign the expression for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj KineticLaw property is configured to kineticlawObj.

3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that you should set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm_d and Km_d and assign them to kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

4 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Vm_d' 'Km_d'});
set(kineticlawObj,'SpeciesVariableNames', {'a'});
```

5 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Vm_d*a/(Km_d + a)
```

Example 2

Create a model, add a reaction, and specify ReactionRate without a kinetic law.

ReactionRate

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a + b -> c + d');
```

2 Specify ReactionRate and verify the assignment.

```
set (reactionObj, 'ReactionRate', 'k*a');
get(reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

k*a
```

3 You cannot simulate the model until you add the parameter k to the modelObj.

```
parameterObj = addparameter(modelObj, 'k');
```

SimBiology adds the parameter to the modelObj with default Value = 1.0 for the parameter.

See Also   addparameter, addreaction, Reversible, sbioparameter, sbioreaction

Purpose   Array of reaction objects

Description   Property to indicate the reactions in a Model object. Read-only array of reaction objects.

A reaction object defines a chemical reaction that occurs between species. The species for the reaction are defined in the Model object property Species.

You can add a reaction to a model object with the method addreaction, and you can remove a reaction from the model object with the method delete.

Characteristics

Applies to	Object: model
Data type	Array of reaction objects
Data values	Reaction object
Access	Read-only

Examples   **1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Verify that the reactions property records the input.

```
get (modelObj, 'Reactions')
```

MATLAB returns:

```
SimBiology Reaction Array

 Index:   Reaction:
   1      a + b -> c + d
```

See Also   addreaction, delete, sbioreaction

RelativeTolerance

Purpose   Specify allowable error relative to component

Description The `RelativeTolerance` property specifies the allowable error tolerance relative to the state vector at each simulation step. The state vector contains values for all the state variables, for example, species amounts for all the species.

`RelativeTolerance` is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object. `RelativeTolerance` is available for the ode solvers (`'ode45'`, `'ode23'`, `'ode113'`, `'ode15s'`, `'ode23s'`, `'ode23t'`, and `'ode23tb'`).

If you set the `RelativeTolerance` at `1e-2`, you are specifying that an error of 1% relative to each state value is acceptable at each simulation step.

At each simulation step, the solver estimates the local error e_i in the `i`th state vector `y`. Simulation converges at that time step if e_i satisfies the following equation:

`|`e_i`|`≤`max(RelativeTolerance*|`y_i`|,AbsoluteTolerance)`

Thus at higher state values, convergence is determined by `RelativeTolerance`. As the state values approach zero, convergence is controlled by `AbsoluteTolerance`. The choice of values for `RelativeTolerance` and `AbsoluteTolerance` will vary depending on the problem. The default values should work for first trials of the simulation; however if you want to optimize the solution, consider that there is a trade-off between speed and accuracy. If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy. If the results appear to be inaccurate, you can decrease the tolerance values but this will slow down the solver. If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`

Data values	>0, <1. Default is `1e-3`.
Access	Read/write

Examples This example shows how to change AbsoluteTolerance.

1 Retrieve the `configset` object from the `modelObj`.

```
modelObj   = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
```

2 Change the AbsoluteTolerance to `1e-8`.

```
set(configsetObj.SolverOptions, 'RelativeTolerance', 1.0e-6);
get(configsetObj.SolverOptions, 'RelativeTolerance')

ans =

   1.0000e-006
```

See Also AbsoluteTolerance

Reversible

Purpose Specify whether reaction is reversible or irreversible

Description The Reversible property defines whether a reaction is reversible or irreversible. The rate of the reaction is defined by the ReactionRate property. For a reversible reaction, the reaction rate equation is the sum of the rate of the forward and reverse reactions. The type of reaction rate is defined by the KineticLaw property. If a reaction is changed from reversible to irreversible or vice versa after KineticLaw is assigned, the new ReactionRate is determined only if Type is MassAction. All other Types result in unchanged ReactionRate. For MassAction, the first parameter specified is assumed to be the rate of the forward reaction.

Characteristics

Applies to	Object: reaction
Data type	boolean
Data values	true, false. Default value is false.
Access	Read/write

Examples Create a model, add a reaction, and assign the expression for the reaction rate equation.

1 Create model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Set the Reversible property for the reactionObj to true and verify this setting.

```
set (reactionObj, 'Reversible', true)
get (reactionObj, 'Reversible')
```

MATLAB returns:

```
ans =
```

1

MATLAB returns 1 for `true` and 0 for `false`.

In the next steps the example illustrates how the reaction rate equation is assigned for reversible reactions.

3 Create a kinetic law object for the reaction object of the type `'MassAction'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

reactionObj KineticLaw property is configured to kineticlawObj.

4 The `'MassAction'` kinetic law for reversible reactions has two parameter variables (`'Forward Rate Parameter'` and `'Reverse Rate Parameter'`) that you should set. The species variables for MassAction are automatically determined. To set the parameter variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) named Kf and Kr and assign the object to kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Kf');
parameterObj2 = addparameter(kineticlawObj, 'Kr');
```

5 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Kf' 'Kr'});
```

6 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Kf*a*b - Kr*c*d
```

Reversible

See Also   addparameter, addreactant, addreaction, ParameterVariableNames, ReactionRate, sbioreaction

Purpose   Specify species and parameter interactions

Description The Rule property contains a rule that defines how certain species and
parameters should interact with one another. For example, a rule could
state that the total number of species A and species B must be some
value. Rule is a MATLAB expression that defines the change in the
species object quantity or a parameter object Value when the rule is
evaluated.

You can add a rule to a model object with the addrule method and
remove the rule with the delete method. For more information on
rules, see addrule and RuleType.

