

SimBiology

For Use with **MATLAB®**

- Computation
- Visualization
- Programming

Reference

Version 2



How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
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

© COPYRIGHT 2005–2006 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks, and SimBiology, SimEvents, and SimHydraulics are trademarks of The MathWorks, Inc.

Other product or brand names are 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

September 2005 Online only
March 2006 Online only
May 2006 Online only
September 2006 Online only

New for Version 1.0 (Release 14SP3+)
Updated for Version 1.0.1 (Release 2006a)
Updated for Version 2.0 (Release 2006a+)
Updated for Version 2.0.1 (Release 2006b)

Functions — By Category

1

Tools	1-2
Projects	1-3
SBML Models	1-4
Object Constructors	1-5
Units	1-6

Functions — Alphabetical List

2

Methods — By Category

3

Objects	3-2
Abstract Kinetic Laws	3-2
Configuration Sets	3-3
Kinetic Laws	3-4
Models	3-5

Parameters	3-6
Reactions	3-7
Root	3-8
Rules	3-9
Species	3-10
Using Object Methods	3-11
Constructing (Creating) Objects	3-11
Using Object Methods	3-11
Help for Objects, Methods and Properties	3-12

Methods — Alphabetical List

4

Properties — By Category

5

Abstract Kinetic Law	5-2
Configuration Sets	5-3
Kinetic Laws	5-4
Models	5-5
Parameters	5-6
Reactions	5-7

Root	5-8
Rules	5-9
Species	5-10
Using Object Properties	5-11
Entering Property Values	5-11
Retrieving Property Values	5-11
Help for Objects, Methods, and Properties	5-12

Properties — Alphabetical List

6

Index

Functions — By Category

Tools (p. 1-2)

Modeling, simulation, and analysis tools

Projects (p. 1-3)

Save and open projects in MATLAB®

SBML Models (p. 1-4)

Export and Import SBML models

Object Constructors (p. 1-5)

Create SimBiology™ objects

Units (p. 1-6)

Perform Unit conversion and create user-defined units

Tools

<code>sbioconsmoiety</code>	Find conserved moieties in SimBiology model
<code>sbiodesktop</code>	Open SimBiology modeling and simulation GUI
<code>sbioensembleplot</code>	Show results of ensemble run using 2-D or 3-D plots
<code>sbioensemblerrun</code>	Multiple stochastic ensemble runs of SimBiology model
<code>sbioensemblestats</code>	Get statistics from ensemble data created using <code>sbioensemblerrun</code>
<code>sbiogetmodel</code>	Get model object that generated simulation data
<code>sbiogetnamedstate</code>	Get state and time data from simulation results
<code>sbiogetsensmatrix</code>	3-D sensitivity matrix from simulation results
<code>sbiohelp</code>	Help for SimBiology functions
<code>sbiolasterror</code>	SimBiology last error message
<code>sbiolastwarning</code>	SimBiology last warning message
<code>sbioparamestim</code>	Perform parameter estimation
<code>sbioreset</code>	Delete all model and simulation objects
<code>sbioselect</code>	Search for objects with specified constraints
<code>sbiosimulate</code>	Simulate model object

Projects

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

SBML Models

sbmlexport

Export SimBiology model to SBML file

sbmlimport

Import SBML-formatted file

Object Constructors

<code>sbioabstractkineticlaw</code>	Construct abstract kinetic law object
<code>sbiomodel</code>	Construct model object
<code>sbioparameter</code>	Construct parameter object
<code>sbioreaction</code>	Construct reaction object
<code>sbioroot</code>	Return SimBiology root object
<code>sbiorule</code>	Construct rule object
<code>sbiospecies</code>	Construct species object

Units

<code>sbioconvertunits</code>	Convert unit and unit value to new unit
<code>sbioregisterunit</code>	Create user-defined unit
<code>sbioregisterunitprefix</code>	Create user-defined unit prefix
<code>sbioshowunitprefixes</code>	Information about registered unit prefixes
<code>sbioshowunits</code>	Information about registered units
<code>sbiounitcalculator</code>	Convert value between units
<code>sbiounregisterunit</code>	Remove user-defined unit from root and library
<code>sbiounregisterunitprefix</code>	Remove user-defined unit prefix from root and library

Functions — Alphabetical List

sbioabstractkineticlaw

Purpose Construct abstract kinetic law object

Syntax

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

Arguments

<i>Name</i>	Enter a name for the abstract kinetic law. Name must be unique in the user-defined kinetic law library. Name is referenced by <i>kineticlawObj</i> .
<i>Expression</i>	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 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 name, 'Name' and with 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

<code>copyobj</code> (any object)	Copy SimBiology object and its children
<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object

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

Example

- 1 Create an abstract kinetic law.

```
abstkineticlawObj = sbioabstractkineticlaw('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);
```

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

```
sbiowhos -kineticlaw -userdefined
```

```
Abstract Kinetic Law Object 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, 'mylaw1');
```

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 abstract kinetic law to user-defined library

Syntax `sbioaddtolibrary (abstkineticlawObj)`

Description `sbioaddtolibrary (abstkineticlawObj)` adds the abstract kinetic law object (`abstkineticlawObj`) to the user-defined library. `abstkineticlawObj` is added to the SimBiology root object's `UserDefinedKineticLaws` list. `abstkineticlawObj` is available automatically in future MATLAB sessions. You can 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 the abstract kinetic law objects in the built-in and user-defined libraries, use the commands `get(sbioroot, 'BuiltInKineticLaws')`, `get(sbioroot, 'UserDefinedKineticLaws')`.

To remove an abstract kinetic law from the user-defined library, use the method `sbioremovefromlibrary`. You will not be able to remove an abstract kinetic law object being used by a kinetic law object.

Example 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('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);
```

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

```
sbiowhos -kineticlaw -userdefined
```

```
Abstract Kinetic Law Object 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, 'mylaw1');
```

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

See Also

`addkineticlaw`, `sbioabstractkineticlaw`, `sbioregisterunit`,
`sbioregisterunitprefix`, `sbioroot`

sbioconsmoiety

Purpose Find conserved moieties in SimBiology model

Syntax

```
[G, Sp]= sbioconsmoiety(mode1Obj)
[G, Sp] = sbioconsmoiety(mode1Obj, alg)
H = sbioconsmoiety(mode1Obj, alg, 'p')
H = sbioconsmoiety(mode1Obj, alg, 'p', FormatArg)
[SI,SD,LO,NR,ND] = sbioconsmoiety(mode1Obj, 'link')
```

Arguments

<i>G</i>	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 <i>G</i> specifies a linear combination of species whose rate of change over time is zero.
<i>Sp</i>	Cell array of species names that labels the columns of <i>G</i> .
<i>mode1Obj</i>	Model object to be evaluated for conserved moieties.
<i>alg</i>	Specify algorithm to use during evaluation of conserved moieties. Valid values are 'qr', 'rreduce', or 'semipos'.
<i>H</i>	Cell array of strings containing the conserved moieties.
<i>p</i>	Prints the output to a cell array of strings.
<i>FormatArg</i>	Specifies formatting for the output <i>H</i> . <i>FormatArg</i> should either be a C-style format string, or a positive integer specifying the maximum number of digits of precision used.

<i>interpolation</i>	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.
<i>SI</i>	Cell array containing the names of independent species in the model.
<i>SD</i>	Cell array containing the names of dependent species in the model.
<i>LO</i>	Link matrix relating SI and SD. The link matrix <i>LO</i> 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.
<i>NR</i>	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 <i>modelObj</i> .
<i>ND</i>	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 <i>modelObj</i> .

Description

$[G, Sp] = \text{sbioconsmoiety}(\text{modelObj})$ 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,LO,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

Shows conserved moieties in a cycle.

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

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

- 2 Look for conserved moieties.

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

```
g =
```

```
    1    1    1
```

```
sp =
```

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

Example 2

Explore semipositive conservation relations in the oscillator model.

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

```
ans =
```

```
'pol + pol_0pA + pol_0pB + pol_0pC'
'0pB + pol_0pB + pA_0pB1 + pA_0pB_pA + pA_0pB2'
'0pA + pol_0pA + pC_0pA1 + pC_0pA2 + pC_0pA_pC'
'0pC + pol_0pC + pB_0pC1 + pB_0pC2 + pB_0pC_pB'
```

sbioconsmoiety

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 will configure the `*Units` property to *Unit* and update the corresponding value property. For example `sbioconvertunits` 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 SimBiology species and parameter 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.

Example 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

```
speciesObj = sbiospecies ('glucose', 23, 'InitialAmountUnits', 'molecule')
```

Species Object Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	glucose	23	molecule

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

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

sbioconvertunits

3 Verify the conversion of units and InitialAmount value.

Units are converted from molecule to mole.

```
get (speciesObj2, 'InitialAmountUnits')
```

```
ans =
```

```
mole
```

InitialAmount value is changed.

```
get (speciesObj2, 'InitialAmount')
```

```
ans =
```

```
3.8192e-023
```

See Also

sbioshowunits

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.

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')`, `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.

Example

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);
```

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

```
sbiowhos -kineticlaw -userdefined
```

sbiocopylibrary

Abstract Kinetic Law Object 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

Purpose Open SimBiology modeling and simulation GUI

Syntax `sbiodesktop`
`sbiodesktop(mode1Obj)`

Arguments

mode1Obj 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 GUI. The SimBiology GUI lets you do the following:

- Build a SimBiology model using reaction pathways and enter 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.
- View results from the simulation.
- Create and/or modify user-defined units and unit prefixes.
- Create and/or modify user-defined abstract kinetic law objects.

`sbiodesktop(mode1Obj)` opens the SimBiology GUI with a top-level SimBiology model object (*mode1Obj*). A top-level SimBiology model object has its property `Parent` set to the SimBiology root object.

In contrast, a SimBiology model object that has its property `Parent` set to another SimBiology model is a submodel and is not stored directly by the SimBiology root object. You cannot load a submodel without loading the parent model.

Example

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

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

See Also

sbioroot

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

Syntax

```
sbioensembleplot(simdata, species_array)
sbioensembleplot(simdata, species_array, time)
FH = sbioensembleplot(simdata, species_array)
FH = sbioensembleplot(simdata, species_array, time)
```

Arguments

<i>simdata</i>	<i>simdata</i> is a cell array of time series objects, where each time series object holds data for a separate simulation run. The <i>simdata</i> object could change in future versions. Use the <i>simdata</i> object to call only <code>sbioensemblestats</code> and <code>sbioensembleplot</code> .
<i>species_array</i>	Either an array of species objects or a cell array of names of species in the model object that was used to create <i>simdata</i> . If there are species with duplicate names in different submodels, either use fully qualified names or use an array of species objects to identify species correctly. For example, a fully qualified name for a species named <code>sp1</code> that is in a submodel named <code>sub2</code> , that is inside a submodel <code>sub1</code> , that is parented to the top-level model <code>top1</code> , is <code>top1.sub1.sub2.sp1</code> .
<i>time</i>	Either a single scalar value or a 2-element numeric vector. When it is a single scalar value, it is expected that the data in <i>simdata</i> is interpolated to a common time vector and that the time value in <i>time</i> is a member of the common time vector within a numerical tolerance of $1.0e-4$. If <i>time</i> is not found in the common time vector, you see an error. When <i>time</i> is a numeric vector, it must have two elements: <code>[time_lower, time_upper]</code> , where <i>time_lower</i> and <i>time_upper</i> are real numbers such that <code>time_lower <= time_upper</code> . The distribution plot are created for data points whose time value lies in the closed interval <code>[time_lower, time_upper]</code> . You can

sbioensembleplot

specify the 2-element numeric vector version of *time* regardless of whether *simdata* is interpolated to a common time vector or not.

FH Array of handles to figure windows.

Description

`sbioensembleplot(simdata, species_array)` shows a 3-D shaded plot of time-varying distribution of one or more species specified in *species_array*.

`sbioensembleplot(simdata, species_array, time)` shows a 2-D distribution plot of the species specified in *species_array* at time *time*.

FH = `sbioensembleplot(simdata, species_array)` or *FH* = `sbioensembleplot(simdata, species_array, time)` returns an array of handles to the figure windows in *FH*.

Examples

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

- 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 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.

```
simdata = sbioensemblerrun(m1, 20);
```

- 4 Create a 2-D distribution plot of the second species at a time value in the range [1.0 1.05]. Note that since `simdata` was not interpolated, you should specify a time range here.

```
sbioensembleplot(simdata, m1.species(2), [1.0 1.05] );
```

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

```
sbioensembleplot(simdata, {'x','z'});
```

See Also

`sbioensemblerrun`, `sbioensemblestats`, `sbioemodel`

sbioensemblerun

Purpose Multiple stochastic ensemble runs of SimBiology model

Syntax `simdata = sbioensemblerun(mode1Obj, numruns, interpolation)`

Arguments

<i>simdata</i>	<i>simdata</i> is a cell array of time series objects, where each time series object holds data for a separate simulation run. The <i>simdata</i> object could change in future versions. Use the <i>simdata</i> object to call only <code>sbioensemblestats</code> and <code>sbioensembleplot</code> .
<i>mode1Obj</i>	Model object to be simulated.
<i>numruns</i>	Integer scalar representing the number of stochastic runs to make.
<i>interpolation</i>	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.

Description

`simdata = sbioensemblerun(mode1Obj, numruns, interpolation)` performs a stochastic ensemble run of the SimBiology model object (*mode1Obj*). *mode1Obj* must be a top-level SimBiology model. A top-level SimBiology model object has its `Parent` property set to the SimBiology root object. The active `configset` on *mode1Obj* is used for performing the multiple 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 of the active `configset` of *mode1Obj* is set to any of the deterministic (ODE) solvers.

Examples

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

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

```
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);
```

- 3 Perform an ensemble of 20 runs with linear interpolation to get a consistent time vector for all time series objects.

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

- 4 Create a 2D distribution plot of the second species at a time value in the range `[1.0 1.05]`.

```
sbioensembleplot(simdata, m1.species(2), [1.0 1.05] );
```

See Also

SimBiology functions `addconfigset`, `getconfigset`, `sbioensemblestats`, `sbioensembleplot`, `setactiveconfigset`

sbioensemblestats

Purpose Get statistics from ensemble data created using sbioensemblerrun

Syntax

```
[t,m] = sbioensemblestats(simdata)
[t,m,v] = sbioensemblestats(simdata)
[t,m,v,n] = sbioensemblestats(simdata)
[t,m] = sbioensemblestats(simdata, species_array)
[t,m,v] = sbioensemblestats(simdata species_array)
[t,m,v,n] = sbioensemblestats(simdata species_array)
[t,m] = sbioensemblestats(simdata, species_array, interpolation)
[t,m,v]= sbioensemblestats(simdata,species_array,
    interpolation)
[t,m,v,n]= sbioensemblestats(simdata,species_array,
    interpolation)
```

Arguments

<i>t</i>	Vector of doubles that holds the common time vector after interpolation.
<i>m</i>	Matrix of mean values from the ensemble data. The number of rows in <i>m</i> is the length of the common time vector <i>t</i> after interpolation and the number of columns is equal to the number of species. The species order corresponding to the columns of <i>m</i> can be obtained from any of the time series objects in <i>simdata</i> using sbiogetnamedstate.
<i>simdata</i>	<i>simdata</i> is a cell array of time series objects, where each time series object holds data for a separate simulation run. The <i>simdata</i> object could change in future versions. Use the <i>simdata</i> object to call only sbioensemblestats and sbioensembleplot.
<i>v</i>	Matrix of variance obtained from the ensemble data. <i>v</i> has the same dimensions as <i>m</i>

<i>n</i>	Cell array of strings that holds names of species whose mean and variance are returned in <i>m</i> and <i>v</i> , respectively. The number of elements in <i>n</i> is the same as the number of columns of <i>m</i> and <i>v</i> . The order of species names in <i>n</i> corresponds to the species order of columns of <i>m</i> and <i>v</i> .
<i>species_array</i>	Either an array of species objects or a cell array of names of species in the model object that was used to create <i>simdata</i> . If there are species with duplicate names in different submodels, either use fully qualified names or use an array of species objects to identify species correctly. For example, a fully qualified name for a species named sp1 that is in a submodel named sub2, that is inside a submodel sub1, that is parented to the top-level model top1, is top1.sub1.sub2.sp1.
<i>interpolation</i>	String variable denoting the interpolation method to be used if data is to 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.

Description

`[t,m] = sbioensemblestats(simdata)` gets mean information *m* as a function of time for all the species in the model used to generate ensemble data *simdata* by running `sbioenssemblerun`. The species order corresponding to the columns of *m* can be obtained from any of the time series objects in *simdata* by using `sbiogetnamedstate`.

`[t,m,v] = sbioensemblestats(simdata)` gets mean *m* and variance *v* as a function of time for all the species in the model used to generate ensemble data *simdata* by running `sbioenssemblerun`.

`[t,m,v,n] = sbioensemblestats(simdata)` gets mean m , variance v , and names n of species corresponding to the columns of m and v .

`[t,m] = sbioensemblestats(simdata, species_array)`, or `[t,m,v] = sbioensemblestats(simdata species_array)`, or `[t,m,v,n] = sbioensemblestats(simdata species_array)`, gets mean m , variance v , and names n only for the species specified in `species_array`.

`[t,m] = sbioensemblestats(simdata, species_array, interpolation)`, or `[t,m,v] = sbioensemblestats(simdata,species_array,interpolation)`, or `[t,m,v,n] = sbioensemblestats(simdata,species_array, interpolation)` gets mean m , variance v , and names n only for the species specified in `species_array` using the interpolation method `interpolation`.

Examples

This example shows you how to get ensemble run statistics with the default interpolation method.