Characteristics

Applies to	Object: rule
Data type	char string
Data values	char string defined as species or parameter objects. Default is empty.
Access	Read/write

Examples **1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Add a rule.

```
ruleObj = addrule(modelObj, '10-a+b')
```

MATLAB returns:

```
SimBiology Rule Array

Index:    RuleType:    Rule:
1         algebraic    10-a+b
```

Rule

See Also   `addrule`, `delete`, `sbiorule`

Purpose   Specify type of rule for rule object

Description   The `RuleType` property indicates the type of rule defined by the rule object. A `Rule` object defines how certain species, parameters, and compartments should interact with one another. For example, a rule could state that the total number of species A and species B must be some value. `Rule` is a MATLAB expression that defines the change in the species object quantity or a parameter object `Value` when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule`.

The types of rules in SimBiology are as follows:

- `initialAssignment` — Lets you specify the initial value of a parameter, species, or compartment capacity, as a function of other model component values in the model.

- `repeatedAssignment` — Lets you specify a value that holds at all times during simulation, and is a function of other model component values in the model.

- `algebraic` — Lets you specify mathematical constraints on one or more parameters, species, or compartments that must hold during a simulation.

- `rate` — Lets you specify the time derivative of a parameter value, species amount, or compartment capacity.

Constraints on Varying Species Using a Rate Rule

If the model has a species defined in concentration, being varied by a `rate` rule, and it is in a compartment with varying volume, you can only use `rate` or `initialAssigment` rules to vary the compartment volume.

Conversely, if you are varying a compartment's volume using a `repeatedAssignment` or `algebraic` rules, then you cannot vary a species (defined in concentration) within that compartment, with a `rate` rule.

RuleType

The reason for these constraints is that, if a species is defined in concentration and it is in a compartment with varying volume, the time derivative of that species is a function of the compartment's rate of change. For compartments varied by rate rules, the solver has that information.

Note that if you specify the species in amounts there are no constraints.

Characteristics

Applies to	Object: rule
Data type	char string
Data values	`'algebraic'`, `'assignment'`, `'rate'`. Default value is `'assignment'`.
Access	Read/write

Examples

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a -> b');
```

2 Add a rule that specifies the quantity of a species c. In the rule expression, k is the rate constant for a -> b.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

3 Change the RuleType from the default (`'algebraic'`) to `'rate'` and verify it using the get command.

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
     Active: 1
 Annotation: ''
       Name: ''
      Notes: ''
```

```
   Parent: [1x1 SimBiology.Model]
     Rule: 'c = k*(a+b)'
 RuleType: 'rate'
      Tag: ''
     Type: 'rule'
 UserData: []
```

See Also   "Changing Model Component Values Using Rules" in the *SimBiology User's Guide*, addrule, delete, sbiorule

Rules

Purpose	Array of rules in model object
Description	The `Rules` property shows the rules in a `Model` object. Read-only array of `SimBiology.Rule` objects.

A *rule* is a mathematical expression that modifies a species amount or a parameter value. A rule defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species A and species B must be some value.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule` and `RuleType`.

Characteristics

Applies to	Object: model
Data type	Array of rule objects
Data values	Rule object
Access	Read-only

Examples

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Add a rule.

```
ruleobj = addrule(modelObj, '10-a+b')
```

MATLAB returns:

```
SimBiology Rule Array

Index:    RuleType:    Rule:
 1        algebraic    10-a+b
```

See Also   `addrule`, `delete`, `sbiorule`

Purpose   Information about simulation

Description   The RunInfo property contains information describing the simulation
                  run that yielded the data in the SimData object.

                  The following information is stored:

                  • Configset — A struct form of the configuration set used during
                    simulation. This would typically be the model's active configset.

                  • Variant — A struct form of the variant(s) used during simulation.

                  • SimulationDate — The date/time of simulation.

                  • SimulationType — Either 'single run' or 'ensemble run',
                    depending on whether the data object was created using the function
                    sbiosimulate or the function sbioensemblerun.

Characteristics

Applies to	Object: SimData
Data type	struct
Data values	Default values are as follows:

```
ConfigSet: []
SimulationDate: ''
SimulationType: ''
Variant: []
```

                                       In practice, the ConfigSet,
                                       SimulationDate, and SimulationType
                                       fields are rarely empty, since they are
                                       populated after simulation.

Access	Read-only

See Also   StopTime, StopTimeType

RuntimeOptions

Purpose	Options for logged species
Description	The RuntimeOptions property holds options for species that will be logged during the simulation run. The run-time options object can be accessed through this property.
	The LogDecimation property of the configuration set object defines how often data is logged.

Property Summary

StatesToLog	Specify species data recorded
Type	Display top-level SimBiology object type

Characteristics

Applies to	Object: configset
Data type	Object
Data values	Run-time options
Access	Read-only

Examples

1 Create a model object, and retrieve its configuration set.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2 Retrieve the RuntimeOptions object from the configset object.

```
runtimeObj = get(configsetObj, 'RunTimeOptions')
Runtime Settings:

    StatesToLog:         all
```

See Also   get, set

SensitivityAnalysis

Purpose   Enable or disable sensitivity analysis

Description   The SensitivityAnalysis property lets you compute the time-dependent sensitivities of all the species states defined by the StatesToLog property with respect to the SpeciesInputFactors and the ParameterInputFactors that you specify in the SensitivityAnalysisOptions property of the configuration set object.

SensitivityAnalysis is a property of the SolverOptions object. SolverOptions is a property of the configuration set object. SensitivityAnalysis is available for the ode solvers ('ode45', 'ode23', 'ode113', 'ode15s', 'ode23s', 'ode23t', and 'ode23tb').

See SensitivityAnalysisOptions for more information on setting up sensitivity analysis. See "Sensitivity Analysis" in the SimBiology User's Guide documentation for a description of sensitivity analysis calculations.