- 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.

```
simdata = sbioenssemblerun(m1, 20);
```

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

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


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

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

See Also

`sbioensemblerrun`, `sbioensembleplot`, `sbiomodel`, `sbiogetnamedstate`

sbiogetmodel

Purpose Get model object that generated simulation data

Syntax `modelObj = sbiogetmodel(timeseriesObj)`

Arguments

<code>timeseriesObj</code>	Time series object returned by the function <code>sbiosimulate</code> .
<code>modelObj</code>	Model object associated with the time series object.

Description

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

If the SimBiology model used to generate the time series object (`timeseriesObj`) is not currently loaded, `modelObj` is empty.

Example

Retrieve the model object that generated the time series object.

- 1 Create a model object, simulate, and then return the results as a time series object.

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

- 2 Get the model that generated the simulation results.

```
modelObj2 = sbiogetmodel(timeseriesObj)
```

```
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`

sbiogetnamedstate

Purpose Get state and time data from simulation results

Syntax

```
[t,x]= sbiogetnamedstate(timeseriesObj)
[t,x]= sbiogetnamedstate(timeseriesObj, 'SpeciesName')
[t,x,SpeciesName]= sbiogetnamedstate(...)
```

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

- `t` is a n -by-1 vector, where n is the number of times the reactions fired. `t` defines the time steps of the firing of the reactions.
- `x` is a n -by- m matrix where n is the number of times the reactions fired and m is the number of SimBiology species in the SimBiology model. Each column of `x` defines the variation in the quantity of a species over time.

`[t,x]= sbiogetnamedstate(timeseriesObj, 'SpeciesName')` returns the state data associated with the SimBiology species of name `SpeciesName` from the simulation run results, (`timeseriesObj`) and returns it to `x`. `SpeciesName` can be a cell array of SimBiology species names. If a species with name, `SpeciesName`, does not exist, SimBiology returns a warning. `SpeciesName` must be the fully qualified species name, for example `ModelName.SpeciesName`. If the species is in a submodel, `SpeciesName` must be `ModelName.SubmodelName.SpeciesName`.

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

Example This example shows a) how to get the data associated with a specified species and, b) how to get the data and associated names for each state (`x`)

1 Import the file for theoscillator model.

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

2 Simulate modelObj.

```
timeseriesObj = sbiosimulate(modelObj);
```

3 Get the data for the species named pol.

```
[t1, data1] = sbiogetnamedstate(timeseriesObj, 'pol');
```

4 Get the data and associated names for each x.

```
[t2, data2, names] = sbiogetnamedstate(timeseriesObj);
```

5 Plot the results of the simulation.

```
plot(t3, data3);  
legend(names);
```

See Also

sbiosimulate

sbiogetsensmatrix

Purpose 3-D sensitivity matrix from simulation results

Syntax
`[T,R,States,Inpfacs] = sbiogetsensmatrix(tsObj)`
`[T,R,States,Inpfacs] = sbiogetsensmatrix(tsObj, StateNames, InpFacNames)`

Arguments

<i>T</i>	Column vector of length <i>m</i> specifying time points for the sensitivity data in <i>R</i> .
<i>R</i>	<i>m</i> -by- <i>n</i> -by- <i>p</i> array of sensitivity data with times, species states, and input factors labeling its first, second, and third dimensions respectively.
<i>States</i>	Contains names of the species states that label the second dimension of <i>R</i> . <i>R</i> (:, <i>i</i> , <i>j</i>) is the time course for the sensitivity of species <i>States</i> { <i>i</i> } to the input factor <i>Inpfacs</i> { <i>j</i> }.
<i>Inpfacs</i>	Contains names of the input factors that label the third dimension of <i>R</i> . <i>R</i> (:, <i>i</i> , <i>j</i>) is the time course for the sensitivity of species <i>States</i> { <i>i</i> } to the input factor <i>Inpfacs</i> { <i>j</i> }.
<i>tsObj</i>	Time series object returned by <code>sbiosimulate</code> . Contains sensitivity data when sensitivity analysis is enabled.
<i>StateNames</i>	Specify states to get sensitivity data from <i>tsObj</i> . Can be an empty array, or a single name, or a cell array of names. When empty array is specified, returns the sensitivity data on all species states contained in <i>tsObj</i> .
<i>InpFacNames</i>	Specify input factors to get sensitivity data from <i>tsObj</i> . Can be an empty array, or a single name, or a cell array of names. When empty array is specified, returns the sensitivity data for all input factors contained in <i>tsObj</i> .

Description

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

The sensitivity data that is logged in `tsObj` 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,States,Inpfacs] = sbiogetsensmatrix(tsObj, StateNames, InpFacNames)` gets sensitivity data for the states specified by `StateNames` and the input factors specified by `InpFacNames`.

Examples

Perform sensitivity analysis of the radiodecay model.

- 1 Import the radio decay model from 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.

```
set(configsetObj.SensitivityAnalysisOptions, ...
    'ParameterInputFactors', sbioselect(modelObj, 'Type', 'parameter'));
get(configsetObj.SensitivityAnalysisOptions, 'ParameterInputFactors')
```

Parameter Object Array

Index:	Name:	Value:	ValueUnits:
1	c	0.5	1/second

- 4** Add species to the SpeciesInputFactors property and display. Use the sbioselect function to retrieve the species objects from the model.

```
set(configsetObj.SensitivityAnalysisOptions, ...  
    'SpeciesInputFactors', modelObj.Species)  
get (configsetObj.SensitivityAnalysisOptions, 'SpeciesInputFactors')
```

Species Object Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	x	1000	molecule
2	z	0	molecule

- 5** Enable SensitivityAnalysis.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true)  
get(configsetObj.SolverOptions, 'SensitivityAnalysis')
```

```
ans =
```

```
1
```

- 6** Simulate and return the results to a time series object.

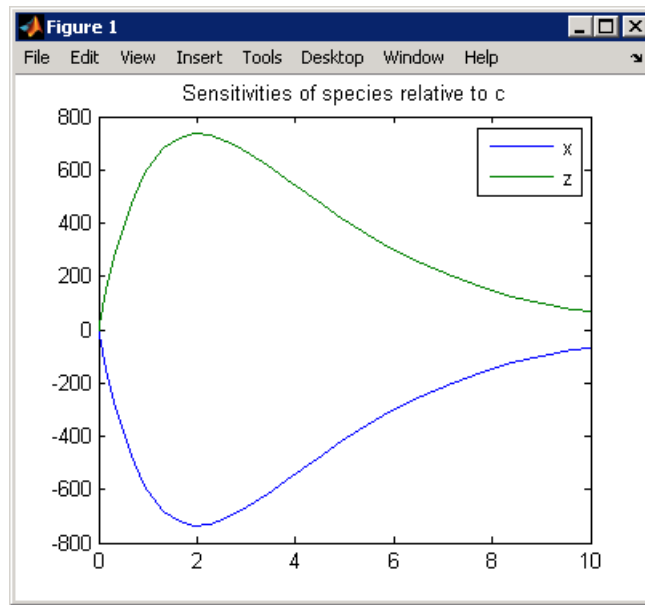
```
tobj = sbiosimulate(modelObj);
```

- 7** Extract sensitivity data from the time series object.

```
[t R snames ifacs] = sbiogetsensmatrix(tobj);
```

- 8** Plot the extracted data.

```
plot(t, R(:, :, 3));  
legend(snames);  
title(['Sensitivities of species relative to ' ifacs{3}]);
```

See Also SimBiology functions `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`.

General information on SimBiology can be returned by specifying *FunctionName* as 'sbio'. General information about a SimBiology object can be returned by specifying *FunctionName* as one of the following: 'AbstractKineticLaw', 'KineticLaw', 'Model', 'Parameter', 'Reaction', 'Root', 'Rule', 'Species', 'Configset', 'CompileOptions', 'ExplicitTauSolverOptions', 'ImplicitTauSolverOptions', 'ODESolverOptions', 'RuntimeOptions', or 'SSASolverOptions'.

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 SimBiology. The fields of the diagnostic structure are:

Type	'error'
MessageID	The message ID for the error, (for example, 'SimBiology:ConfigSetNameClash')
Message	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 The following example shows you how to use `verify` and `sbiolasterror`.

1 Import a model.

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

```
SimBiology Model - RadioactiveDecay
```

```
Model Components:
```

```
Models:          0
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

<i>diagstruct</i>	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 SimBiology. 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 ...
```

Description `sbioloadproject projFilename` loads a SimBiology project from a project file (*projFilename*). If no extension is specified SimBiology assumes a default extension of `.sbproj`.

`sbioloadproject projFilename variableName` loads only the variable *variableName* from the project file.

`sbioloadproject projFilename variableName1 variableName2 ...` loads the specified variables, from the project. The contents of the project file can be displayed by using the `sbiowhos` command.

See Also `sbiosaveproject`, `sbiowhos`, `sbioaddtolibrary`, `sbioremovefromlibrary`

Purpose Construct model object

Syntax

```

modelObj = sbiomodel('NameValue')
modelObj = sbiomodel(...'PropertyName', PropertyValue...)
    
```

Arguments

<i>NameValue</i>	Required property to specify a unique name for a model object. Enter a character string.
<i>PropertyName</i>	Property name for a Model object from the Property Summary table below.
<i>PropertyValue</i>	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 and 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 the model as a submodel to another model object using the model object method addmodel or add as a submodel to a model object with the function copyobj. Add objects to a model object using the methods addkineticlaw, addmodel, addparameter, addreaction, addrule, and addspecies.

All top-level SimBiology model objects can be retrieved from the SimBiology root object. A top-level SimBiology model object has its Parent property set to the SimBiology root object. Submodels have the Parent property set to another SimBiology model and are not stored by the SimBiology root object.

Method Summary

<code>addconfigset (model)</code>	Create configuration set object and add to model object
<code>addmodel (model)</code>	Add submodel object to model object
<code>addparameter (model, kineticlaw)</code>	Create parameter object and add to model or kinetic law object
<code>addreaction (model)</code>	Create reaction object and add to model object
<code>addrule (model)</code>	Create rule object and add to model object
<code>addspecies (model)</code>	Create species object and add to model object
<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object
<code>getadjacencymatrix (model)</code>	Get adjacency matrix from model object
<code>getconfigset (model)</code>	Get configuration set object from model object
<code>getstoichmatrix (model)</code>	Get stoichiometry matrix from model object
<code>removeconfigset (model)</code>	Remove configuration set from model
<code>setactiveconfigset (model)</code>	Set the active configuration set for model object
<code>verify (model)</code>	Validate and verify SimBiology model

**Property
Summary**

Annotation	Store link to URL or file
Models	Contains 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
Species	Array of species 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 summary of modelObj contents.

```
modelObj
```

```
SimBiology Model - cell
```

```
Model Components:
```

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

See Also

addconfigset, addkineticlaw, addmodel, addparameter,
addreaction, addrule, addspecies, sbioroot, copyobj, sbiosimulate

MATLAB functions get, set

Purpose Perform parameter estimation

Syntax

```
[k, result]= sbioparamestim(mode1Obj, tspan, xtarget)
[...]= sbioparamestim(..., species_array)
[...]= sbioparamestim(..., species_array, parameter_array)
[...]= sbioparamestim(..., species_array, parameter_array,
    k0)
[...]= sbioparamestim(..., species_array, parameter_array,
    k0, method)
```

Arguments

<i>k</i>	Vector of estimated parameter values.
<i>result</i>	struct with fields that provide information about the progress of optimization.
<i>tspan</i>	n-by-1 vector representing the time span of the target data <i>xtarget</i> .
<i>xtarget</i>	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 (non-constant) parameters. The number of rows of <i>xtarget</i> must be the same as the number of rows of <i>tspan</i> .
<i>species_array</i>	Either an array of species objects or a cell array of names of species in <i>mode1Obj</i> whose amounts should be matched during the estimation process. The length of the <i>species_array</i> must be the same as the number of columns in <i>xtarget</i> . If there are species with duplicate names in different submodels, either use fully qualified names to identify the species correctly or use an array of species objects to identify species correctly. <i>sbioparamestim</i> assumes that order of the species in <i>species_array</i> is the same as the order used to specify columns of <i>xtarget</i> . For example, a fully

qualified name for a species named `sp1` that is in a submodel named `sub2`, that is inside a submodel `sub1`, that is parented to the top level model `top1`, is `top1.sub1.sub2.sp1`.

<i>parameter_array</i>	<i>parameter_array</i> is either an array of parameter objects or a cell array of names of parameters in <i>modelObj</i> whose values should be estimated. If you do not specify <i>parameter_array</i> , SimBiology estimates all the parameters in the model. When a vector of parameter initial values, (<i>k0</i>), is not specified, <code>sbioparamestim</code> takes the initial values from <i>modelObj</i> . When there are parameters with duplicate names (across submodels or kinetic laws), use either parameter objects or fully qualified parameter names to identify the right parameter object. For a parameter named <code>param1</code> that is scoped at a submodel level in submodel <code>sub1</code> in the top level model <code>top1</code> , the fully qualified name is <code>top1.sub1.param1</code> . If a parameter <code>param2</code> is scoped to the kinetic law of a reaction named <code>reaction2</code> in a top level model <code>top2</code> , its fully qualified name is <code>top2.reaction2.param2</code> .
<i>k0</i>	Array of doubles that holds initial values of parameters to be estimated. The length of <i>k0</i> is same as that of <i>parameter_array</i> . If left unspecified, SimBiology takes initial values for parameters from the model(<i>modelObj</i>).
<i>method</i>	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`.

SimBiology 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)` estimates parameters of the SimBiology model object (*modelObj*) to match all its states given by 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)` estimates parameters of the SimBiology model object, *modelObj* to match only some of its states given by *species_array*.

`[...]= sbioparamestim(..., species_array, parameter_array)` estimates only some of the parameters of the SimBiology model object *modelObj*.

`[...]= 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.

sbioparamestim

Examples

In the examples below, it is assumed that you have a valid SimBiology model object (`m1`) with species named `sp1`, `sp2`, `sp3` and parameters named `p1` and `p2`.

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, use default method.

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

Example 2

Estimate all parameters, but match only some states.

```
% Names of species to match
sp_array = {'sp1', 'sp2', 'sp3'};

%Estimate all parameters
[k, result] = sbioparamestim(m1, tspan, xtarget, sp_array);
```

Example 3

Estimate some parameters, but match all states.

```
% Names of parameters to estimate
p_array = {'p1', 'p2'};
% Select all species
sp_array = sbioselect(m1, 'Type', 'species');
[k, result] = sbioparamestim(m1, tspan, xtarget, sp_array, p_array);
```

Example 4

Estimate parameters specified in `p_array`, species specified in `sp_array`, using different algorithms

```
[k1,r1] = sbioparamestim(m1, tspan, xtarget, sp_array, p_array, ...
    {}, 'fmincon');
[k2,r2] = sbioparamestim(m1, tspan, xtarget, sp_array, p_array, ...
```



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

Example 5

Estimate parameters specified in `p_array`, species specified in `sp_array`, and change default optimization options to use user-specified options

```
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});
```

See Also

- SimBiology functions `sbiomodel`, `sbiogetnamedstate`
- MATLAB function `optimset`
- Genetic Algorithm and Direct Search Toolbox function `gaoptimset`, `psoptimset`

sbioparameter

Purpose Construct parameter object

Syntax

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

Arguments

<i>Obj</i>	Model object or kinetic law object.
<i>NameValue</i>	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 <code>sbioselect</code> to find an object with a specific Name property value.
<i>ValueValue</i>	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 (`parameterObj`) in a simulation, you need to 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. SimBiology objects are constructed with the functions `sbiomodel`, `addmodel`, `addkineticlaw`, and `addreaction`

`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

<code>copyobj</code> (any object)	Copy SimBiology object and its children
<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object

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

sbioparameter

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

Purpose Construct reaction object

Syntax

```

reactionObj = sbioreaction('ReactionValue')
reactionObj = sbioreaction('ReactantsValue',
    'ProductsValue')
reactionObj = sbioreaction('ReactantsValue',
    RStoichCoefficients, 'ProductsValue', PStoichCoefficients)
reactionObj = sbioreaction(...'PropertyName', PropertyValue...)

```

Arguments

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

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', 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 = 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, SimBiology 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

<code>addkineticlaw (reaction)</code>	Create kinetic law object and add to reaction object
<code>addproduct (reaction)</code>	Add product species object to reaction object
<code>addreactant (reaction)</code>	Add species object as a reactant to reaction object
<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object
<code>rmproduct (reaction)</code>	Remove species object from reaction object products
<code>rmreactant (reaction)</code>	Remove species object from reaction object reactants

Property Summary

Active	Indicate object 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 a reaction is reversible or irreversible
Stoichiometry	Species coefficients in a 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.  
help(robj1, '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](#)

sbioregisterunit

Purpose Create user-defined unit

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 $\text{Multiplier} \times \text{Composition}$ and records the unit in the `UserDefinedUnits` vector of `sbiroot` and adds it to the user-defined library.

`sbioregisterunit('Name', 'Composition', Multiplier, Offset)` creates a unit with the specified offset. Available units can be listed 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 SimBiology uses 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 6.0221×10^{23} molecule. The *Multiplier* is 6.0221×10^{23} .
- *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.

Examples **Example 1**

```
sbioregisterunit('pint2','inch^3',28.875);  
sbioregisterunit('celsius2','fahrenheit',9/5,32);
```

Example 2

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

```
sbioregisterunit('firstconstant', '1/second', 1);  
sbioregisterunit('secondconstant', '1/molecule*second', 1);
```

- 2 Display the unit using the command sbiowhos

```
sbiowhos -userdefined -unit
```

```
SimBiology UserDefined Units
```

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

See Also

sbioshowunits, sbioregisterunitprefix, sbiounregisterunit

sbioregisterunitprefix

Purpose Create user-defined unit prefix

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 `sbiroot` and to the user-defined library. You can see the available unit prefixes with the `sbishowunitprefixes` 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.