Characteristics

Applies to	Object: SolverOptions
Data type	logical
Data values	1, 0, true, false. Default is false.
Access	Read/write

Examples   This example shows how to enable SensitivityAnalysis.

1 Retrieve the configset object from the modelObj.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2 Enable SensitivityAnalysis.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true);
get(configsetObj.SolverOptions, 'SensitivityAnalysis')
```

SensitivityAnalysis

```
ans =

    on
```

See Also   SensitivityAnalysisOptions, SolverOptions, SolverType,
StatesToLog

Purpose   Specify sensitivity analysis options

Description   The `SensitivityAnalysisOptions` property is an object that holds the sensitivity analysis options in the configuration set object. Sensitivity analysis is only supported for deterministic (ODE) simulations.

Properties of `SensitivityAnalysisOptions` are summarized in "Property Summary" on page 6-130.

When sensitivity analysis is enabled, the following command

```
[t,x,names] = sbiosimulate(modelObj)
```

returns `[t,x,names]`, where

- `t` is an `n-by-1` vector, where `n` is the number of steps taken by the ode solver and `t` defines the time steps of the solver.

- `x` is an `n-by-m` matrix, where `n` is the number of steps taken by the ode solver and `m` is:

      ```
      Number of states specified in StatesToLog +
      (Number of species specified in StatesToLog*Number of input factors)
      ```

  A SimBiology state includes species and nonconstant parameters.

- `names` is the list of states logged and the list of sensitivities of the species specified in `StatesToLog` with respect to the input factors.

For an example of the output, see "Examples" on page 6-130.

You can add a number of configuration set objects with different `SensitivityAnalysisOptions` to the model object with the `addconfigset` method. Only one configuration set object in the model object can have the `Active` property set to `true` at any given time.

SensitivityAnalysisOptions

Property Summary	`Normalization`	Specify normalization type for sensitivity analysis
	`ParameterInputFactors`	Specify parameter input factors for sensitivity analysis
	`SpeciesInputFactors`	Specify species inputs for sensitivity analysis
	`SpeciesOutputs`	Specify species outputs for sensitivity analysis

Characteristics

Applies to	Object: configuration set
Data type	Object
Data values	`SensitivityAnalysisOptions` properties as summarized in "Property Summary" on page 6-130.
Access	Read-only

Examples

This example shows how to set `SensitivityAnalysisOptions`.

1 Import the radio decay model from SimBiology demos.

```
modelObj  = sbmlimport('radiodecay');
```

2 Retrieve the `configset` object from the `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

3 Add a parameter to the `ParameterInputFactors` property and display. Use the `sbioselect` function to retrieve the parameter object from the model.

```
set(configsetObj.SensitivityAnalysisOptions,'ParameterInputFactors', ...
    sbioselect(modelObj, 'Type', 'parameter', 'Name', 'c'));
```

```
get (configsetObj.SensitivityAnalysisOptions, 'ParameterInputFactors')

SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
  1        c        0.5       1/second
```

4 Add a species to the `SpeciesInputFactors` property and display. Use the `sbioselect` function to retrieve the species object from the model.

```
set(configsetObj.SensitivityAnalysisOptions,'SpeciesInputFactors', ...
    sbioselect(modelObj,'Type', 'species', 'Name', 'z'));
get (configsetObj.SensitivityAnalysisOptions, 'SpeciesInputFactors');
set(configsetObj.SensitivityAnalysisOptions, ...
'SpeciesOutputs', sbioselect(modelObj, 'Type', 'species'));
```

5 Enable SensitivityAnalysis.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true);
get(configsetObj.SolverOptions, 'SensitivityAnalysis')

ans =

    1
```

6 Simulate and return the results to three output variables. See "Description" on page 6-129 for more information.

```
[t,x,names] = sbiosimulate(modelObj);
```

7 Display the names.

```
names

names =

    'x'
    'z'
    'd[x]/d[z]_0'
```

```
'd[z]/d[z]_0'
'd[x]/d[c]'
'd[z]/d[c]'
```

8 Display state values x.

```
x
```

The display follows the column order shown in `names` for the values in `x`. The rows correspond to `t`.

See Also   `addconfigset`, `getconfigset`

SolverOptions

Purpose   Specify model solver options

Description   The SolverOptions property is an object that holds the model solver options in the configset object. Changing the property SolverType changes the options specified in the SolverOptions object.

Properties of SolverOptions are summarized in "Property Summary" on page 6-133.

Property Summary

AbsoluteTolerance	Specify largest allowable absolute error
ErrorTolerance	Specify explicit or implicit tau error tolerance
LogDecimation	Specify recorded simulation output frequency
MaxIterations	Specify nonlinear solver maximum iterations in implicit tau
MaxStep	Specify upper bound on solver step size
RandomState	Set random number generator
RelativeTolerance	Specify allowable error relative to component
SensitivityAnalysis	Enable or disable sensitivity analysis
Type	Display top-level SimBiology object type

Characteristics

Applies to	Object: configset
Data type	Object

6-133

SolverOptions

Data values	Solver options depending on `SolverType`. Default is `SolverOptions` for default `SolverType` (`ode15s`).
Access	Read-only

Examples

This example shows the changes in `SolverOptions` for various `SolverType` settings.