Example This example shows you how to register a unit prefix.

```
sbioregisterunitprefix('peta', 15);  
sbiowhos -userdefined -unitprefix
```

```
SimBiology UserDefined Unit Prefixes
```

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

See Also `sbishowunits`, `sbishowunitprefixes`, `sbiounregisterunit`

Purpose

Remove abstract kinetic law or unit from user-defined library

Syntax

```
sbioremovefromlibrary (abstkineticlawObj)  
sbioremovefromlibrary ('Type', 'Name')
```

Description

`sbioremovefromlibrary` removes an abstract kinetic law or a unit from the user-defined library. `sbioremovefromlibrary (abstkineticlawObj)` removes the abstract kinetic law object (`abstkineticlawObj`) from the user-defined library. `abstkineticlawObj` will no longer be available automatically in future MATLAB sessions.

SimBiology 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`.

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

`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 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.

Example

Shows you 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.

sbioremovefromlibrary

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined
```

```
Abstract Kinetic Law Object 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`, `sbioregisterunit`, `sbioregisterunitprefix`, `sbiounregisterunit`, `sbiounregisterunitprefix`

Purpose Delete all model and simulation objects

Syntax `sbioreset`

Description `sbioreset` delete 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 the top-level SimBiology model objects, available units, unit prefixes and kinetic law objects. A top-level SimBiology model object has its `Parent` property set to the SimBiology root object. A SimBiology model object that has its `Parent` property set to another SimBiology model is a submodel and is not stored by the SimBiology root.

To add an abstract kinetic law to the SimBiology root user-defined library, use the `addtolibrary` 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 Shows you 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** The command `clear all` clears the workspace, but 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** The command `sbioreset` deletes 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
modelObj = sbioroot('modelName')
```

Arguments

<i>rootObj</i>	Return sbioroot to this object.
<i>modelObj</i>	Return the model with name <i>modelName</i> to this object.
<i>modelName</i>	Specify the name of the model that is on the root 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.

modelObj = sbioroot('modelName') returns the top-level SimBiology model with name, modelName to modelObj. A top-level SimBiology model object has its Parent property set to the SimBiology root object. A SimBiology model object that has its Parent property set to another SimBiology model is a submodel and is not stored by the SimBiology root.

The units define the set of core units and user-defined units. A user-defined unit can be added with the sbioregisterunit function. You can remove user-defined unit with the sbiounregisterunit function. The unit prefixes define the set of core unit prefixes and user-defined unit prefixes.

You can add a user-defined unit prefix with the sbioregisterunitprefix function. Remove a user-defined unit prefix with the sbiounregisterunitprefix function. The abstract kinetic law objects define the core abstract kinetic law objects and user-defined abstract kinetic law objects. SimBiology uses abstract kinetic law objects when configuring a SimBiology reaction object's KineticLaw property with the addkineticlaw function.

Method Summary

<code>copyobj</code> (any object)	Copy SimBiology object and its children
<code>delete</code> (any object)	Delete SimBiology object
<code>reset</code> (root)	Delete all model objects from the root object

Property Summary

<code>BuiltInKineticLaws</code>	Contain built-in kinetic laws
<code>BuiltInUnitPrefixes</code>	Contain built-in unit prefixes
<code>BuiltInUnits</code>	Contain built-in units
<code>Models</code>	Contains all model objects
<code>Type</code>	Display top-level SimBiology object type
<code>UserDefinedKineticLaws</code>	Contain user-defined kinetic laws
<code>UserDefinedUnitPrefixes</code>	Contain user-defined unit prefixes
<code>UserDefinedUnits</code>	Contain user-defined units

Examples

- 1 Get all SimBiology model objects contained by the root.

```
rootObj = sbioroot;
allmodels = get(rootObj, 'Models');
```

- 2 Get the model with name `cell` (if model is in root).

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

See Also

`addkineticlaw`, `sbioimodel`, `sbioregisterunit`, `sbiounregisterunit`, `sbioreset`

sbiorule

Purpose Construct rule object

Syntax

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

Arguments

<i>RuleValue</i>	Enter a character string within quotes. For example, enter the algebraic rule 'Va*Ea + Vi*Ei - K2'.
<i>RuleTypeValue</i>	Enter 'algebraic', 'assignment', or 'rate'. An algebraic or rate rule is evaluated at each time step during the simulation. An assignment rule is evaluated once before the simulation starts. Note: if a species or parameter is marked constant, you can still assign an initial value using an assignment rule. The amount or value gets assigned according to the rule and then remains constant during the simulation.

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

<code>copyobj</code> (any object)	Copy SimBiology object and its children
<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object

Property Summary

Active	Indicate object 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

Example 1

Construct a rule object and copy to a model object.

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

Example 2

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

```
help(robj, 'RuleType')
```

Example 3

List the properties for a rule.

```
get(robj)  
  
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`. SimBiology returns an error if `projFilename.sbproj` is not writable.

`sbiosaveproject` creates the binary SimBiology project file named `simbiology.sbproj`. SimBiology 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(fileName, '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
```

See Also `sbioloadproject`, `sbiowhos`, `sbioaddtolibrary`, `sbioremovefromlibrary`

sbioselect

Purpose

Search for objects with specified constraints

Syntax

```
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', Number, ...)
```

Arguments

<i>Obj</i>	SimBiology object to search. If an object is not specified, SimBiology searches the root.
<i>PropertyName</i>	Any property of <i>Obj</i> .
<i>PropertyValue</i>	Valid PropertyValue.
<i>TypeValue</i>	Type of SimBiology object to search, for example sbiomodel, species, reaction, kineticlaw.
<i>Condition</i>	Constraint to impose on the search. See the following table for a list of Conditions.

Description

sbioselect searches for objects with specified constraints.

Out = sbioselect(Obj, 'PropertyName', PropertyValue) finds the objects with the property name (*PropertyName*) and property value (*PropertyValue*) contained in any SimBiology object (*Obj*) or an array of SimBiology objects. If an object is not specified, SimBiology searches the root.

Out = sbioselect(Obj, 'Type', 'TypeValue', 'PropertyName', PropertyValue) finds the objects of *Type*, *TypeValue*, with the property name (*PropertyName*) and property value (*PropertyValue*) contained in any SimBiology object (*Obj*) or an array of SimBiology objects. *TypeValue* is the type of SimBiology object 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 matches the condition (*Condition*).

Condition	Example PropertyName	Example PropertyValue
>	InitialAmount	50
>	InitialAmount	50
==	InitialAmount	50
==i	Name	x
~=	InitialAmount	50
~=i	Name	x
>=	InitialAmount	50
<=	InitialAmount	50
between	InitialAmount	[200 300]
~between	InitialAmount	[200 300]
contains	Reactant	Species or species object array
regexp	Name	Value*
~regexp	Name	Value*
regexp <i>i</i>	Name	Value*
~regexp <i>i</i>	Name	Value*

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

Any combination of property name/property value pairs and conditions can be combined in the `sbioselect` command.

`out= sbioselect(Obj, 'depth', Number, ...)` finds objects using a model search depth of *Number*. Valid numbers are positive integer values and `inf`. If *Number* is `inf`, `Obj` and all of its children are searched. If *Number* is 1, children of `Obj` will not be searched. By default, *Number* is `inf`.

- The condition types supported for numeric properties are `>`, `<`, `=`, `~=`, `>=`, `between` and `~between`. Conditions for range are `'between'` and `'~between'`.

```
PName= sbioselect (InitialAmount, 'Between', [200, 300])
```

- The condition types supported for string properties are `==`, `==i`, `~=`, `~=i`, `regexp`, `~regexp`, `regexpi` and `~regexpi`. Case conditions are `'CaseSensitive'` and `'CaseInsensitive'`. The `CaseSensitive` and `CaseInsensitive` conditional values can be used only for those properties whose values are strings. If they are used on a property whose value is not a string, it is ignored.

```
Out = sbioselect(Robj, 'Name', 'CaseSensitive', MyModel)
```

- The condition `'Contains'` can be used only for those properties whose values are an array of SimBiology objects. The value for `Contains` is one of the objects that should be in the array. `PValue` is a species or species object array.

```
Out = sbioselect(reactionObj, 'Reactant', 'Contains',  
modelObj.Species)
```

- The `regexp` and `regexpi` conditional value supports any of the expressions supported by the functions `regexp` and `regexpi` (`regexp`). When a string property value is searched for without specifying a condition, it 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'`.

```
Out = sbioselect(Robj, 'Name', 'RegExp', T*)
```

- Conditions for relational operators: >, <, =, ~=, >=, <=

Propertyname=InitialAmount, Condition is >, and PropertyValue=50

Examples

Find a SimBiology model called cell in the root object.

```
sbioselect(sbioroot, 'Type', 'sbioModel', 'Name', 'cell')
```

Find all SimBiology reactions of SimBiology model (modelObj) that use species A as a reactant.

```
speciesA = sbioselect(modelObj, 'Type', 'species', 'Name', ...
    'A');
out = sbioselect(modelObj, 'Type', 'reaction', 'Where', ...
    'Reactants','contains', speciesA);
```

Find all SimBiology species of SimBiology model (modelObj) that have an InitialAmount that ranges between 100 and 300.

```
out = sbioselect(m, 'Type', 'species', 'Where', ...
    'InitialAmount', 'between', [100 300]);
```

See Also

regexp

sbioshowunitprefixes

Purpose Information about registered unit prefixes

Syntax

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

Description `sbioshowunitprefixes` returns information about registered unit prefixes.

`Name = sbioshowunitprefixes` returns the names of the registered unit prefixes to `Name` as a cell array of strings.

`[Name, Multiplier] = sbioshowunitprefixes` returns the multiplier for each prefix in `Name` to `Multiplier`. `Multiplier` is 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`. `Builtin` is 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.

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

- `Name` is the name of the 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 `Name` to the base unit. For example the multiplier in picomole is $10e-12$.

Examples

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

See Also `sbioregisterunit`, `sbioshowunits`, `sbioconvertunits`

Purpose

Information about registered units

Syntax

```
Name = 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')
```

Description

Name = sbioshowunits returns the names of the registered units to *Name* as a cell array of strings.

[*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*. *Builtin* is 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.

[*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.

- *Name* is the name of the built-in or user-defined unit. *Name* must begin with characters and can contain characters, underscores or numbers.
- *Composition* shows the combination of base and derived units that defines the unit *Name*. For example molarity is mole/liter.

sbioshowunits

- *Multiplier* is 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* 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.

Examples

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

See Also

sbioregisterunit, sbioshowunitprefixes, sbioconvertunits

Purpose

Simulate model object

Syntax

```
[T,X] = sbiosimulate(modelObj)
[T,X] = sbiosimulate(modelObj, configsetObj)
timeseriesObj = sbiosimulate(modelObj)
```

Description

`[T,X] = sbiosimulate(modelObj)` simulates a model object (`modelObj`) using the active configuration set attached to the model (`modelObj`).

- `modelObj` — SimBiology model object. Enter the variable name for a model object.

`[T,X] = 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.

- `configsetObj` — A `configset` object stores simulation-specific information. A SimBiology model can contain multiple configuration sets with only one being active at any given time. The active configuration set contains the settings that are used during the simulation.
- `T` — 1-by-`n` vector where `n` is the number of times the reactions fired. `T` defines the time steps of the firing of the reactions.
- `X` — `n`-by-`m` matrix where `n` is the number of times the reactions fired and `m` is the number of species in the model or the number of `StatesToLog`. Each column of `X` defines the variation in the amount of a species over time. To get the simulation values for the first species logged:

```
FirstSpecies = X(:,1)
```

`timeseriesObj = sbiosimulate(modelObj)` simulates a model object (`modelObj`) using the active configuration set attached to the

model (*modelObj*) and returns the results to time series object (*timeseriesObj*).

Use *timeseriesObj.time* to access the time vector from the simulation results. Use *timeseriesObj.data* to access the state matrix from the simulation results. *timeseriesObj* also contains information about the configuration set used to simulate the model. Use `get(timeseriesObj)` to view this information.

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`. To use command-line help to get more information on these methods, for example, type `'help SimBiology.Model.getConfigset'`.

Property Summary

Configuration set property summary

Active	Indicate object 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 you 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 theoscillator 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', 'ode45');  
set(configsetObj, 'StopTime', 10);
```

- 4 Simulate 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).

sbiosimulate

- 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 `modelObj`.

```
timeseriesObj = sbiosimulate(modelObj);
```

- 3 Plot the results of the simulation.

```
plot(timeseriesObj.Time, timeseriesObj.Data);  
legend(timeseriesObj.SpeciesNames)
```

See Also

SimBiology object constructor `sbiomodel`, model object method `addconfigset`

Purpose Construct species object

Syntax

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

Arguments

<i>NameValue</i>	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.
<i>InitialAmountValue</i>	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 species object name (*NameValue*)

In order for a species object to be used in a simulation, the species object must be added to a SimBiology model object using copyobj. You can useaddspecies to simultaneously create a species object and add it to a model object. A SimBiology model object is constructed with the function sbiomodel.

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.

Specifying Species Names in SimBiology

SimBiology species names can have any number, letter, or character, for example, N-acetyl-D-glucosamine. The names of species are case-sensitive.

Note the following constraints,

- Name cannot be left empty.
- Name cannot be the literal words `null` and `time`. Note that you could specify species names with these words contained within the name, for example, `nullaminoacids` or `nullnucleotides`.
- If you are using a species name that is not a valid MATLAB variable name, do the following:
 - Enclose the name in square brackets when writing a reaction rate equation or a rule. For example, enclose `[DNA polymerase+]` within brackets in reaction rates and rules.
 - Enter the name without brackets when you are creating the species or when you are adding the reaction. For example, enter `DNA polymerase+` when specifying the name of the species or while writing the reaction.
- Name cannot be the characters `i`, `or` `j`, or contain `->`, `<`, `>`, `[`, or `]`. The names `i` and `j` are reserved MATLAB characters. Because expressions in abstract kinetic laws, reaction rates, and rules are

considered to be MATLAB code, SimBiology evaluates i or j as an imaginary number and not as the value of species i or j . For example, an expression $V*S^i/K$ is interpreted to have three names, V , S , and K , instead of four. Use brackets to protect such variables.

- If a variable in a reaction rate equation or rule has the same name as a MATLAB function, SimBiology evaluates the expression as a call to the MATLAB function. In general, when creating variable names, you should avoid using MATLAB function names or variable names that are invalid in MATLAB.

If you change the Name 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.

SimBiology automatically updates species names 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

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
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 with name H2O and initial amount 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(sobj)
```

```

        Annotation: ''
        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 with the names `glucose_6_phosphate` and `glucose_6_phosphate_dehydrogenase`.

```

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

```

- 2 Set initial amount of `glucose_6_phosphate` to 100 and verify.

```

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

```

MATLAB returns

```

ans =

    100

```

See Also

SimBiology method `addspecies`

MATLAB functions `get` and `set`

sbiunitcalculator

Purpose Convert value between units

Syntax `result = sbiunitcalculator('fromUnits', 'toUnits', Value)`

Description `result = sbiunitcalculator('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 = sbiunitcalculator('mile/hour', 'meter/second', 1)`

See Also `sbioshowunits`

Purpose Remove user-defined unit from root and library

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`.

Example **1** Create units for the rate constants of a first order and second order reactions.

```
sbioregisterunit('firstconstant', '1/second', 1);  
sbioregisterunit('secondconstant', '1/molecule*second', 1);
```

2 Display the unit using the command `sbiowhos`

```
sbiowhos -userdefined -unit
```

```
SimBiology UserDefined Units
```

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

3 Unregister one of the units and display the user-defined units available.

```
sbiounregisterunit('firstconstant');  
sbiowhos -userdefined -unit
```

sbionregisterunit

SimBiology UserDefined Units

Index:	Name:	Composition:	Multiplier:	Offset:
1	secondconstant	1/molecule*second	1.000000	0.000000

See Also

sbioshowunits, sbioregisterunit, sbionregisterunitprefix, sbioroot, sbiowhos

Purpose Remove user-defined unit prefix from root and library

Syntax `sbionregisterunitprefix('Name')`

Description `sbionregisterunitprefix('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`.

Example **1** Register a unit prefix.

```
sbioregisterunitprefix('peta', 15);  
sbiowhos -userdefined -unitprefix
```

```
SimBiology UserDefined Unit Prefixes
```

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

2 Unregister the unit prefix.

```
sbionregisterunitprefix('peta');
```

See Also `sbioshowunitprefixes`, `sbioshowunits`, `sbioregisterunitprefix`, `sbionregisterunit`, `sbiroot`, `sbiowhos`

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 the table below:

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 contents of 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 an Systems Biology Markup Language (SBML) format. Enter either a filename or a path and filename supported by your operating system. If the filename does not have the extension `.xml`, then `.xml` is appended to end of the filename.

Description

`sbmlexport(modelObj)` exports a SimBiology model object (`modelObj`) to a file with a Systems Biology Markup Language (SBML) Level 2 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.

Functional Characteristics and Limitations

The `sbmlexport` function,

- Exports SBML Level 2 compatible files
- Exports SBML compliant unit definitions
- Does not support submodels
- Does not export features that are not supported by SBML. These are,

- Abstract kinetic law name and corresponding expression (note that sbmlexport exports the reaction rate equation)
- Configurations sets
- Active property
- UserData
- Tag

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

sbmlimport, sbiomodel, sbiosaveproject

sbmlimport

Purpose Import SBML-formatted file

Syntax `modelObj = sbmlimport('FileName')`

Arguments

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

Description

`modelObj = sbmlimport('FileName')` imports a 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.