1 Retrieve the `configset` object from the `modelObj`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2 Configure the `SolverType` to `ode45`.

```
set(configsetObj, 'SolverType', 'ode45');
get(configsetObj, 'SolverOptions')


Solver Settings: (ode)

     AbsoluteTolerance:    1.000000e-006
     RelativeTolerance:    1.000000e-003
```

3 Configure the `SolverType` to `ssa`.

```
set(configsetObj, 'SolverType', 'ssa');
get(configsetObj, 'SolverOptions')


Solver Settings: (ssa)

     LogDecimation:          1
     RandomState:           []
```

4 Configure the `SolverType` to `impltau`.

```
set(configsetObj, 'SolverType', 'impltau');
get(configsetObj, 'SolverOptions')


Solver Settings: (impltau)

    ErrorTolerance:      3.000000e-002
    LogDecimation:       1
    AbsoluteTolerance:   1.000000e-002
    RelativeTolerance:   1.000000e-002
    MaxIterations:       15
    RandomState:         []
```

5 Configure the SolverType to expltau.

```
set(configsetObj, 'SolverType', 'expltau');
get(configsetObj, 'SolverOptions')


Solver Settings: (expltau)

    ErrorTolerance:      3.000000e-002
    LogDecimation:       1
    RandomState:         []
```

See Also   addconfigset, getconfigset

SolverType

Purpose   Select solver type for simulation

Description   The SolverType property lets you specify the solver to use for a simulation. For a discussion about solver types, see "Selecting a Solver" in the SimBiology User's Guide documentation.

Changing the solver type changes the options (properties) specified in the SolverOptions property of the configset object. If you change any SolverOptions, these changes are persistent when you switch SolverType. For example, if you set the ErrorTolerance for the expltau solver and then change to impltau when you switch back to expltau, the ErrorTolerance will have the number you assigned.

Characteristics

Applies to	Object: configset
Data type	enum
Data values	'ssa', 'expltau', 'impltau', 'ode45', 'ode23', 'ode113', 'ode15s', 'ode23s', 'ode23t', 'ode23tb', 'sundials'. Default is ode15s.
Access	Read/write

Examples   **1** Retrieve the configset object from the modelObj.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)


Configuration Settings - default (active)
    SolverType:        ode15s
    StopTime:          10.000000

  SolverOptions:
    AbsoluteTolerance:  1.000000e-006
    RelativeTolerance:  1.000000e-003
```

```
        RuntimeOptions:
          StatesToLog:         all

        CompileOptions:
          UnitConversion:      true
          DimensionalAnalysis: true
```

2 Configure the SolverType to ode45.

```
    set(configsetObj, 'SolverType', 'ode45')
    configsetObj

        Configuration Settings - default (active)
          SolverType:          ode45
          StopTime:            10.000000

        SolverOptions:
          AbsoluteTolerance:   1.000000e-006
          RelativeTolerance:   1.000000e-003

        RuntimeOptions:
          StatesToLog:         all

        CompileOptions:
          UnitConversion:      true
          DimensionalAnalysis: true
```

See Also   getconfigset, set

Species

Purpose   Array of species in compartment object

Description   The Species property is a property of the compartment object and
                   indicates all the species in a compartment object. Species is a read-only
                   array of SimBiology species objects.

                   In the model object, Species contains a flat list of all the species
                   that exist within all the compartments in the model. You should
                   always access a species through its compartment rather than the
                   model object. Use the format *compartmentName.speciesName*,
                   for example, nucleus.DNA. Another example of the syntax is
                   modelObj.Compartments(2).Species(1). The Species property in the
                   model object might not be available in a future version of the software.

                   Species are entities that take part in reactions. A species object is added
                   to the Species property when a reaction is added to the model object
                   with the method addreaction. A species object can also be added to the
                   Species property with the method addspecies.

                   If you remove a reaction with the method delete, and a species is no
                   longer being used by any of the remaining reactions, the species object
                   is *not* removed from the Species property. You have to use the delete
                   method to remove species.

                   There are reserved characters that cannot be used in species object
                   names. See Name for more information.

Characteristics

Applies to	Object: compartment
Data type	Array of species objects
Data values	Species object. Default is [] (empty).
Access	Read-only

See Also   addcompartment, addreaction, addspecies, delete

Purpose   Specify species inputs for sensitivity analysis

Description   Use the `SpeciesInputFactors` property to specify the species with respect to which you want to compute the sensitivities of the species states in your model.

`SpeciesInputFactors` is a property of the `SensitivityAnalysisOptions` object. `SenstivityAnalysisOptions` is a property of the configuration set object.

The SimBiology software calculates sensitivities with respect to the initial amounts of the species specified in this property. When you simulate a model with `SensitivityAnalysis` enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species specified in `StatesToLog`. For a description of the output, see the `SensitivityAnalysisOptions` property description.

Characteristics

Applies to	Object: `SensitivityAnalysisOptions`
Data type	Species object or array of species objects
Data values	Species object array. Default is `[]` (empty).
Access	Read/write

Examples   This example shows how to set `SpeciesInputFactors` for sensitivity analysis.

1 Import the radio decay model from the SimBiology demos.

```
modelObj = sbmlimport('radiodecay');
```

2 Retrieve the configuration set object from `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

3 Add a species to the `SpeciesInputFactors` property and display. Use the `sbioselect` function to retrieve the species object from the model.

SpeciesInputFactors

```
set(configsetObj.SensitivityAnalysisOptions,'SpeciesInputFactors', ...
   sbioselect(modelObj, 'Type', 'species', 'Name', 'z'));
get (configsetObj.SensitivityAnalysisOptions, 'SpeciesInputFactors')

SimBiology Species Array

Index:  Compartment:  Name:  InitialAmount: InitialAmountUnits:
  1        unnamed       z          0                  molecule
```

See Also   ParameterInputFactors, sbioselect, SensitivityAnalysis,
SensitivityAnalysisOptions

Purpose   Specify species outputs for sensitivity analysis

Description   The SpeciesOutputs property allows you to specify the species for which you want to compute sensitivities. SpeciesOutputs is a property of the SensitivityAnalysisOptions object. SenstivityAnalysisOptions is a property of the configuration set object.

The SimBiology software calculates sensitivities with respect to the values of the parameters specified in ParameterInputFactors and the initial amounts of the species specified in SpeciesInputFactors. When you simulate a model with SensitivityAnalysis enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species specified in SpeciesOutputs. For a description of the output, see the SensitivityAnalysisOptions property description.

Characteristics

Applies to	Object: SensitivityAnalysisOptions
Data type	Species object or array of species objects
Data values	Species object array. Default is [] (empty).
Access	Read/write

Examples   This example shows how to set SpeciesOutputs for sensitivity analysis.

1 Import the radio decay model from the SimBiology demos.