Functional Characteristics and Limitations

- `sbmlimport` supports SBML Levels 1 and 2.
- If there are MATLAB incompatible variable names in any mathematical expression (for example, rules or rate expressions), SimBiology inserts brackets around those variable names in that expression.
- If the model has a single compartment, the model is read in as a top-level model. If there are multiple compartments, SimBiology returns a warning and does not read the SBML file.
- SimBiology does not support model volume and volume units, function definitions, events, and piecewise kinetics.
- SimBiology does not support brackets in species or parameter names.

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

sbmlexport, sbiosimulate
MATLAB functions get and set

Methods — By Category

Objects (p. 3-2)	SimBiology objects
Abstract Kinetic Laws (p. 3-2)	Work with abstract kinetic law objects
Configuration Sets (p. 3-3)	Work with configuration set objects
Kinetic Laws (p. 3-4)	Create parameter objects and work with kinetic law objects
Models (p. 3-5)	Create SimBiology objects and work with model objects
Parameters (p. 3-6)	Work with parameter objects
Reactions (p. 3-7)	Create kinetic law and species objects and work with reaction objects
Root (p. 3-8)	Work with the root object
Rules (p. 3-9)	Work with rule objects
Species (p. 3-10)	Methods for species objects
Using Object Methods (p. 3-11)	Command-line syntax for using methods with SimBiology objects

Objects

AbstractKineticLaw object	Kinetic law information in library
Configset object	Solver settings information for model simulation
KineticLaw object	Kinetic law information for reaction
Model object	Model and component information
Parameter object	Parameter and scope information
Reaction object	Options for model reactions
Root object	Hold models, unit libraries, and abstract kinetic law libraries
Rule object	Hold rule for species and parameters
Species object	Options for model species

Abstract Kinetic Laws

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

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

Kinetic Laws

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

Models

<code>addconfigset (model)</code>	Create configuration set object and add to model object
<code>addmodel (model)</code>	Add submodel object to model object
<code>addparameter (model, kineticlaw)</code>	Create parameter object and add to model or kinetic law object
<code>addreaction (model)</code>	Create reaction object and add to model object
<code>addrule (model)</code>	Create rule object and add to model object
<code>addspecies (model)</code>	Create species object and add to model object
<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object
<code>getadjacencymatrix (model)</code>	Get adjacency matrix from model object
<code>getconfigset (model)</code>	Get configuration set object from model object
<code>getstoichmatrix (model)</code>	Get stoichiometry matrix from model object
<code>removeconfigset (model)</code>	Remove configuration set from model
<code>setactiveconfigset (model)</code>	Set the active configuration set for model object
<code>verify (model)</code>	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

Reactions

<code>addkineticlaw</code> (reaction)	Create kinetic law object and add to reaction object
<code>addproduct</code> (reaction)	Add product species object to reaction object
<code>addreactant</code> (reaction)	Add species object as a reactant to reaction object
<code>copyobj</code> (any object)	Copy SimBiology object and its children
<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object
<code>rmproduct</code> (reaction)	Remove species object from reaction object products
<code>rmreactant</code> (reaction)	Remove species object from reaction object reactants

Root

copyobj (any object)

Copy SimBiology object and its children

delete (any object)

Delete SimBiology object

reset (root)

Delete all model objects from the root object

Rules

copyobj (any object)

Copy SimBiology object and its children

delete (any object)

Delete SimBiology object

display (any object)

Display summary of SimBiology object

Species

copyobj (any object)

Copy SimBiology object and its children

delete (any object)

Delete SimBiology object

display (any object)

Display summary of SimBiology object

Using Object Methods

Command-line syntax for using methods with SimBiology objects

Constructing (Creating) Objects
(p. 3-11)

Using Object Methods (p. 3-11)

Help for Objects, Methods and
Properties (p. 3-12)

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 references by a reaction, us the reaction object methods `addkineticlaw`, `addparameter`, `addproduct`, and `addreactant`.

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

Note, `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.

<code>help sbio</code>	Display a list of functions and methods.
<code>help FunctionName</code>	Display function information.
<code>sbiohelp('MethodName')</code>	Display method information.
<code>sbiohelp('PropertyName')</code>	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 MATLAB `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	<code>sbioabstractkineticlaw</code>	Construct abstract kinetic law object
----------------------------	-------------------------------------	---------------------------------------

Method Summary

<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object

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

addconfigset (model)

Purpose Create configuration set object and add to model object

Syntax

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

Arguments

<i>modelObj</i>	Model object. Enter a variable name.
<i>NameValue</i>	Descriptive name for a configuration set object. Reserved words 'active' and 'default' are not allowed.
<i>configsetObj</i>	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 below in the property summary.

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*.

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

<code>copyobj</code> (any object)	Copy SimBiology object and its children
<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object

Property Summary

Properties for configuration set objects

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

addconfigset (model)

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 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

Model object methods `getconfigset`, `removeconfigset`, `setactiveconfigset`

MATLAB functions `get` and `set`.

addkineticlaw (reaction)

Purpose Create kinetic law object and add to reaction object

Syntax

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

Arguments

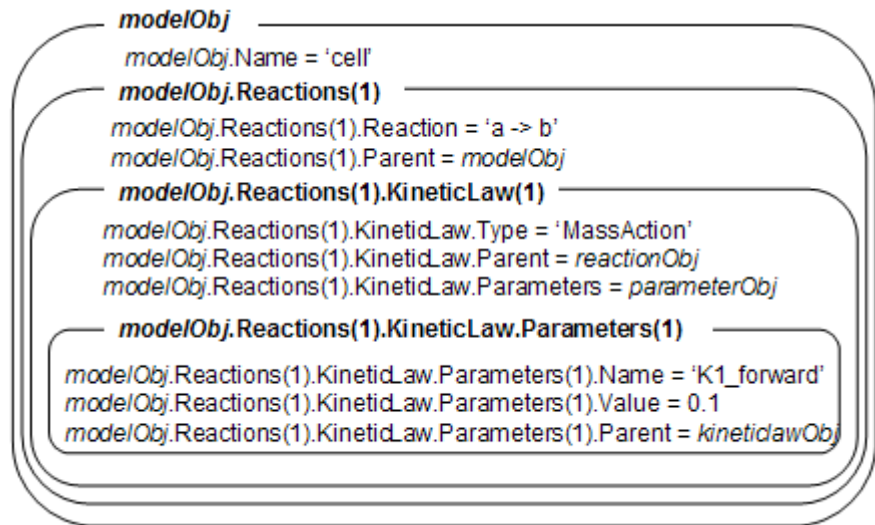
<i>reactionObj</i>	Reaction object. Enter a variable name for a reaction object.
<i>KineticLawNameValue</i>	Property to select the type of kinetic law object to create. For builtin kinetic law valid values are: 'Unknown', 'MassAction', 'Henri-Michaelis-Menten', 'Henri-Michaelis-Menten-Reversible', 'Hill-Kinetics', 'Iso-Uni-Uni', 'Ordered-Bi-Bi', 'Ping-Pong-Bi-Bi', 'Competitive-Inhibition', 'NonCompetitive-Inhibition', 'UnCompetitive-Inhibition'. Find valid <i>KineticLawNameValues</i> by querying the SimBiology root object with the commands: <code>get(sbioroot, 'BuiltInKineticLaws')</code> , and <code>get(sbioroot, 'UserDefinedKineticLaws').sbiowhos -kineticlaw</code> lists <code>BuiltInKineticLaws</code> and <code>UserDefinedKineticLaws</code> in the SimBiology root. The root contains all <code>BuiltInKineticLaws</code> and all <code>UserDefinedKineticLaws</code> that are added using <code>sbioaddtolibrary</code> or <code>addtolibrary</code> .

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 6-27 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 below in the property summary.

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

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

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');
```

addkineticlaw (reaction)

```
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)
```

SimBiology adds an abstract kinetic law expression to the reaction object (*reactionObj*).

The Henri-Michaelis-Menten kinetic law has two parameters (V_m and K_m) 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 V_m_d , another parameter object (*parameterObj2*) named K_m_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);
```

SimBiology 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'});
```

SimBiology 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.

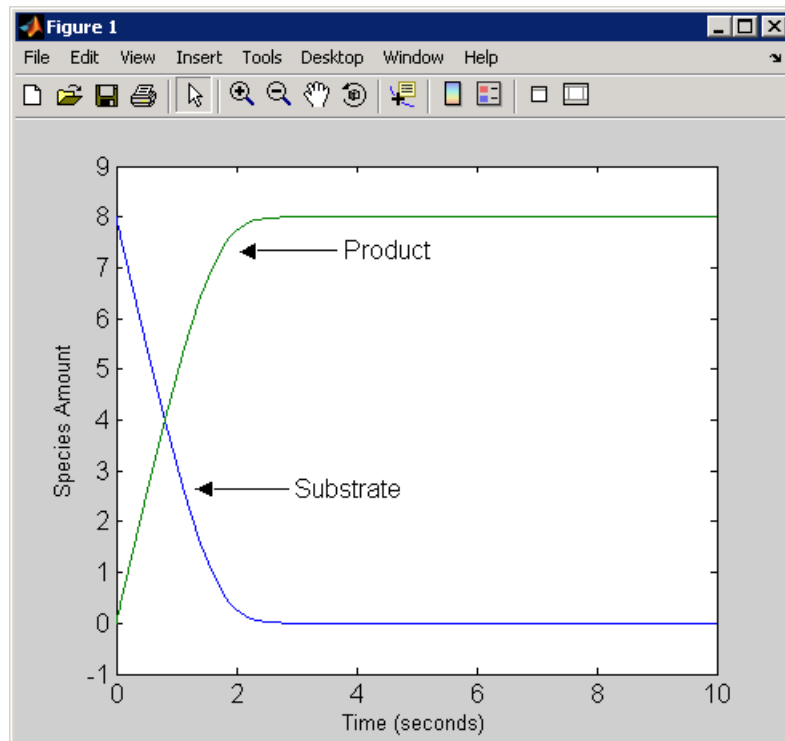
- 5 Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
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)
```

addkineticlaw (reaction)



Example 2

Example using 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')
```

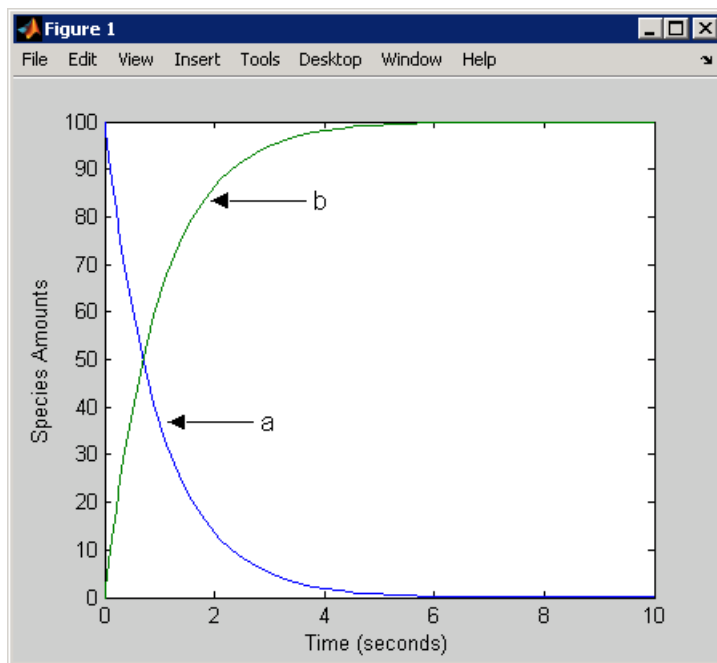
```
ans =  
    k_forward*a
```

- 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 default value = 1.0.

addkineticlaw (reaction)



See Also `addreaction`, `setparameter`

Purpose Add submodel object to model object

Syntax
`submodelObj = addmodel(modelObj, 'NameValue')`
`submodelObj = addmodel(...'PropertyName', PropertyValue...)`

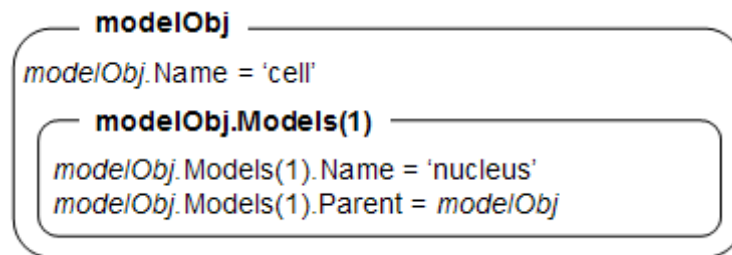
Arguments

<code>modelObj</code>	Model object. Enter a name for a model object.
<code>NameValue</code>	Descriptive name for a model object. Enter a unique character string. A model object can be referenced by other objects using this property.
<code>submodelObj</code>	Model object to be added as 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')
```



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`.

addmodel (model)

`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')`.

Examples

```
modelObj = sbiomodel('cell');  
submodelObj = addmodel(modelObj, 'nucleus');  
submodelObj = addmodel(modelObj, 'cytoplasm');
```

See Also

`sbiomodel`

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

<i>Obj</i>	Model or kinetic law object. Enter a variable name for the object.
<i>NameValue</i>	Property for a parameter object. Enter a unique character string. NameValue can be a cell array of parameter names. 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.
<i>ValueValue</i>	Property for a parameter object. Enter a number.

Description

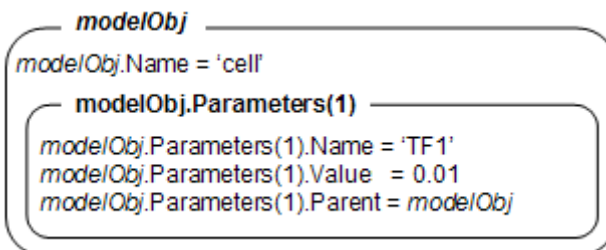
`parameterObj = addparameter(Obj, 'NameValue')` creates a parameter object and returns the object (*parameterObj*). In the parameter object, this method assigns a value (*NameValue*) to the property Name, assigns a value 1 to the property Value, and assigns the model or kinetic law object to the property 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 parameter. If a parameter object is defined with a model object as its

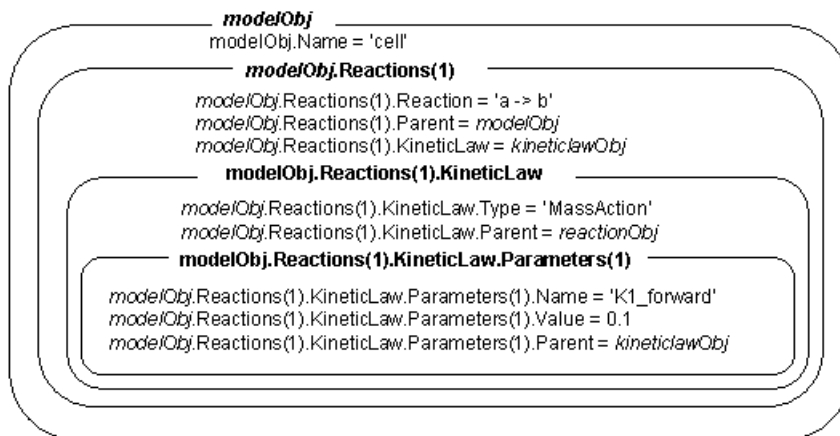
addparameter (model, kineticlaw)

parent, then all objects within the model (including all rules, submodels 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,

addparameter (model, kineticlaw)

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

<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object

Property Summary

Properties for parameter objects

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

Example

1 Create model object, then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

addparameter (model, kineticlaw)

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

Parameter Object 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

Parameter Object Array

Index:	Name:	Value:	ValueUnits:
1	K2	1	

See Also

MATLAB functions—copyobj, delete

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

<i>reactionObj</i>	Reaction object. Enter a name for the reaction object.
<i>NameValue</i>	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 <code>sbiiselect</code> to find an object with a specific <i>NameValue</i> .
<i>speciesObj</i>	Species object.
<i>Stoichcoeffieient</i>	Stoichiometric coefficients for products, length of array equal to length of <i>NameValue</i> or length of <i>speciesObj</i> .

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*, and assigns the parent object of the *reactionObj* to the property *Parent*. 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*.

If the parent object (always a model object) of a reaction does not include a species with the specified name (*NameValue*), a species object is created and assigned to the parent object property *Species*. You can create a species object with the function *sbiospecies*, or create and add a species object to a model 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, this method 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, this method 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 *SimBiology* updates the reaction 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.