```
modelObj = sbmlimport('radiodecay');
```

2 Retrieve the configuration set object from modelObj.

```
configsetObj = getconfigset(modelObj);
```

3 Add a species to the SpeciesOutputs property and display. Use the sbioselect function to retrieve the species object from the model.

SpeciesOutputs

```
set(configsetObj.SensitivityAnalysisOptions,'SpeciesOutputs', ...
   sbioselect(modelObj, 'Type', 'species', 'Name', 'z'));
get (configsetObj.SensitivityAnalysisOptions, 'SpeciesOutputs')


SimBiology Species Array

Index:  Compartment:  Name:  InitialAmount: InitialAmountUnits:
  1        unnamed      z          0                 molecule
```

See Also   ParameterInputFactors, sbioselect, SensitivityAnalysis, SensitivityAnalysisOptions, SpeciesInputFactors

Purpose   Cell array of species used in reaction rate equation

Description   The SpeciesVariableNames property shows the species used by the
                    kinetic law object to determine the ReactionRate equation in the
                    reaction object. Use setspecies to assign SpeciesVariableNames.
                    When you assign species to SpeciesVariableNames, SimBiology
                    software maps these species names to SpeciesVariables in the kinetic
                    law object.

                    The ReactionRate property of a reaction object shows the result
                    of a mapping from an abstract kinetic law. The ReactionRate
                    is determined by the kinetic law object Expression property by
                    mapping ParameterVariableNames to ParameterVariables and
                    SpeciesVariableNames to SpeciesVariables.

Characteristics

Applies to	Object: kinetic law
Data type	Cell array of strings
Data values	Cell array of species names
Access	Read/write

Examples   Create a model, add a reaction, and assign the SpeciesVariableNames
                   for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type
   'Henri-Michaelis-Menten'

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

   The reactionObj KineticLaw property is configured to
   kineticlawObj.

3 The `'Henri-Michaelis-Menten'` kinetic law has one species variable
(S) that you should set. To set this variable:

```
setspecies(kineticlawObj,'S', 'a');
```

4 Verify that the species variable is correct.

```
get (kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =

'a'
```

See Also   Expression, ParameterVariables, ParameterVariableNames,
ReactionRate, setparameter, SpeciesVariables

Purpose   Species in abstract kinetic law

Description   This property shows species variables that are used in the `Expression` property of the kinetic law object to determine the `ReactionRate` equation in the reaction object. Use the MATLAB function `set` to assign `SpeciesVariables` to an abstract kinetic law. For more information, see abstract kinetic law.

Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Defined by abstract kinetic law
Access	Read/write in abstract kinetic law. Read-only in kinetic law.

Examples   Create a model, add a reaction, and assign the `SpeciesVariableNames` for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

`reactionObj` KineticLaw property is configured to `kineticlawObj`.

3 View the species variable for `'Henri-Michaelis-Menten'` kinetic law.

```
get (kineticlawObj, 'SpeciesVariables')
```

MATLAB returns:

SpeciesVariables

```
ans =

    'S'
```

See Also   Expression, ParameterVariables, ParameterVariableNames,
ReactionRate, set, setparameter, SpeciesVariableNames

Purpose   Specify species data recorded

Description   The StatesToLog property indicates the species data to log during a simulation. This is the data returned in x during execution of $(t,x) =$ sbiosimulate(modelObj). By default all species are logged.

Characteristics

Applies to	Object: RunTimeOptions
Data type	Object or vector of objects
Data values	Species objects to log. Default is All.
Access	Read/write

Examples   This example shows how to assign species to StatesToLog.

1 Create a model object by importing the file oscillator.xml.

```
modelObj  = sbmlimport('oscillator');
```

2 Retrieve the first and second species in modelObj.

```
speciesObj1 = modelObj.Species(1);
speciesObj2 = modelObj.Species(2);
```

3 Retrieve the configsetObj of modelObj.

```
configsetObj = getconfigset(modelObj);
```

4 Set the StatesToLog to record three species: two using the retrieved species objects and one using indexing and view the species in StatesToLog.

```
set (configsetObj.RuntimeOptions, 'StatesToLog', ...
    [speciesObj1, speciesObj2, modelObj.Species(3)]);
get(configsetObj.RuntimeOptions, 'StatesToLog')
```

Stoichiometry

Purpose   Species coefficients in reaction

Description   The `Stoichiometry` property specifies the species coefficients in a reaction. Enter an array of `doubles` indicating the stoichiometry of reactants (negative value) and products (positive value). Example: [-1 -1 2].

The `double` specified cannot be `0`. The reactants of the reaction are defined with a negative number. The products of the reaction are defined with a positive number. For example, the reaction 3 H + A-> 2 C + F has the `Stoichiometry` value of [-3 -1 2 1].

When this property is configured, the `Reaction` property updates accordingly. In the above example, if the `Stoichiometry` value was set to [-2 -1 2 3], the reaction is updated to 2H + A -> 2C + 3F.

The length of the `Stoichiometry` array is the sum of the `Reactants` array and the `Products` array. To remove a product or reactant from a reaction, use the `rmproduct` or `rmreactant` function. Add a product or reactant and set stoichiometry with methods `addproduct` and `addreactant`.

ODE solvers support `double` stoichiometry values such as `0.5`. Stochastic solvers and dimensional analysis currently support only integers in `Stoichiometry`, therefore you must balance the reaction equation and specify integer values for these two cases.

`A -> null` has a stoichiometry value of [`-1`]. `null -> B` has a stoichiometry value of [`1`].

Characteristics

Applies to	Object: reaction
Data type	Double array
Data values	1-by-n double, where n is length (products) + length (reactants). Default is [ ] (empty).
Access	Read/write

Examples

1 Create a reaction object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, '2 a + 3 b -> d + 2 c');
```