See “Specifying Species Names in *SimBiology*” on page 4-41 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 + C -> U');
```

- 2 Modify the reaction of the *reactionObj* from $A + C \rightarrow U$ to $A + C \rightarrow U + 2 H$.

addproduct (reaction)

```
speciesObj = addproduct(reactionObj, 'H', 2);
```

See Also sbiospecies, addspecies

Purpose

Add species object as a reactant to reaction object

Syntax

```
speciesObj = addreactant(reactionObj, 'NameValue')
addreactant(reactionObj, speciesObj, Stoichcoeffieient)
addreactant(reactionObj, 'NameValue', Stoichcoeffieient)
```

Arguments

<i>reactionObj</i>	Reaction object.
<i>NameValue</i>	Name property of a species object. Enter a unique character string, for example, 'glucose 6-phosphate'. A species object can be referenced by other objects using this property. You can use the function <code>sbioselect</code> to find an object with a specific Name property value.
<i>speciesObj</i>	Species object or cell array of species objects.
<i>Stoichcoeffieient</i>	Stoichiometric coefficients for reactants, length of array equal to length of <i>NameValue</i> or length of <i>speciesObj</i> .

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, and assigns the parent object of the *reactionObj* to the property Parent. 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.

If the parent object (always a model object) of a reaction does not include a species with the specified name (*'NameValue'*), SimBiology creates a species object and assigns the parent object of the *reactionObj* to the parent object property Species. You can create a species object with the function `sbiospecies`, or create and add a species object to a model object with the method `addspecies`.

addreactant (reaction)

`addreactant(reactionObj, speciesObj, Stoichcoeffieient)`, in the species object (`speciesObj`), this method 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, this method 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`.

A species object must have a unique name at the level at which it is created. For example, a model object cannot contain two species objects named `H2O`. However, a submodel of the model that contains the species `H2O` can also contain a species named `H2O`.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species `SimBiology` updates the reaction 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.

See “Specifying Species Names in SimBiology” on page 4-41 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 sbiospecies, addspecies

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

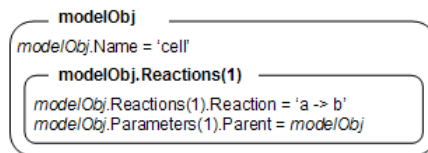
<i>modelObj</i>	SimBiology model object
<i>ReactionValue</i>	Specify the reaction equation. Enter a character string. A hyphen preceded by a space and followed by a right angle bracket (->) indicate reactants going forward to products. A hyphen with left and right angle brackets (<->) indicate a reversible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

<i>RStoichCoefficients</i>	Stoichiometric coefficients for reactants, length of array equal to length of <i>ReactantsValue</i> .
<i>PStoichCoefficients</i>	Stoichiometric coefficients for products, length of array equal to length of <i>ProductsValue</i> .

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')
```



If a species specified in a reaction does not exist, a species object is created and assigned to the model object property *Species*. You can manually add a species to a model 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 can be copied to a SimBiology model object with the function, `copyobj`. You can remove 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

addreaction (model)

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

<code>addkineticlaw (reaction)</code>	Create kinetic law object and add to reaction object
<code>addproduct (reaction)</code>	Add product species object to reaction object
<code>addreactant (reaction)</code>	Add species object as a reactant to reaction object
<code>copyobj (any object)</code>	Copy SimBiology object and its children

<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object
<code>rmproduct</code> (reaction)	Remove species object from reaction object products
<code>rmreactant</code> (reaction)	Remove species object from reaction object reactants

Property Summary

Properties for reaction objects

Active	Indicate object 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 a reaction is reversible or irreversible
Stoichiometry	Species coefficients in a reaction
Tag	Specify label for SimBiology object

addreaction (model)

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, 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

<i>modelObj</i>	Model object to which to add the rule.
<i>RuleValue</i>	Enter a character string within quotes. For example, enter the algebraic rule 'Va*Ea + Vi*Ei - K2'.
<i>RuleTypeValue</i>	Enter 'algebraic', 'assignment', or 'rate'. An algebraic or rate rule is evaluated at each time step during the simulation. An assignment rule is evaluated once before the simulation starts. Note: if a species or parameter is marked constant, you can still assign an initial value using an assignment rule. The amount or value gets assigned according to the rule and then remains constant during the simulation.

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

<code>copyobj</code> (any object)	Copy SimBiology object and its children
<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object

Property Summary

Properties for rule objects

<code>Active</code>	Indicate object use during simulation
<code>Annotation</code>	Store link to URL or file
<code>Name</code>	Specify name of object
<code>Notes</code>	HTML text describing SimBiology object
<code>Parent</code>	Indicate parent object
<code>Rule</code>	Specify species and parameter interactions
<code>RuleType</code>	Specify type of rule for rule object

addrule (model)

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 default RuleType.

- 1 Create a model object, and then add a rule object.

```
modelObj = sbiomodel('cell');  
ruleObj = addrule(modelObj, '0.1*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: '0.1*B-A'  
RuleType: 'algebraic'  
Tag: ''  
Type: 'rule'  
UserData: []
```

Add rule with RuleType property set to rate.

- 1 Create model object, 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 (model)

Purpose Create species object and add to model object

Syntax

```
speciesObj = addspecies(modelObj, NameValue)
speciesObj = addspecies(modelObj, NameValue,
    InitialAmountValue)
speciesObj = addspecies(...'PropertyName', PropertyValue...)
```

Arguments

<i>modelObj</i>	Model object.
<i>NameValue</i>	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.
<i>InitialAmountValue</i>	Initial amount value for the species object. Enter double. Positive real number, default = 0.

Description

speciesObj = `addspecies(modelObj, 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, assigns the model object (*modelObj*) to the property Parent. In the model object, this method assigns the species object to the property Species.

speciesObj = `addspecies(modelObj, NameValue, InitialAmountValue)`, in addition to the above, this method assigns an initial amount (*InitialAmountValue*) for the species.

You can also add a species to a reaction using the methods `addreactant` and `addproduct`. When a reaction is defined with a species not in the model object (*modelObj*), SimBiology creates a species object.

A species object must have a unique name at the level at which it is created. For example, a model object cannot contain two species objects named H2O. However, a submodel of the model that contains the species H2O can also contain 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 model (`modelObj`) with `get(modelObj, 'Species')` or the properties of the first species with `get(modelObj.Species(1))`.

A species in a rule has to be in the model object with the rule. This is different from parameters in a rule, where a parameter can be in the model object or in the kinetic law object or resolve hierarchically.

`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.

Specifying Species Names in SimBiology

SimBiology species names can have any number, letter, or character, for example, N-acetyl-D-glucosamine. The names of species are case-sensitive.

Note the following constraints,

- Name cannot be left empty.
- Name cannot be the literal words `null` and `time`. Note that you could specify species names with these words contained within the name, for example, `nullaminoacids` or `nullnucleotides`.
- If you are using a species name that is not a valid MATLAB variable name, do the following:
 - Enclose the name in square brackets when writing a reaction rate equation or a rule. For example, enclose `[DNA polymerase+]` within brackets in reaction rates and rules.
 - Enter the name without brackets when you are creating the species or when you are adding the reaction. For example, enter `DNA polymerase+` when specifying the name of the species or while writing the reaction.

addspecies (model)

- Name cannot be the characters `i`, or `j`, or contain `->`, `<`, `>`, `[`, or `]`. The names `i` and `j` are reserved MATLAB characters. Because expressions in abstract kinetic laws, reaction rates, and rules are considered to be MATLAB code, SimBiology evaluates `i` or `j` as an imaginary number and not as the value of species `i` or `j`. For example, an expression $V \cdot S^i / K$ is interpreted to have three names, `V`, `S`, and `K`, instead of four. Use brackets to protect such variables.
- If a variable in a reaction rate equation or rule has the same name as a MATLAB function, SimBiology evaluates the expression as a call to the MATLAB function. In general, when creating variable names, you should avoid using MATLAB function names or variable names that are invalid in MATLAB.

If you change the Name 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.

SimBiology automatically updates species names 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

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, one is a reactant and the other is the enzyme catalyzing the reaction.

- 1 Create a model object with the name `my_model`.

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

- 2 Add two species objects with the names `glucose_6_phosphate` and `glucose_6_phosphate_dehydrogenase`.

```
speciesObj1 = addspecies (modelObj, 'glucose_6_phosphate');  
speciesObj2 = addspecies (modelObj, ...
```

addspecies (model)

```
'glucose_6_phosphate_dehydrogenase');
```

- 3** Set initial amount of glucose_6_phosphate to 100 and verify.

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

MATLAB returns

```
ans =  
  
100
```

- 4** Use get to note that modelObj contains the species object array.

```
get(modelObj, 'Species')
```

MATLAB returns,

Species Object Array

Index:	Name:	Initial Amount:	Initial AmountUnits:
1	glucose_6_phosphate	100	
2	glucose_6_phosphate_dehydrogenase	0	

- 5** Retrieve information about the first species in the array.

```
get(modelObj.Species(1))
```

```
Annotation: ''  
ConstantAmount: 'false'  
InitialAmount: 100  
InitialAmountUnits: ''  
Name: 'glucose_6_phosphate'
```

```
Notes: ''
Parent: [1x1 SimBiology.Model]
Tag: ''
Type: 'species'
UserData: []
```

See Also

addproduct, addreactant, addreaction

MATLAB functions– get and set

Configset 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

See “Property Summary” on page 4-47 for links to configset object property reference pages.

Properties define the characteristics of an object. Use the MATLAB `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	<code>addconfigset (model)</code>	Create configuration set 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

Property Summary

Active	Indicate object 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

Configset object

See Also

AbstractKineticLaw object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object

Purpose Copy SimBiology object and its children

Syntax

```
copiedObj = copyobj(Obj, parentObj)  
copiedObj = copyobj(modelObj)
```

Arguments

<i>Obj</i>	Abstract kinetic law, configuration set, kinetic law, model, parameter, reaction, rule, or species object.
<i>parentObj</i>	If <i>copiedObj</i> is configuration set, reaction, rule or species object, <i>parentObj</i> must be a model object. If <i>copiedObj</i> is a parameter object, <i>ParentObj</i> must be a model or kinetic law object. If <i>copiedObj</i> is a model object, <i>ParentObj</i> must be a model object (for example, in the case of submodels) or sbioroot.
<i>modelObj</i>	Model object to be copied.

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.

- *Obj*— Can be abstract kinetic law, configuration set, kinetic law, model, parameter, reaction, rule, or species object.
- *parentObj* — If *copiedObj* is configuration set, reaction, rule or species object, *parentObj* must be a model object. If *copiedObj* is a parameter object, *ParentObj* must be a model or kinetic law object. If *copiedObj* is a model object, *ParentObj* must be a model object (for example, in the case of submodels) or sbioroot.

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.

copyobj (any object)

Example

Create a reaction object separate from a model object and then add it to a model.

- 1 Create a model object and create a separate reaction object.

```
modelObj = sbiomodel('cell');  
reactionObj = sbioreaction('a -> b');
```

- 2 Create a copy of the reaction object and assign it to the model object.

```
reactionObjCopy = copyobj(reactionObj, modelObj);  
modelObj.Reactions
```

```
Reaction Object Array  
Index:    Reaction:  
1         a -> b
```

See Also

`sbiomodel`, `sbioreaction`, `sbiroot`

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, 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`

Purpose Display summary of SimBiology object

Syntax `display(Obj)`

Arguments

Obj SimBiology object: abstract kinetic law, configuration set, kinetic law, model, parameter, reaction, rule, or species.

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 *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')
```

getadjacencymatrix (model)

Purpose Get adjacency matrix from model object

Syntax

```
M = getadjacencymatrix(mode1Obj)
M = getadjacencymatrix(mode1Obj, 'flat')
[M, Headings] = getadjacencymatrix(mode1Obj)
[M, Headings, Mask]=getadjacencymatrix(mode1Obj)
```

Arguments

<i>M</i>	Adjacency matrix for <i>mode1Obj</i>
<i>mode1Obj</i>	specify model object <i>mode1Obj</i>
'flat'	Return adjacency matrix for only specified <i>mode1Obj</i> not for objects contained in the <i>mode1Obj</i>
<i>Headings</i>	Return row and column headings to <i>Headings</i>
<i>Mask</i>	Returns 1 for species object 0 for reaction object to <i>Mask</i>

Description getadjacencymatrix returns adjacency matrix for model object.

M = getadjacencymatrix(*mode1Obj*) returns adjacency matrix for model object, (*mode1OBJ*) to *M*.

An adjacency matrix is defined by listing all species contained by *mode1Obj* and all reactions contained by *mode1Obj* 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(*mode1Obj*, 'flat') returns the adjacency matrix to *M* and defines the adjacency matrix for only *mode1Obj*. If *mode1Obj* is a SimBiology model then *M* is the adjacency matrix for the reactions and species contained by *mode1Obj*. *M* does not include any submodel reaction or species information.

`[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*. In the above example, *Headings* would be {'A', 'B', 'C', 'R1'}.

`[M, Headings, Mask]=getadjacencymatrix(modelObj)` returns an array of ones and zeros to *Mask* where a 1 represents a species object and a 0 represents a reaction object. In the above example, *Mask* would be [1 1 1 0].

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

```
getConfigsetObj = getConfigset(modelObj, 'NameValue')  
getConfigsetObj = getConfigset(modelObj)  
getConfigsetObj = getConfigset(modelObj, 'active')
```

Arguments

<i>modelObj</i>	Model object. Enter a variable name for a model object.
<i>NameValue</i>	Name of the getConfigset object.
<i>getConfigsetObj</i>	Object holding the simulation specific information.

Description

getConfigsetObj = getConfigset(*modelObj*, 'NameValue') returns the configuration set attached to *modelObj* that is named *NameValue*, to *getConfigsetObj*.

getConfigsetObj = getConfigset(*modelObj*) returns a vector of all attached configuration sets, to *getConfigsetObj*.

getConfigsetObj = getConfigset(*modelObj*, 'active') retrieves the active configuration set.

A configuration set object stores simulation specific information. A SimBiology model can contain multiple getConfigsets with one being active at any given time. The active configuration set contains the settings that are used during the simulation.

Use the setActivegetConfigset function to define the active getConfigset. *modelObj* always contains at least one getConfigset object with name configured to 'default'. Additional getConfigset objects can be added to *modelObj* with the method, addgetConfigset .

Example

1 Retrieve the defaultgetConfigset object from the *modelObj*.

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

getparameters (kineticlaw)

Purpose Get specific parameters in kinetic law object

Syntax

```
parameterObj = getparameters(kineticlawObj)
parameterObj = getparameters(kineticlawObj,
    'ParameterVariablesValue')
```

Arguments

<i>kineticlawObj</i>	Retrieve parameters used by kinetic law object.
<i>ParameterVariablesValue</i>	Retrieve parameters used by kinetic law object corresponding to the specified parameter in 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.

Example

Create a model, add a reaction and assign the ParameterVariableNames 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** 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 (V_m and K_m) 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

Parameter Object Array

Index:	Name:	Value:	ValueUnits:
1	Va	1	

```
parameterObj4 = getparameters (kineticlawObj, 'Km')
```

See Also

addparameter, getspecies, setparameter

getspecies (kineticlaw)

Purpose Get specific species in kinetic law object

Syntax

```
speciesObj = getspecies(kineticlawObj)
speciesObj = getspecies(kineticlawObj,
    'SpeciesVariablesValue')
```

Arguments

<i>kineticlawObj</i>	Retrieve species used by kinetic law object.
<i>SpeciesVariablesValue</i>	Retrieve species used by 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 SimBiology updates the reaction 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.

Example

Create a model, then add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

- 1 Create model object, 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

Species Object Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	a	0	

See Also

addspecies, setspecies, getparameters, setparameter

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

<i>M</i>	Adjacency matrix for <i>modelObj</i> .
<i>modelObj</i>	Specify model object <i>modelObj</i> .
'flat'	Return stoichiometry matrix for only specified <i>modelObj</i> not for objects contained in the <i>Obj</i> .
<i>objSpecies</i>	Return list of <i>Obj</i> species by Name property of species.
<i>objReactions</i>	Return list of <i>Obj</i> reactions by Name property of reactions.

Description

getstoichmatrix returns a stoichiometry matrix for a model object.

M = getstoichmatrix(*modelObj*) returns a stoichiometry matrix for SimBiology a 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`. If `Obj` is a SimBiology model then `M` is the stoichiometry matrix for the reactions and species contained by `modelObj`. `M` does not include any submodel reaction or species information.

`[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 `Obj`. In the above example, `ObjReactions` would be {'R1', 'R2'}.

Example

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`

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, SimBiology 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. SimBiology 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. SimBiology 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.

SimBiology substitutes in the expression, the names of parameter variables and species variables in the order specified in the ParameterVariables and SpeciesVariables properties respectively.

SimBiology 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 4-69

Properties define the characteristics of an object. Use the MATLAB `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 4-65.

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:

$$V_{\max} * S / (K_m + S)$$

In SimBiology Henri-Michaelis-Menten is a built-in abstract kinetic law, where V_{\max} 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 Getting Started with SimBiology 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
and S = A
```

KineticLaw object

Define V_a and K_a in the `ParameterVariableNames` property to substitute the variables that are in the `ParameterVariables` property (V_{max} and K_m). 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'});
```

SimBiology now assigns the rate equation in the reaction object as follows:

$$V_a * A / K_a + A$$

For a detailed procedure, see “Examples” on page 4-70.

The following table below 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: $V_{\max} * S / (K_m + S)$	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.
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.

KineticLaw object

Property	Property Purpose	Abstract Kinetic Law Object	Kinetic Law Object
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

<code>display (any object)</code>	Display summary of SimBiology object
<code>getparameters (kineticlaw)</code>	Get specific parameters in kinetic law object
<code>getspecies (kineticlaw)</code>	Get specific species in kinetic law object
<code>setParameter (kineticlaw)</code>	Specify specific parameters in kinetic law object
<code>setspecies (kineticlaw)</code>	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
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

KineticLaw object

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 in SimBiology.

- 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 Vmax and Km. To set these variables, first create the parameter variables as parameter objects

(parameterObj1, parameterObj2) with names Va, Ka, and 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 components are species, reactions, parameters, rules, and submodels. 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 4-73 for links to model property reference pages.

Properties define the characteristics of an object. For example, a model object includes properties for species, reactions, parameters, rules, and submodels. Use the MATLAB `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 top-level SimBiology model object has its `Parent` property set to the SimBiology root object. Submodels have the `Parent` property set to another SimBiology model and are not stored by the SimBiology root object.

Constructor Summary

<code>addmodel (model)</code>	Add submodel object to model object
<code>sbiomodel</code>	Construct model object

Method Summary

<code>addconfigset (model)</code>	Create configuration set object and add to model object
<code>addmodel (model)</code>	Add submodel object to model object

<code>addparameter (model, kineticlaw)</code>	Create parameter object and add to model or kinetic law object
<code>addreaction (model)</code>	Create reaction object and add to model object
<code>addrule (model)</code>	Create rule object and add to model object
<code>addspecies (model)</code>	Create species object and add to model object
<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object
<code>getadjacencymatrix (model)</code>	Get adjacency matrix from model object
<code>getConfigset (model)</code>	Get configuration set object from model object
<code>getstoichmatrix (model)</code>	Get stoichiometry matrix from model object
<code>removeconfigset (model)</code>	Remove configuration set from model
<code>setactiveconfigset (model)</code>	Set the active configuration set for model object
<code>verify (model)</code>	Validate and verify SimBiology model

Property Summary

Annotation	Store link to URL or file
Models	Contains all model objects

Model object

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
Species	Array of species 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. In 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 its submodels 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 4-76 for links to parameter object property reference pages.

Properties define the characteristics of an object. Use the MATLAB `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

<code>addparameter (model, kineticlaw)</code>	Create parameter object and add to model or kinetic law object
<code>sbioparameter</code>	Construct parameter object

Method Summary

<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object

Parameter object

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 objectKineticLaw object, Model object, Reaction object, Root object, Rule object, Species 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 + phosphocreatine
```

```
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 4-78 for links to reaction object property reference pages.

Properties define the characteristics of an object. Use the MATLAB `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

<code>addreaction (model)</code>	Create reaction object and add to model object
<code>sbioreaction</code>	Construct reaction object

Method Summary

<code>addkineticlaw (reaction)</code>	Create kinetic law object and add to reaction object
<code>addproduct (reaction)</code>	Add product species object to reaction object
<code>addreactant (reaction)</code>	Add species object as a reactant to reaction object
<code>copyobj (any object)</code>	Copy SimBiology object and its children

Reaction object

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
rmproduct (reaction)	Remove species object from reaction object products
rmreactant (reaction)	Remove species object from reaction object reactants

Property Summary

Active	Indicate object 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 a reaction is reversible or irreversible
Stoichiometry	Species coefficients in a reaction
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 objectKineticLaw 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

<code>modelObj</code>	Model object from which to remove configuration set.
<code>NameValue</code>	Name of the configuration set.
<code>configsetObj</code>	Configuration set object that is to be removed from model object

Description

`removeconfigset(modelObj, 'NameValue')` removes the configset object with name, `NameValue` from 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 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.

Example

- 1 Create a model object by importing the file `oscillator.xml` and add a configset.

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

```
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

reset (root)

Purpose Delete all model objects from the 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 the top-level SimBiology model objects, available units, unit prefixes, and abstract kinetic law objects. A top-level SimBiology model object has its Parent property set to the SimBiology root object. A SimBiology model object that has its Parent property set to another SimBiology model is a submodel and is not stored by the SimBiology root.

To add an abstract kinetic law to the SimBiology root user-defined library, use the `addtolibrary` 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` that 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

`sbioregisterunit`, `sbioregisterunitprefix`, `sbioroot`, `sbioreset`,
`sbiohelp`

rmproduct (reaction)

Purpose Remove species object from reaction object products

Syntax `rmproduct(reactionObj, SpeciesName)`
`rmproduct(reactionObj, speciesObj)`

Arguments

<i>reactionObj</i>	Reaction object.
<i>SpeciesName</i>	Name for a model object. Enter a species name or cell array of species names.
<i>speciesObj</i>	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 will be returned.

Examples

Example 1

Shows you how to remove a product 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 -> creatine + ATP + Pi');  
rmproduct(reactionObj, 'Pi')
```

Reaction Object 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

rmreactant (reaction)

Purpose Remove species object from reaction object reactants

Syntax
`rmreactant(reactionObj, SpeciesName)`
`rmreactant(reactionObj, speciesObj)`

Arguments

<i>reactionObj</i>	Reaction object.
<i>SpeciesName</i>	Name for a species object. Enter a species name or cell array of species names.
<i>speciesObj</i>	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

Shows you 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')
```

Reaction Object 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(r, m.Species(2));
reactionObj.Reaction
```

```
ans =
      A -> C
```

See Also

[rmproduct](#), [delete](#)

Root object

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, available units, unit prefixes, and available abstract kinetic law objects.

The units define the set of core units and user-defined units. The abstract kinetic law objects define the built-in abstract kinetic law objects and user-defined abstract kinetic law objects.

You can retrieve top-level SimBiology model objects from the SimBiology root object. A top-level SimBiology model object has its Parent property set to the SimBiology root object. Submodels have the Parent property set to another SimBiology model and are not stored by the SimBiology root object.

See “Property Summary” on page 4-89 for links to root object property reference pages.

Properties define the characteristics of an object. Use the MATLAB 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
delete (any object)	Delete SimBiology object
reset (root)	Delete all model objects from the root object

Property Summary

BuiltInKineticLaws	Contain built-in kinetic laws
BuiltInUnitPrefixes	Contain built-in unit prefixes
BuiltInUnits	Contain built-in units
Models	Contains all model objects
Type	Display top-level SimBiology object type
UserDefinedKineticLaws	Contain user-defined kinetic laws
UserDefinedUnitPrefixes	Contain user-defined unit prefixes
UserDefinedUnits	Contain user-defined units

See Also

AbstractKineticLaw object, Configset objectKineticLaw object, Model object, Parameter object, Reaction object, Rule object, Species object

Rule 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. There are three types of rules in SimBiology:

- **Algebraic** — Algebraic rules are evaluated continuously during a simulation. An algebraic rule takes the form $0 = \text{Expression}$ and the rule is specified as the `Expression`. For example, to write a mass conservation expression such as `species_total = species1 + species2`, where `species_total` is the independent variable, write the rule as

```
species1 + species2 - species_total
```

- **Assignment** — Assignment rules are evaluated once at the beginning of a simulation. Assignment rules are expressed as `Variable = Expression`. For example, you could write an assignment rule to set the amount of `species1` to be proportional to `species2` as follows:

```
species1 = k/species2  
(where k is a known constant with units = concentration^2)
```

- **Rate** — Rate rules are evaluated continuously during a simulation. Rate rules are determined by $d\text{Variable}/dt = \text{Expression}$, which is expressed in SimBiology as `Variable = Expression`. For example, to define the rate of change in the quantity of `species3` ($dspecies3/dt$), write the rule in SimBiology as follows:

```
species3 = k * (species1 + species2)
```

One example case for a rate rule is when `Species1` is at the boundary of the system, but the rate of input of `species1` to the system can be determined by a rate rule.

Note If you plan to use a parameter in a rule, remember to add the parameter to the model object. Further, if you are using an algebraic or rate rule to vary the value of a parameter during the simulation, set the `ConstantValue` property to `false` for the parameter object.

See “Property Summary” on page 4-91 for links to rule property reference pages.

Properties define the characteristics of an object. Use the MATLAB `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

<code>addrule (model)</code>	Create rule object and add to model object
<code>sbiorule</code>	Construct rule object

Method Summary

<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object

Property Summary

<code>Active</code>	Indicate object use during simulation
<code>Annotation</code>	Store link to URL or file
<code>Name</code>	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 objectKineticLaw object, Model object, Parameter object, Reaction object, Root object, Species object

Purpose Set the 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]
```

setactiveconfigset (model)

```
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

Purpose

Specify specific parameters in kinetic law object

Syntax

```
setParameter(kineticlawObj, 'ParameterVariablesValue',  
            'ParameterVariableNamesValue')
```

Arguments

<i>ParameterVariableValue</i>	Specify value of parameter variable in kinetic law object.
<i>ParameterVariableNamesValue</i>	Specify the parameter name with which to configure parameter variable in kinetic law object. Determines parameters in ReactionRate equation.

Description

Configure ParameterVariableNames in 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 *kineticlawObj* ParameterVariableNames 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.

Example

Create a model, add a reaction, and assign the ParameterVariableNames 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');
```

setparameter (kineticlaw)

- 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

Purpose Specify species in kinetic law object

Syntax `setspecies(kineticlawObj, 'SpeciesVariablesValue',
'SpeciesVariableNamesValue')`

Arguments

<i>SpeciesVariablesValue</i>	Specify species variable in kinetic law object.
<i>SpeciesVariableNamesValue</i>	Specify the species name with which to configure species variable in kinetic law object. Determines species in ReactionRate equation

Description

`setspecies` configures 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 `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`.

Example

Create a model, add a reaction and assign the `SpeciesVariableNames` 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');
```

setspecies (kineticlaw)

- 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

Purpose

Options for model 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. SimBiology creates the necessary species objects.

Alternatively, create and add a species object to a model, using the `addspecies` method at the command-line. In the SimBiology desktop, you can add a species in the **Species** pane. In the **Project Explorer**, double-click **Species** to open the **Species** pane.

See “Property Summary” on page 4-100 for links to species property reference pages. Properties define the characteristics of an object. Use the MATLAB `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

<code>addspecies (model)</code>	Create species object and add to model object
<code>sbiospecies</code>	Construct species object

Method Summary

Methods for species objects

<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object
<code>display (any object)</code>	Display summary of SimBiology object

Species object

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


Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object

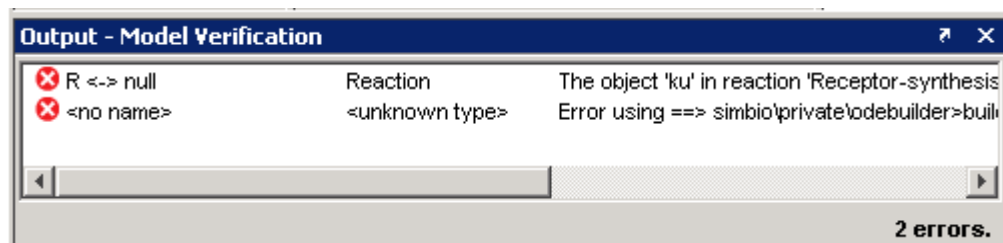
Purpose Validate and verify SimBiology model

Syntax `verify(modelObj)`

Description `verify(modelObj)` performs checks on a model object (*modelObj*) to verify that you can simulate the model. SimBiology generates stacked errors and warnings if any problems are found. To see the entire list of errors and warnings, use `sbiolasterror` and `sbiolastwarning`.

Verification in the SimBiology GUI

While you are building your model in the SimBiology desktop you can click  at any time to generate a list of any errors and warnings in the model. The errors and warnings appear in the **Output** pane. Following is an example of the error generated when the reaction rate of a reaction is set to a parameter that you have not defined in SimBiology.



Double-click the error row to move to the location of the error.

Examples

```
m = sbmlimport('radiodecay.xml');  
verify(m);
```

See Also `sbiolasterror`, `sbiolastwarning`

Properties — By Category

Abstract Kinetic Law (p. 5-2)	Properties for abstract kinetic law objects
Configuration Sets (p. 5-3)	Properties for configuration set objects
Kinetic Laws (p. 5-4)	Properties for kinetic law objects
Models (p. 5-5)	Properties for model objects
Parameters (p. 5-6)	Properties for parameter objects
Reactions (p. 5-7)	Properties for reaction objects
Root (p. 5-8)	Properties for the root object
Rules (p. 5-9)	Properties for rule objects
Species (p. 5-10)	Properties for species objects
Using Object Properties (p. 5-11)	Command-line syntax for entering and retrieving property values

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

Configuration Sets

Active	Indicate object 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

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
Models	Contains 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
Species	Array of species 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

Reactions

Active	Indicate object 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 a reaction is reversible or irreversible
Stoichiometry	Species coefficients in a reaction
Tag	Specify label for SimBiology object
Type	Display top-level SimBiology object type
UserData	Specify data to associate with object

Root

BuiltInKineticLaws	Contain built-in kinetic laws
BuiltInUnitPrefixes	Contain built-in unit prefixes
BuiltInUnits	Contain built-in units
Models	Contains all model objects
Type	Display top-level SimBiology object type
UserDefinedKineticLaws	Contain user-defined kinetic laws
UserDefinedUnitPrefixes	Contain user-defined unit prefixes
UserDefinedUnits	Contain user-defined units

Rules

Active	Indicate object 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

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

Using Object Properties

Command-line syntax for entering and retrieving property values.

Entering Property Values (p. 5-11)	Use either MATLAB functions or object dot notation to enter or change property values
Retrieving Property Values (p. 5-11)	Use either MATLAB functions or object dot notation to get property values
Help for Objects, Methods, and Properties (p. 5-12)	Use the command <code>sbiohelp</code> to get information about properties

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 i^{th} state vector y . Simulation converges at that time step if e_i satisfies the following equation:

$$|e_i| \leq \max(\text{RelativeTolerance} * |y_i|, \text{AbsoluteTolerance})$$

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
Access	Read/Write

Example

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 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. When a reaction or rule object Active property is set to be false, the simulation does not include the reaction or rule. This is a convenient way to test a model with and without a reaction or rule. For configset object, use the method `setactiveconfigset`, to set the object Active property to true.

Characteristics

Applies to	Objects: configset, reaction, rule
Data type	boolean
Data values	true or false. Default value is true. For default configset object default is true, for added configset object default is false.
Access	Read/Write

Example

1 Create a model object.

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

2 Add 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 Active property to 'false' and verify.

```
set (reactionObj, 'Active', false);  
get (reactionObj, 'Active')
```

MATLAB returns

```
ans =  
  
0
```

See Also

[addreaction](#), [addrule](#), [setactiveconfigset](#), [addconfigset](#)

Annotation

Purpose Store link to URL or file

Description The Annotation property stores URL or filename linking to information about a model.

Characteristics

Applies to	Objects: abstract kinetic law, kinetic law, model, parameter, reaction, root, rule, species
Data type	char string, URL
Data values	Character string with a directory path and filename or a URL.
Access	Read/Write

Example

1 Create a model object

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

2 Set annotation for model object

```
set (modelObj, 'annotation', 'www.reactome.org')
```

3 Verify the assignment.

```
get (modelObj, 'annotation')
```

MATLAB returns

```
ans =
```

```
www.reactome.org
```

See Also

`sbiomodel`, `addkineticlaw`, `addparameter`, `addreaction`, `addrule`, `addspecies`, `sbioroot`

Purpose

Indicate species boundary condition

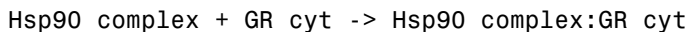
Description

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. In SimBiology, all species are state variables regardless of BoundaryCondition or ConstantAmount property.

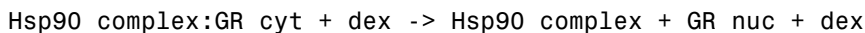
By default BoundaryCondition is false and SimBiology uses reaction rate equations to determine the rate of change of a species quantity in the model. Boundary condition is used when a species is modeled as a participant of reactions but the species quantity is not determined by a reaction rate equation. 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,



In response to the synthetic hormone dexamethasone (dex), GR moves from the cytoplasm to the nucleus.



For dex,

BoundaryCondition

```
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 SimBiology 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

Example

- 1 Create a model object

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


- 2 Add a species object and verify that boundary condition property setting is 'false' or 0.

```
speciesObj = addspecies(modelObj, 'glucose');  
get(speciesObj, 'BoundaryCondition')
```

MATLAB returns

```
ans =  
  
0
```

- 3 Set 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

Description BuiltInKineticLaws is a SimBiology root object property showing all abstract kinetic laws that are shipped with SimBiology. 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 6-27 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

Examples

Example 1

This example uses the command `sbiowhos` to show the current list of built-in kinetic laws.

```
sbiowhos -builtin -kineticlaw
```

```
Abstract Kinetic Law Object Array
```

Index:	Library:	Name:	Expression:
1	BuiltIn	Unknown	Unknown
2	BuiltIn	MassAction	MassAction

3	BuiltIn	Henri-Michaelis-Menten	$V_m * S / (K_m + S)$
---	---------	------------------------	-----------------------

Example 2

This example shows the current list of built-in kinetic laws by accessing the root object.

```
rootObj = sbioroot;  
get(rootObj, 'BuiltInKineticLaws')
```

Abstract Kinetic Law Object Array

Index:	Library:	Name:	Expression:
1	BuiltIn	Unknown	Unknown
2	BuiltIn	MassAction	MassAction
3	BuiltIn	Henri-Michaelis-Menten	$V_m * S / (K_m + S)$

See Also

UserDefinedKineticLaws, BuiltInUnits, BuiltInUnitPrefixes
MATLAB functions `get` and `set`

BuiltInUnitPrefixes

Purpose Contain built-in unit prefixes

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, because, SimBiology enables you 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. Both methods are illustrated in the examples below.

You cannot add, modify, or delete BuiltInUnitsPrefixes.

Characteristics

Applies to	Object: root
Data type	char string
Data values	Valid units
Access	Read-only

Examples

Example 1

This example uses the command `sbiowhos` to show the current list of built-in unit prefixes.

```
sbiowhos -builtin -unitprefix
```

Example 2

This example shows the current list of built-in unit prefixes by accessing the root object.

```
rootObj = sbioroot;  
get(rootObj, 'BuiltInUnitPrefixes')
```

See Also

BuiltInUnits, UserDefinedUnits, BuiltInKineticLaws
MATLAB functions get and set.

BuiltInUnits

Purpose Contain built-in units

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, because, SimBiology enables you 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. Both methods are illustrated in the examples below.

You cannot add, modify, or delete BuiltInUnits.

Characteristics

Applies to	Object: root
Data type	char string
Data values	Valid units.
Access	Read-only

Examples

Example 1

This example shows the current list of built-in units using the `sbiowhos` command.

```
sbiowhos -builtin -units
```

Example 2

This example shows the current list of built-in units by accessing the root object.

```
rootObj = sbioroot;  
get(rootObj, 'BuiltInUnits')
```

See Also

BuiltInUnitPrefixes, UserDefinedUnits, BuiltInKineticLaws

MATLAB functions get and set.