2 Verify the Reaction and Stoichiometry properties for reactionObj.

```
get(reactionObj,'Stoichiometry')
```

MATLAB returns:

```
ans =

-2    -3    1    2
```

3 Set stoichiometry to [-1 -2 2 2].

```
set (reactionObj, 'Stoichiometry', [-1 -2 2 2]);
get (reactionObj, 'Stoichiometry')
```

MATLAB returns:

```
ans =

 -1    -2    2    2
```

4 Note with get that the Reaction property updates automatically.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =

a + 2 b -> 2 d + 2 c
```

See Also   addproduct, addreactant, addreaction, Reaction, rmproduct, rmreactant

StopTime

Purpose	Set stop time for simulation
Description	The StopTime property sets the stop time for a simulation. The type of StopTime is specified in the property StopTimeType.

Characteristics

Applies to	Object: configset
Data type	double
Data values	Enter a positive number. Default is 10.
Access	Read/write

Examples

1 Retrieve the configset object from modelObj.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
```

2 Configure the StopTime to 20.

```
set(configsetObj, 'StopTime', 20)
get(configsetObj, 'StopTime')

ans =

    20
```

See Also   StopTimeType, TimeUnits

Purpose   Specify type of stop time for simulation

Description   The StopTimeType property sets the type of stop time for a simulation. The stop time is specified in the StopTime property of the configset object. Valid types are approxWallTime, numberOfLogs, and simulationTime. The default is simulationTime.

- simulationTime — Specify the stop time for the simulation. The solver determines and sets the time steps and the simulation stops when it reaches the specified StopTime.

- approxWallTime — Specify the approximate stop time according to the clock. For example,10s of approxWallTime is approximately 10s of real time.

- numberOfLogs — Specify the total number of simulation steps to be recorded during the simulation. For example if you want to log three simulation steps, the numberOfLogs is 3. The simulation will stop after the specified numberOfLogs.

You can change the StopTimeType setting with the set function.

Characteristics

Applies to	Object: configset
Data type	enum
Data values	approxWallTime, numberOfLogs, and simulationTime
Access	Read/write

Examples   **1** Retrieve the configset object from modelObj.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2 Configure the StopTimeType to approxWallTime.

StopTimeType

```
set(configsetObj, 'StopTimeType', 'approxWallTime');
get(configsetObj, 'StopTimeType')

ans =

approxWallTime
```

See Also   set, StatesToLog, StopTime, TimeUnits

Tag

Purpose Specify label for SimBiology object

Description The Tag property specifies a label associated with a SimBiology object. Use this property to group objects and then use sbioselect to retrieve. For example, use the Tag property in reaction objects to group synthesis or degradation reactions. You can then retrieve all synthesis reactions using sbioselect. Similarly, for species objects you can enter and store classification information, for example, membrane protein, transcription factor, enzyme classifications, or whether a species is an independent variable. You can also enter the full form of the name of the species. This is useful when viewing the model in the Block Diagram Explorer. For example, the species object Name could be G6P for convenience, but in the Tag you should enter the full name, Glucose-6 phosphate. The graphical representation of the model in the Block Diagram Explorer (available in the SimBiology desktop: simbiology) can be sorted by the Tag field, and this feature provides a method to view the full name.

Characteristics

Applies to	Objects: abstract kinetic law, kinetic law, model, parameter, reaction, rule, and species
Data type	char string
Data values	Any char string
Access	Read/write

Examples **1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a reaction object and set the Tag property to 'Synthesis Reaction'.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
set (reactionObj, 'Tag', 'Synthesis Reaction')
```

3 Verify the Tag assignment.

Tag

```
get (reactionObj, 'Tag');
```

MATLAB returns:

```
ans =

    'Synthesis Reaction'
```

See Also addkineticlaw, addparameter, addreaction, addrule, addspecies, sbioabstractkineticlaw, sbiomodel, sbioroot

Purpose   Event trigger

Description   A `Trigger` is a condition that must become true for an event to be
executed. You can a combination of relational and logical operators to
build a trigger expression. `Trigger` can be a string, an expression, or a
function handle that when evaluated returns a value of `true` or `false`.
Triggers can access species, parameters, and compartments.

A trigger can contain the keyword `'time'`, to define an event that
occurs at a specific time during the simulation.

For more information about how the SimBiology software handles
events, see "How Events Are Evaluated" in the SimBiology User's Guide
documentation. For examples of event functions, see "Specifying Event
Triggers" in the SimBiology User's Guide documentation.

Characteristics

Applies to	Object: event
SimBiology type	String, function handle
SimBiology values	Specify MATLAB expression as string. Default is `''` (None).
Access	Read/write

Examples   **1** Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator');
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

2 Set the `Trigger` property of the event object.

```
set(eventObj, 'Trigger', '(time >=5) && (speciesA<1000)');
```

3 Get the `Trigger` property.