CompileOptions

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	DimensionalAnalysis	Perform dimensional analysis on model
	Type	Display top-level SimBiology object type
	UnitConversion	Perform unit conversion

Characteristics

Applies to	Object: configset object
Data type	Object
Data values	Compile time options
Access	Read-only

Example

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');
```

```
DimensionalAnalysis: 1
```


Type: 'compileoptions'
UnitConversion: 1

See Also MATLAB functions `get`, `set`

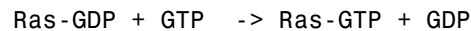
ConstantAmount

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 following is 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.



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*.

The property ConstantAmount is for species objects; the property ConstantValue is for parameter objects.

Characteristics

Applies to	Object: species
Data type	boolean
Data values	true or false. The default value is false.
Access	Read/Write

Example 1 Create a model object with name my_model.

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

2 Add a species object and verify that the ConstantAmount property setting is 'false' or 0

```
speciesObj = addspecies (modelObj, 'glucose');  
get (speciesObj, 'ConstantAmount')
```

MATLAB returns

```
ans =
```

```
0
```

3 Set constant amount to 'true' and verify

```
set (speciesObj, 'ConstantAmount', true);  
get (speciesObj, 'ConstantAmount')
```

MATLAB returns

```
ans =
```

```
1
```

See Also

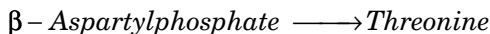
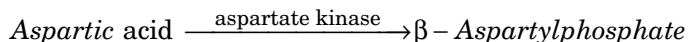
`addspecies`, `BoundaryCondition`

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. For 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:



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`.

$$k_aspartate_kinase - (1/threonine)$$

The property ConstantValue is for parameter objects; the property ConstantAmount is for species objects.

Characteristics

Applies to	Object: parameter
Data type	boolean
Data values	true or false. Default value is 'true'.
Access	Read/Write

Example

- 1 Create a model object.

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

- 2 Add 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`

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, SimBiology checks whether the physical quantities of the units involved in reactions and rules, match and are applicable.

For example, consider a reaction $a + b \rightarrow c$. Using mass action kinetics, the reaction rate is defined as $a \cdot b \cdot k$ 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 $1 / (M \cdot \text{second})$. If you specify k with another equivalent unit definition, for example $1 / [(\text{moles/liter}) \cdot \text{second}]$, DimensionalAnalysis checks whether the physical quantities match. If the physical quantities do not match, you see an error and the model is not simulated, UnitConversion is the next step after DimensionalAnalysis.

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

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.

DimensionalAnalysis

```
set(optionsObj, 'DimensionalAnalysis' false)
```

See Also

getconfigset, sbiosimulate
MATLAB functions get and set.

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 (τ) 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 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; default is 3e-2
Access	Read/Write

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

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 builtin 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** 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** $V_m * [S] / (K_m + [S])$, where V_m and K_m 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.

Expression

By applying the abstract kinetic law Henri-Michaelis-Menten to a reaction $A \rightarrow B$ with V_a mapping to V_m and A mapping to S the rate equation for the reaction becomes $V_a * [A] / (K_a + [A])$.

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

$$r = k \prod_{i=1}^{n_r} [S_i]^{m_i}$$

where $[S_i]$ is the concentration of the i^{th} reactant, m_i is the stoichiometric coefficient of $[S_i]$, n_r is the number of reactants and k is the mass action reaction rate constant.

SimBiology 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: kineticlaw, abstract 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 with Henri-Michaelis-Menten kinetics

- 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 reactionObj is 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

Expression

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

Example with Mass Action kinetics.

- 1 Create a model object, 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
```

See Also

Abstract and kinetic law object properties: [SpeciesVariables](#), [ParameterVariables](#)

Kinetic law object properties: [KineticLawName](#), [Parameters](#), [SpeciesVariableNames](#), [ParameterVariableNames](#)

Reaction object property: ReactionRate

Functions: sbioaddtolibrary, sbiowhos

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

Example

Add a species to a model and set the initial amount of the species.

1 Create a model object with named `my_model`.

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

2 Add the species object with the name `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 builtin units. To get a list of the defined units use the `sbioshowunits` function. If `InitialAmountUnits` changes from one unit definition to another, the `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`.

Characteristics

Applies to	object: species
Data type	char string
Data values	unit from Units list; Default = '' (None)
Access	Read/Write

Example

1 Create a model object named `my_model`.

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

2 Add a species object with the name `glucose`.

```
speciesObj = addspecies (modelObj, 'glucose');
```

3 Set the initial amount to 100, `InitialAmountUnits` to `molecule`, and verify.

```
set (speciesObj, 'InitialAmount',100, ...  
    'InitialAmountUnits', 'molecule');  
get (speciesObj,'InitialAmountUnits')
```

MATLAB returns

```
ans =
```

InitialAmountUnits

molecule

See Also

InitialAmount, sbioshowunits, sbioconvertunits,
sbioregisterunit

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

Example

Example with Henri-Michaelis-Menten kinetics

1 Create a model object, 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

Kinetic Law Object Array

Index:	KineticLawName:
1	Henri-Michaelis-Menten

See Also

Kinetic law object properties: KineticLawName, Parameters, SpeciesVariableNames, ParameterVariableNames

Reaction object property: ReactionRate

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 builtin or user-defined abstract kinetic law library. See “Abstract Kinetic Law” on page 6-27 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: <code>kineticlaw</code>
Data type	<code>char</code> string
Data values	<code>char</code> 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 `KineticLawName` of `kineticlawObj`

```
get (kineticlawObj, 'KineticLawName')
```

MATLAB returns

```
ans =
```

KineticLawName

Henri-Michaelis-Menten

See Also

Abstract and kinetic law object properties: Expression, SpeciesVariables, ParameterVariables

Kinetic law object properties: Parameters, SpeciesVariableNames, ParameterVariableNames

Reaction object property: ReactionRate

Functions: sbioaddtolibrary, sbiowhos

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

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')
```

LogDecimation

ans =

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 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

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 []
Access	Read/Write

See Also SimBiology property `RelativeTolerance`
MATLAB function `odeset`

Models

Purpose Contains all model objects

Description The `Model` 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 removed 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 (<code>[]</code>).
Access	Read-only

Example

1 Create a model object

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

2 Add submodels to model object and verify

```
submodelObj1 = addmodel (modelObj, 'nucleus');  
submodelObj2 = addmodel (modelObj, 'mitochondrion');
```

```
get (modelObj, 'Models')
```

MATLAB returns

SimBiology Model Object Array

Index:	Name:
1	nucleus
2	mitochondrion

See Also

`sbiomodel`, `addmodel`

Name

Purpose

Specify name of object

Description

The property Name identifies a SimBiology object. Species, parameter, and model objects can be referenced by other objects using the Name property, therefore Name must be unique for these objects.

Use the function `sbioselect` to find an object with the same Name property value.

There are reserved characters that cannot be used in object names:

- Models cannot have an empty in Name.
- Species 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 `i`, `j`, `->` `<` `>`,`[`, and `]`.
 - If you are using a species name that is not a valid MATLAB variable name, do the following:
 - Enclose the name in square brackets when writing a reaction rate equation or a rule.
 - Enter the name without brackets when you are creating the species or when you are adding the reaction.For example, enclose `[DNA polymerase+]` within brackets in reaction rates and rules; enter `DNA polymerase+` when specifying the name of the species or while writing the reaction. For more information on valid MATLAB variable names see `genvarname`.
 - The literal words `null` and `time`. Note that you could specify species names with these words contained within the name. For example `nullaminoacids`, or `nullnucleotides` .
 - The characters `i`, `j`, `->` `<` `>`,`[`, and `]`.

- Parameters cannot have an empty Name or have the name time.

Characteristics

Applies to	Objects: kineticlaw, model, parameter, reaction, rule, species
Data type	char string
Data values	Any char string except reserved words and characters.
Access	Read/Write

Example

- 1 Create a model object with the name my_model.

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

- 2 Add a reaction object to the model object

```
reactionObj = addreaction(modelObj, 'Aspartic acid -> beta-Aspartyl-P04')
```

MATLAB returns

Reaction Object Array

```
Index:    Reaction:
1         Aspartic acid -> beta-Aspartyl-P04
```

- 3 Set reaction Name and verify

```
set (reactionObj, 'Name', 'Aspartate kinase reaction');
get (reactionObj, 'Name')
```

MATLAB returns

```
ans =

Aspartate kinase reaction
```

Name

See Also

addkineticlaw, addmodel, addparameter, addreaction, addrule,
addspecies, sbiomodel

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.

$$dx(t) / dk$$

- 'Half' specifies normalization relative to the numerator (species quantity) only.

$$(1/x(t)) (dx(t) / dk)$$

- 'Full' specifies that the data should be made dimensionless.

$$(k/x(t)) (dx(t) / dk)$$

Characteristics

Applies to	Object: <code>SensitivityAnalysisOptions</code>
Data type	enum
Data values	'None', 'Half', 'Full'. Default is 'None'.
Access	Read/Write

See Also

`ParameterInputFactors`, `SensitivityAnalysis`,
`SensitivityAnalysisOptions`, `SpeciesInputFactors`

Notes

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: kinetic law, model, parameter, reaction, rule, species
Data type	char string
Data values	Any char string
Access	Read/Write

Example

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

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, SimBiology 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: <code>SensitivityAnalysisOptions</code>
Data type	parameter object or array of parameter objects
Data values	Parameter object array. Default is <code>[]</code> .
Access	Read/Write

Examples

This example shows how to set `ParameterInputFactors` for sensitivity analysis.

- 1 Import the radio decay model from 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')
```

Parameter Object Array

Index:	Name:	Value:	ValueUnits:
1	c	0.5	1/second

See Also

SimBiology function `sbioselect`

SimBiology properties `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 all components within the model (including all rules, submodels 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: <code>model</code> , <code>kineticlaw</code>
Data type	array of parameter objects
Data values	Parameter objects; Default value is empty (<code>[]</code>).
Access	Read-only

Example

1 Create a model object, 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')
```

Parameter Object 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')
```

Parameter Object Array

Index:	Name:	Value:	ValueUnits:
1	K2	1	

See Also

addparameter, delete, sbioparameter
MATLAB functions get and 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` 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: <code>kineticlaw</code>
Data type	Cell array of strings
Data values	Cell array of parameters
Access	Read/Write

Example 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 (V_m and K_m) 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

Reaction object property: ReactionRate

Abstract kinetic law object and kinetic law object properties:

Expression, SpeciesVariables, ParameterVariables

Kinetic law object property: SpeciesVariableNames

Method: setparameter

Purpose Parameters in abstract kinetic law

Description **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.

Example

Create a model, add a reaction and assign the SpeciesVariableNames for the reaction rate equation.

- 1 Create a model object, 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.

ParameterVariables

```
get (kineticlawObj, 'ParameterVariables')
```

MATLAB returns

```
ans =
```

```
'Vm' 'Km'
```

See Also

Reaction object property: ReactionRate

Abstract kinetic law object and kinetic law object properties:
Expression, SpeciesVariables

Kinetic law object property: SpeciesVariableNames,
ParameterVariableNames

Method: setparameter

MATLAB function set

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.

- The top level model always has the SimBiology root as the Parent
- A model object can have another model object as Parent; this is the case for submodels.
- Reaction and species objects, are limited to a model object or [] as Parent.
- Parameter objects, are limited to a model object or a kinetic law object as Parent.
- Rule object, are limited to a model object or [] as Parent
- An abstract kinetic law object has [] as Parent until it has been added to the library, then has the SimBiology root as Parent

Characteristics

Applies to	Object: abstractkineticlaw, kineticlaw, model, parameter, reaction, rule, species
Data type	Object
Data values	SimBiology component object or empty []. Default value is run-time.
Access	Read-only

See Also sbiomodel, addkineticlaw, addmodel, addparameter, addreaction

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 `functionset`.

Characteristics

Applies to	Object: reaction
Data type	Array of objects
Data values	Species objects. Default is [].
Access	Read-only

Example

1 Create a model object

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

2 Add reaction objects

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

3 Verify assignment.

```
productsObj = get(reactionObj, 'Products')
```

MATLAB returns

```
Species Object Array
```

```
Index:   Name:   InitialAmount:   InitialAmountUnits:
```

1	c	0
2	d	0

See Also `addkineticlaw, addspecies, addproduct, 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 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 resets the state at startup. The default value of RandomState is [].

Characteristics

Applies to	Object: SolverOptions for SSA, expltau, impltau
Data type	int
Data values	Default is [].
Access	Read/Write

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

Reactants

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	Objects: reaction
Data type	Species object or array of species objects
Data values	Species objects, default is []
Access	Read-only

Example

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

Species Object Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	a	0	

2 b 0

See Also addreaction, addspecies, addreactant, rmreactant

Reaction

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 Reactions.

If the Reaction property value is modified, the Species property value of the reaction object's parent is updated. If applicable, the Reactant and/or Product properties of the reaction object are also updated.

For example, if an additional species is added to the reaction, the species object is added to the model object Species property value. The species is also added to either the Reactant or Product property value. If a species is removed from a reaction, the species object is not removed from the Species property value. However, it is removed from the Reactant or Product property. The delete function can be used to remove the species object from the Species property value.

While the following are valid reactions,

```
A > null
null -> B
```

reactions that combine species with null are invalid.

```
A + null -> B
A -> B + null
```

Note the use of spaces around species names and stoichiometric values.

```
glucose + 2 ADP + 2 Pi -> 2 lactic acid + 2 ATP + 2 H2O
```

Characteristics

Applies to	Object: reaction
Data type	char string
Data values	Valid reaction string, default is ''
Access	Read/Write

Example

- 1 Create a model object, 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

sbireaction, addreaction

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 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.

Once you have specified the ReactionRate without KineticLaw, if 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 ''
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, 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 (V_m and K_m) and one species variable (S) that you should set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with a names V_{m_d}, K_{m_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, 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

[sbioreaction](#), [addreaction](#), [sbioparameter](#), [addparameter](#), [Reversible](#)

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	Objects: model
Data type	Array of reaction objects
Data values	Reaction object
Access	Read-only

Example

- 1 Create a model object, 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

Reaction Object Array

```
Index:    Reaction:
    1      a + b -> c + d
```

See Also

`sbireaction`, `addreaction`, `delete`

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 `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:

$$|e_i| \leq \max(\text{RelativeTolerance} * |y_i|, \text{AbsoluteTolerance})$$

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

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 a 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

Example

Create a model, add a reaction, and assign the expression for the reaction rate equation.

- 1 Create model object, 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) with names Kf, 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

sbioreaction, addreaction, addparameter, addreactant,
ParameterVariableNames, ReactionRate

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

Example

1 Create a model object, 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

Rule Object 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 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 `sbiorule`

The three rule types defined are algebraic, assignment, and rate:

- **Algebraic** — Algebraic rules are evaluated continuously during a simulation. An algebraic rule takes the form $0 = \text{Expression}$, and the rule is specified as the Expression. For example, a mass conservation expression such as `species_total = species1 + species2`, where `species_total` is the independent variable, would be written as

```
species1 + species2 - species_total
```

- **Assignment** — Assignment rules are evaluated once at the beginning of a simulation. Assignment rules are expressed as `Variable = Expression`. For example write an assignment rule to set the amount of `species1` to be proportional to `species2`;

```
species1 = k/species2  
(where k is a known constant with units = concentration^2)
```

- **Rate** — Rate rules are evaluated continuously during a simulation. Rate rules are determined by $d\text{Variable}/dt = \text{Expression}$, which is expressed in SimBiology as `Variable = Expression`. For example, write a rate rule to define the rate of change in the quantity of a new species, `species3`, using the expression.

```
dspecies1/dt = k * (species1 + species2)
```

RuleType

Write the rule in SimBiology as

```
species3 = k * (species1 + species2)
```

Characteristics

Applies to	Object: rule
Data type	char string
Data values	'algebraic', 'assignment', 'rate'. Default value is 'assignment'.
Access	Read/write

Example

- 1 Create a model object, 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 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 sbiorule, addrule, delete

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

Example

1 Create a model object, 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

Rule Object Array

Index:	RuleType:	Rule:
1	algebraic	10-a+b

See Also addrule, delete, sbiorule

RuntimeOptions

Purpose Options for logged species

Description The RuntimeOptions property holds options for species that will be logged during the simulation run. The runtime 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

Example

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 MATLAB functions `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').

While computing sensitivities, SimBiology uses the "complex-step approximation" to calculate derivatives of reaction rates. This technique yields accurate results for the vast majority of typical reaction kinetics, which involve only simple mathematical operations and functions. When a reaction rate involves a non-analytic function, this technique can lead to inaccurate results. An example of such a non-analytic function is the MATLAB function abs. If SimBiology sensitivity analysis gives questionable results on a model whose reaction rates contain unusual functions, you may be running into limitations of the complex-step method. Contact the MathWorks Technical Support group for additional information.

See SensitivityAnalysisOptions for more information on sensitivity analysis.

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')
```

```
ans =
```

```
on
```

See Also

SensitivityAnalysisOptions, SolverOptions, SolverType, StatesToLog

SensitivityAnalysisOptions

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 6-89.

When sensitivity analysis is enabled, the following command

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

returns [t,x,names], where

- t is a 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 a 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 non-constant 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.

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.

Property Summary

Normalization	Specify normalization type for sensitivity analysis
ParameterInputFactors	Specify parameter input factors for sensitivity analysis
SpeciesInputFactors	Specify species input factors for sensitivity analysis

Characteristics

Applies to	Object: configuration set object
Data type	Object
Data values	SensitivityAnalysisOptions properties as summarized in “Property Summary” on page 6-89 .
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')
```

SensitivityAnalysisOptions

Parameter Object 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')
```

Species Object Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	z	0	molecule

- 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 for more information.

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

- 7** Display 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

SimBiology 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 the property summary on this page.

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

Data values	Solver options depending on SolverType. Default is SolverOptions for default SolverType (ode15s).
Access	Read-only

Example

Illustrates 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: []
```

SolverOptions

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

Purpose Select solver type for simulation

Description The SolverType property let you specify the solver to use for a simulation. The valid SolverType values are 'ssa', 'expltau', 'impltau', 'ode45', 'ode23', 'ode113', 'ode15s', 'ode23s', and 'ode23t'. The default solver is ode15s. For a discussion about these solver types, see “Selecting a Solver”.