```
get(eventObj, 'Trigger')
```

Trigger

See Also   Event object, EventFcns

Purpose   Show simulation time steps

Description   The Time property shows the time points in a simulation.

Characteristics

Applies to	Object: SimData
Data type	double
Data values	Vector of doubles
Access	Read-only

See Also   StopTime, StopTimeType

TimeUnits

Purpose   Show stop time units for simulation

Description   The TimeUnits property shows units for the stop time for a simulation.
The type of StopTime is specified in the property StopTimeType. Unit is
in seconds.

Characteristics

Applies to	Object: configset
Data type	string
Data values	Default value is second.
Access	Read-only

See Also   StopTime, StopTimeType

Purpose   Display top-level SimBiology object type

Description   The Type property indicates a SimBiology object type. When you create a SimBiology object, the value of Type is automatically defined.

For example, when a Species object is created, the value of the Type property is automatically defined as `'species'`.

Characteristics

Applies to	Objects: abstract kinetic law, configuration set, CompileOptions, kinetic law, model, parameter, reaction, root, rule, species, RuntimeOptions, and SolverOptions
Data type	char string
Data values	abstract_kinetic_law, configset, compileoptions, kineticlaw, parameter, reaction, root, rule, runtimeoptions, sbiomodel, species, and solveroptions
Access	Read-only

See Also   sbiomodel, sbioparameter, sbioreaction, sbioroot, sbiorule, sbiospecies

UnitConversion

Purpose   Perform unit conversion

Description   The `UnitConversion` property specifies whether to perform unit conversion for the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object.

When `UnitConversion` is set to `true`, the SimBiology software converts the matching physical quantities to one consistent unit system in order to resolve them. This conversion is in preparation for correct simulation, but species amounts are returned in the user-specified units.

For example, consider a reaction `a + b  > c`. Using mass action kinetics the reaction rate is defined as `a*b*k` where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are 0.01M and 0.005M respectively, then units of `k` are `1/(M*second)`. If you specify `k` with another equivalent unit definition, for example, `1/((molecules/liter)*second)`, `UnitConversion` occurs after `DimensionalAnalysis`.

Unit conversion requires dimensional analysis. If `DimensionalAnalysis` is off, and you turn `UnitConversion` on, then `DimensionalAnalysis` is turned on automatically. If `UnitConversion` is on and you turn off `DimensionalAnalysis`, then `UnitConversion` is turned off automatically.

If `UnitConversion` fails, then you see an error when you simulate (`sbiosimulate`).

If `UnitConversion` is set to `false`, the simulation uses the given object values.

Unit conversion involving temperature supports `Celsius` as the temperature unit. Avoid using mixed temperature units as you might get an error.

Characteristics

Applies to	Object: CompileOptions (in configset object)
Data type	boolean
Data values	true or false. Default value is false.
Access	Read/write

Examples This example shows how to retrieve and set unitconversion from the default true to false in the default configuration set in a model object.

1 Import a model.

```
modelObj = sbmlimport('oscillator')


SimBiology Model - Oscillator

   Model Components:
     Models:          0
     Parameters:      0
     Reactions:       42
     Rules:           0
     Species:         23
```

2 Retrieve the configset object of the model object.

```
configsetObj = getconfigset(modelObj)

 Configuration Settings - default (active)
    SolverType:         ode15s
    StopTime:           10.000000

   SolverOptions:
     AbsoluteTolerance:   1.000000e-006
     RelativeTolerance:   1.000000e-003
```

UnitConversion

```
    RuntimeOptions:
      StatesToLog:          all

    CompileOptions:
      UnitConversion:       false
      DimensionalAnalysis:  true
```

3 Retrieve the `CompileOptions` object.

```
optionsObj = get(configsetObj,'CompileOptions')
```

```
Compile Settings:

    UnitConversion:       false
    DimensionalAnalysis:  true
```

4 Assign a value of false to `UnitConversion`.

```
set(optionsObj,'UnitConversion', true)
```

See Also   `get`, `getconfigset`, `sbiosimulate`, `set`

Purpose   Specify data to associate with object

Description   Property to specify data that you want to associate with a SimBiology object. The object does not use this data directly, but you can access it using the function `get` or dot notation.

Characteristics

Applies to	Objects: abstract kinetic law, configuration set, compartment, data, event, kinetic law, model, parameter, reaction, rule, species, or unit
Data type	Any
Data values	Any. Default is empty.
Access	Read/write

See Also   sbioabstractkineticlaw, sbiomodel, sbioparameter, sbioreaction, sbioroot, sbiorule, sbiospecies, sbiounit, sbiounitprefix

UserDefinedKineticLaws

Purpose   Contain user-defined kinetic laws

> **Note** UserDefinedKineticLaws has been removed and produces an
> error. Use UserDefinedLibrary instead.

Description   The UserDefinedKineticLaws property is a SimBiology root object
property showing all user-defined abstract kinetic laws. Use the
command sbiowhos -userdefined -kineticlaw to see the list of
user-defined kinetic laws. You can use user-defined kinetic laws when
you use the command addkineticlaw to create a kinetic law object for
a reaction object. Refer to the kinetic law by name when you create
the kinetic law object, for example:

```
kineticlawObj = addkineticlaw(reactionObj, 'my_kinetic_law');
```

You can add, modify, or delete UserDefinedKineticLaws. Create an
abstract kinetic law with the command sbioabstractkineticlaw
and add it to the user-defined kinetic law library with the
command sbioaddtolibrary. sbioaddtolibrary also updates the
UserDefinedKineticLaws property of the root object.

See "Abstract Kinetic Law" on page 6-56 for a definition and more
information.

Characteristics

Applies to	Object: root
Data type	char string
Data values	Valid kinetic laws
Access	Read/write

See Also   AbstractKineticLaw object, sbioaddtolibrary,
UserDefinedLibrary

Purpose   Library of user-defined components

Description   UserDefinedLibrary is a SimBiology root object property containing all user-defined components of unit, unit prefixes, and abstract kinetic laws that you define. You can add, modify, or delete components in the user-defined library. The UserDefinedLibrary property is an object that contains the following properties:

- Units — Contains any user-defined units. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined units either by using the command sbiowhos -userdefined -unit, or by accessing the root object.

- UnitPrefixes — Contains any user-defined unit prefixes. You can specify unit prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined unit prefixes either by using the command sbiowhos -userdefined -unitprefix, or by accessing the root object.

- KineticLaws — Contains any user-defined unit abstract kinetic laws. Use the command sbiowhos -userdefined -kineticlaw to see the list of user-defined kinetic laws. You can use user-defined kinetic laws when you use the command addkineticlaw to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');.

  See "Abstract Kinetic Law" on page 6-56 for a definition and more information.

Characteristics

Applies to	Object: root
Data type	object

UserDefinedLibrary

Data values	Unit, unit prefix, and abstract kinetic law objects
Access	Read-only

Characteristics for `UserDefinedLibrary` properties:

- `Units`

Applies to	`UserDefinedLibrary` property
Data type	Unit objects
Data values	Units
Access	Read/write

- `UnitPrefixes`

Applies to	`BuiltInLibrary` property
Data type	Unit prefix objects
Data values	Unit prefixes
Access	Read/write

- `KineticLaws`

Applies to	`BuiltInLibrary` property
Data type	Abstract kinetic law object
Data values	Kinetic laws
Access	Read/write

Examples

Example 1

This example uses the command `sbiowhos` to show the current list of user-defined components.

```
sbiowhos -userdefined -kineticlaw
```

```
sbiowhos -userdefined -unit
sbiowhos -userdefined -unitprefix
```

Example 2

This example shows the current list of user-defined components by accessing the root object.

```
rootObj = sbioroot;
get(rootObj.UserDefinedLibrary, 'KineticLaws')
get(rootObj.UserDefinedLibrary, 'Units')
get(rootObj.UserDefinedLibrary, 'UnitPrefixes')
```

See Also   BuiltInLibrary, sbioaddtolibrary, sbioremovefromlibrary, sbioroot, sbiounit, sbiounitprefix

UserDefinedUnitPrefixes

Purpose   Contain user-defined unit prefixes

Note UserDefinedUnitPrefixes has been removed and produces an error. Use UserDefinedLibrary instead.

Description   The UserDefinedUnitPrefixes property is a SimBiology root object property showing all user-defined unit prefixes. You can specify units with prefixes for species amounts and parameter values to do dimensional analysis and unit conversion during simulation. The valid units and unit prefixes are either built in or user defined. Use the command sbiowhos -userdefined -unit to see the list of user-defined units.

You can add, modify, or delete UserDefinedUnitPrefixes. You can define a unit prefix with the command sbioregisterunitprefix, which enables you to create the unit and add it to the user-defined unit prefixes library, and also add it to the UserDefinedUnitPrefixes property of the root object.

Characteristics

Applies to	Object: root
Data type	char string
Data values	Valid unit prefixes
Access	Read/write

See Also   sbioaddtolibrary, UserDefinedLibrary, UnitPrefix object

Purpose   Contain user-defined units

Note `UserDefinedUnits` has been removed and produces an error. Use `UserDefinedLibrary` instead.

Description   The `UserDefinedUnits` property is a SimBiology root object property showing all user-defined units. You can specify units for species amounts and parameter values to do dimensional analysis and unit conversion during simulation. The valid units are either built in or user defined. Use the command `sbiowhos -userdefined -unit` to see the list of user-defined units.

You can add, modify, or delete `UserDefinedUnits`. You can define a unit with the command `sbioregisterunit`, which enables you to create the unit and add it to the user-defined units library, and also add it to the `UserDefinedUnits` property of the root object.

Characteristics

Applies to	Object: root
Data type	`char string`
Data values	Valid units
Access	Read/write

See Also   `sbioaddtolibrary`, `UserDefinedLibrary`, `Unit object`

Value

Purpose	Assign value to parameter object

Description   The Value property is the value of the parameter object. The parameter object defines an assignment that can be used by the model object and/or the kinetic law object. Create parameters and assign Value using the method addparameter.

Characteristics

Applies to	Object: parameter
Data type	double
Data values	Any double. Default value is 1.0.
Access	Read/write

Examples   Assign a parameter with a value to the model object.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a parameter to the model object (modelObj) with Value 0.5.

```
parameterObj1 = addparameter (modelObj, 'K1', 0.5)
```

MATLAB returns:

```
SimBiology Parameter Array

Index:   Name:   Value:   ValueUnits:
  1       K1      0.5
```

See Also   addparameter, sbioparameter

Purpose   Parameter value units

Description   The `ValueUnits` property indicates the unit definition of the parameter
                     object `Value` property. `ValueUnits` can be one of the built-in units.
                     To get a list of the built-in units, use the `sbioshowunits` function. If
                     `ValueUnits` changes from one unit definition to another, the `Value` does
                     not automatically convert to the new units. The `sbioconvertunits`
                     function does this conversion.

                     You can add a parameter object to a model object or a kinetic law object.

Characteristics

Applies to	Object: parameter
Data type	char string
Data values	Unit from units library. Default is `''` (empty).
Access	Read/write

Examples   Assign a parameter with a value to the model object.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
```

2 Add a parameter with `Value` `0.5`, and assign it to the model object
  (`modelObj`).

```
parameterObj1 = addparameter(modelObj, 'K1', 0.5, 'ValueUnits', '1/second')
```

MATLAB returns:

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1        K1       0.5        1/second
```

See Also   addparameter, sbioconvertunits, sbioparameter, sbioshowunits

ValueUnits

Index