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'. Default is ode15s.
Access	Read/Write

Example 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
```

SolverType

```
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
MATLAB function set

Purpose

Array of species in model object

Description

The `Species` property indicates the species in a `Model` object. Read-only array of SimBiology species objects.

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:

Species names cannot be empty, and note the following reserved words, characters and constraints:

- The literal words `null` and `time`. Note that you could specify species names with these words contained within the name. For example `nullaminoacids`, or `nullnucleotides`.
- The characters `i`, `j`, `->` `<` `>`, `[`, and `]`.
- If you are using a species name that is not a valid MATLAB variable name, do the following:
 - Enclose the name in square brackets when writing a reaction rate equation or a rule.
 - Enter the name without brackets when you are creating the species or when you are adding the reaction.

For example, enclose `[DNA polymerase+]` within brackets in reaction rates and rules; enter `DNA polymerase+` when specifying the name of the species or while writing the reaction.

Species

Characteristics

Applies to	Object: model
Data type	Array of species objects
Data values	Species object, default is empty []
Access	Read-only

See Also

`sbiospecies`, `addreaction`, `addspecies`, `delete`

Purpose Specify species input factors for sensitivity analysis

Description SpeciesInputFactors is a property of the SensitivityAnalysisOptions object. SensitivityAnalysisOptions is a property of the configuration set object. Use SpeciesInputFactors to specify the species with respect to which you want to compute the sensitivities of the species states in your model. SimBiology 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, SimBiology 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 [].
Access	Read/Write

Examples This example shows how to set SpeciesInputFactors for sensitivity analysis.

- 1 Import the radio decay model from 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')
```

Species Object Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	z	0	molecule

See Also

SimBiology function `sbioselect`

SimBiology properties `SensitivityAnalysis`,
`SensitivityAnalysisOptions`, `ParameterInputFactors`

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 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

Example 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.

SpeciesVariableNames

- 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

Reaction object property: ReactionRate

Abstract kinetic law object and kinetic law object properties:

Expression, SpeciesVariables, ParameterVariables

Kinetic law object property: ParameterVariableNames

Method: setparameter

Purpose Species in abstract kinetic law

Description Property showing 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.

Example

Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

- 1 Create a model object, 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

Reaction object property: ReactionRate

Abstract kinetic law object and kinetic law object properties:
Expression, ParameterVariables

Kinetic law object property: ParameterVariableNames,
SpeciesVariableNames

Method: setparameter

MATLAB function set

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

Example

Illustrates 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 the `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')
```

StatesToLog

Species Object Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	pA	100	
2	pB	0	
3	pC	0	

Purpose Species coefficients in a 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 $3\text{H} + \text{A} \rightarrow 2\text{C} + \text{F}$ 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 $2\text{H} + \text{A} \rightarrow 2\text{C} + 3\text{F}$.

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` functions. 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 only support integers in Stoichiometry, therefore you must balance the reaction equation and specify integer values for these two cases.

$\text{A} \rightarrow \text{null}$ has a stoichiometry value of [-1]. $\text{null} \rightarrow \text{B}$ 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 [] (empty)
Access	Read/Write

Stoichiometry

Example

- 1 Create a reaction object

```
reactionObj = sbioreaction('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

`sbioreaction`, `addreaction`, `addproduct`, `addreactant`, `rmproduct`,
`rmreactant`, `Reaction`

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

Example

1 Retrieve the configset object from the 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

StopTimeType

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

Example 1 Retrieve the configset object from the modelObj.

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

2 Configure the StopTimeType to approxWallTime.

```
set(configsetObj, 'StopTimeType', 'approxWallTime')  
get(configsetObj, 'StopTimeType')
```

```
ans =
```

```
approxWallTime
```

See Also

StopTime, StatesToLog, TimeUnits

MATLAB function set

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 `sbioreselect` 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 `sbioreselect`. 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 `sbiodesktop`) 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, species
Data type	char string
Data values	Any char string
Access	Read/Write

Example

1 Create a model object.

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

2 Add reaction object and set Tag property to 'Synthesis Reaction'.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');  
set (reactionObj, 'Tag', 'Synthesis Reaction')
```

3 Verify Tag assignment.

```
get (reactionObj, 'Tag');
```

MATLAB returns

ans =

'Synthesis Reaction'

See Also

sbiomodel, sbioabstractkineticlaw, addkineticlaw, addparameter, addreaction, addrule, addspecies, sbioroot

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 seconds.

Characteristics

Applies to	Object: configset
Data type	string
Data values	Default value is second.
Access	Read-only

See Also StopTimeType, StopTime

Purpose Display top-level SimBiology object type

Description The Type property indicates a SimBiology object type. When you create an object in SimBiology, 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, SolverOptions.
Data type	char string
Data values	abstract_kinetic_law, configset, compileoptions, kineticlaw, parameter, reaction, root, rule, runtimeoptions, sbiomodel, species, 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, SimBiology 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 $a + b \rightarrow c$. 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`.

If `UnitConversion` fails, then you see an error when you simulate (`sbiosimulate`).

If `UnitConversion` is set to false, SimBiology uses the given object values.

Characteristics

Applies to	Object: <code>CompileOptions</code> (in <code>configset</code> object)
Data type	boolean
Data values	true or false. Default value is false.
Access	Read/Write

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
```

```
RuntimeOptions:
```

```
StatesToLog:     all
```

```
CompileOptions:
```

```
UnitConversion:  false
DimensionalAnalysis: true
```

3 Retrieve the CompileOptions object.

```
optionsObj = get(configsetObj, 'CompileOptions')
```

```
Compile Settings:
```

```
UnitConversion:  false
```

UnitConversion

```
DimensionalAnalysis: true
```

4 Assign a value of false to UnitConversion.

```
set(optionsObj, 'UnitConversion', true)
```

See Also

getConfigset, sbiosimulate.

MATLAB functions get and 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, kinetic law, model, parameter, reaction, rule, species
Data type	Any
Data values	Any. Default is empty
Access	Read/Write

See Also sbiomodel, sbioabstractkineticlaw, sbioparameter, sbioreaction, sbioroot, sbiorule, sbiospecies

UserDefinedKineticLaws

Purpose Contain user-defined kinetic laws

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 6-27 for a definition and more information.

Characteristics

Applies to	Object: root
Data type	char string
Data values	Valid kinetic laws
Access	Read/Write

Examples

Example 1

This example shows the current list of user-defined kinetic laws, using the command `sbiowhos`.

```
sbiowhos -userdefined -kineticlaw
```

```
Abstract Kinetic Law Object Array
```


Index:	Library:	Name:	Expression:
1	UserDefined	AKL1	S+P-S*P
2	UserDefined	AKL2	P+S*k
3	UserDefined	AKL3	P-S*k
4	UserDefined	AKL4	P*S*k

Example 2

This example shows the current list of user-defined kinetic laws by accessing the root object.

```
rootObj = sbioroot;  
get(rootObj, 'UserDefinedKineticLaws')
```

Abstract Kinetic Law Object Array

Index:	Library:	Name:	Expression:
1	UserDefined	AKL1	S+P-S*P
2	UserDefined	AKL2	P+S*k
3	UserDefined	AKL3	P-S*k
4	UserDefined	AKL4	P*S*k

Example 3

This example shows you how to add a user-defined kinetic law and how it is displayed in UserDefinedKineticLaws.

1 Create an abstract kinetic law.

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

2 Assign the parameter and species variables to the expression.

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

UserDefinedKineticLaws

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined
```

```
Abstract Kinetic Law Object Array
```

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

Alternatively,

```
rootObj = sbioroot;  
get(rootObj, 'UserDefinedKineticlaws')
```

```
Abstract Kinetic Law Object Array
```

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

See Also

`BuiltInKineticLaws`, `BuiltInUnits`, `BuiltInUnitPrefixes`

MATLAB functions `get` and `set`

Purpose Contain user-defined unit prefixes

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, because, SimBiology enables you 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

Example This example shows how to create a user-defined unit prefix and access it through the `UserDefinedUnitPrefixes` property.

- 1 Create a unit prefix with a multiplier of 10^{-5} (`sbioregisterunitprefix` requires you to specify the exponent).

```
sbioregisterunitprefix('peta', 15);
```

- 2 Display the unit prefix, using the command `sbiowhos`.

```
sbiowhos -userdefined -unitprefix
```

```
SimBiology UserDefined Unit Prefixes
```

UserDefinedUnitPrefixes

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

Alternatively, to display only names, use the following commands:

```
r = sbioroot
r.UserDefinedUnitPrefixes
```

```
ans =
```

```
    'peta'
```

See Also

BuiltInUnitPrefixes, BuiltInUnits, UserDefinedUnits,
UserDefinedKineticLaws

Purpose Contain user-defined units

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, because, SimBiology enables you 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

Example This example shows how to create a user-defined unit and access it through the UserDefinedUnits property.

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

```
sbioregisterunit('firstorderconstant', '1/second', 1);  
sbioregisterunit('secondorderconstant', '1/molecule*second', 1);
```

- 2 Display the unit, using the command `sbiowhos`.

```
sbiowhos -userdefined -unit
```

```
SimBiology UserDefined Units
```

UserDefinedUnits

Index:	Name:	Composition:	Multiplier:	Offset:
1	[1x18 char]	1/second	1.000000	0.000000
2	[1x19 char]	1/molecule*second	1.000000	0.000000

Alternatively, to display only names, use the following commands:

```
r = sbioroot
r.UserDefinedUnits
```

```
ans =
```

```
    'secondorderconstant'
    'firstorderconstant'
```

See Also

BuiltInUnitPrefixes, BuiltInUnits, UserDefinedUnitPrefixes,
UserDefinedKineticLaws

Purpose Assign value to parameter object

Description The Valueproperty 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

Example Assign a parameter with value to the model object

- 1 Create a model object, 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

Parameter Object Array

Index:	Name:	Value:	ValueUnits:
1	K2	0.5	

See Also addparameter, sbioparameter

ValueUnits

Purpose Parameter value units

Description The ValueUnits property indicates the unit definition of the parameter object Value property. ValueUnits can be one of the builtin units. To get a list of the builtin 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

Example

Assign a parameter with value to the model object.

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

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

- 2 Add a parameter with Value 0.5 , assign it to the model object (modelObj).

```
parameterObj1 = addparameter(modelObj, 'K1', 0.5, 'ValueUnits', 1/second)
```

MATLAB returns

Parameter Object Array

Index:	Name:	Value:	ValueUnits:
1	K4	0.5	1/second

See Also

sbioparameter, addparameter, sbioshowunits, sbioconvertunits

A

AbsoluteTolerance property
reference 6-2

abstract kinetic law object
reference 4-2

Active property
reference 6-4

addconfigset method
reference 4-4

addkineticlaw method
reference 4-8

addmodel method
reference 4-17

addparameter method
reference 4-19

addproduct method
reference 4-24

addreactant method
reference 4-27

addreaction method
reference 4-30

addrule method
reference 4-36

addspecies method
reference 4-40

Annotation property
reference 6-6

B

BoundaryCondition property
reference 6-7

BuiltInKineticLaws property
reference 6-10

BuiltInUnitPrefixes property
reference 6-12

BuiltInUnits property
reference 6-14

C

CompileOptions property
reference 6-16

configset object
reference 4-46

Conserved Moieties
function for 2-8

ConstantAmount property
reference 6-18

ConstantValue property
reference 6-20

copyobj method
reference 4-49

D

delete method
reference 4-51

DimensionalAnalysis property
reference 6-22

display method
reference 4-53

E

Ensemble Runs
function for 2-19 2-22 2-24

ErrorTolerance property
reference 6-25

Expression property
reference 6-27

F

functions

- sbioabstractkineticlaw 2-2
- sbioaddtolibrary 2-6
- sbioconsmoiety 2-8
- sbioconvertunits 2-13
- sbiocopylibrary 2-15

sbiodesktop 2-17
sbioensembleplot 2-19
sbioenssemblerun 2-22
sbioensemblestats 2-24
sbiogetmodel 2-28
sbiogetnamedstate 2-30
sbiogetsensmatrix 2-32
sbiohelp 2-36
sbiolasterror 2-37
sbiolastwarning 2-41
sbioloadproject 2-42
sbiomodel 2-43
sbioparamestim 2-47
sbioparameter 2-52
sbioreaction 2-55
sbioregisterunit 2-60
sbioregisterunitprefix 2-62
sbioremovefromlibrary 2-63
sbioreset 2-65
sbiroot 2-68
sbiорule 2-70
sbiosaveproject 2-73
sbioselect 2-74
sbioshowunitprefixes 2-78
sbioshowunits 2-79
sbiosimulate 2-81
sbiospecies 2-85
sbiounitcalculator 2-90
sbiounregisterunit 2-91
sbiounregisterunitprefix 2-93
sbiowhos 2-94
sbmlexport 2-96
sbmlimport 2-98
setactiveconfigset 4-93
setparameter 4-95
setspecies 4-97

G

getadjacencymatrix method

reference 4-54
getconfigset method
reference 4-56
getparameters method
reference 4-58
getspecies method
reference 4-60
getstoichmatrix method
reference 4-62

I

InitialAmount property
reference 6-32
InitialAmountUnits property
reference 6-33

K

kinetic law object
reference 4-64
KineticLaw property
reference 6-35
KineticLawName property
reference 6-37

L

LogDecimation property
reference 6-39

M

MaxIterations property
reference 6-41
MaxStep property
reference 6-43
methods
addconfigset 4-4
addkineticlaw 4-8
addmodel 4-17

- addparameter 4-19
- addproduct 4-24
- addreactant 4-27
- addreaction 4-30
- addrule 4-36
- addspecies 4-40
- copyobj 4-49
- delete 4-51
- getadjacencymatrix 4-54
- getconfigset 4-56
- getparameters 4-58
- getspecies 4-60
- getstoichmatrix 4-62
- removeconfigset 4-80
- reset 4-82
- rmproduct 4-84
- rmreactant 4-86
- verify 4-101

Methods

- display 4-53

model object

- reference 4-72 4-99

Models property

- reference 6-44

Moiety Conservation

- function for 2-8

N

- Name property
 - reference 6-46
- Normalization property
 - reference 6-49
- Notes property
 - reference 6-50

O

- object
 - abstract kinetic law 4-2

- configset 4-46
- kinetic law 4-64
- model 4-72 4-99
- parameter 4-75
- reaction 4-77
- root 4-88
- rule 4-90

P

- Parameter Estimation
 - function for 2-47
- parameter object
 - reference 4-75
- ParameterInputFactors property
 - reference 6-51
- Parameters property
 - reference 6-53
- ParameterVariableNames property
 - reference 6-55
- ParameterVariables property
 - reference 6-57
- Parent property
 - reference 6-59
- Products property
 - reference 6-60
- properties
 - AbsoluteTolerance 6-2
 - Active 6-4
 - Annotation 6-6
 - BoundaryCondition 6-7
 - BuiltInKineticLaws 6-10
 - BuiltInUnitPrefixes 6-12
 - BuiltInUnits 6-14
 - CompileOptions 6-16
 - ConstantAmount 6-18
 - ConstantValue 6-20
 - DimensionalAnalysis 6-22
 - ErrorTolerance 6-25
 - Expression 6-27

- InitialAmount 6-32
 - InitialAmountUnits 6-33
 - KineticLaw 6-35
 - KineticLawName 6-37
 - LogDecimation 6-39
 - MaxIterations 6-41
 - MaxStep 6-43
 - Models 6-44
 - Name 6-46
 - Normalization 6-49
 - Notes 6-50
 - ParameterInputFactors 6-51
 - Parameters 6-53
 - ParameterVariableNames 6-55
 - ParameterVariables 6-57
 - Parent 6-59
 - Products 6-60
 - RandomState 6-62
 - Reaction 6-66
 - ReactionRate 6-68
 - Reactions 6-71
 - RelativeTolerance 6-72
 - Reversible 6-74
 - Rule 6-77
 - Rules 6-82
 - RuleType 6-79
 - RuntimeOptions 6-84
 - SensitivityAnalysis 6-86
 - SensitivityAnalysisOptions 6-88
 - SolverOptions 6-92
 - SolverType 6-95
 - Species 6-97
 - SpeciesInputFactors 6-99
 - SpeciesVariableNames 6-101
 - SpeciesVariables 6-103
 - StatesToLog 6-105
 - Stoichiometry 6-107
 - StopTime 6-109
 - StopTimeType 6-110
 - Tag 6-112
 - TimeUnits 6-114
 - Type 6-115
 - UnitConversion 6-116
 - UserData 6-119
 - UserDefinedKineticLaws 6-120
 - UserDefinedUnitPrefixes 6-123
 - UserDefinedUnits 6-125
 - Value 6-127
 - ValueUnits 6-128
 - Properties
 - Reactants 6-64
- R**
- RandomState property
 - reference 6-62
 - Reactants property
 - reference 6-64
 - reaction object
 - reference 4-77
 - Reaction property
 - reference 6-66
 - ReactionRate property
 - reference 6-68
 - Reactions property
 - reference 6-71
 - RelativeTolerance property
 - reference 6-72
 - removeconfigset method
 - reference 4-80
 - reset method
 - reference 4-82
 - Reversible property
 - reference 6-74
 - rmproduct method
 - reference 4-84
 - rmreactant method
 - reference 4-86
 - root object
 - reference 4-88

rule object
reference 4-90

Rule property
reference 6-77

Rules property
reference 6-82

RuleType property
reference 6-79

RuntimeOptions property
reference 6-84

S

sbioabstractkineticlaw function
reference 2-2

sbioaddtolibrary function
reference 2-6

sbioconsmoiety function
reference 2-8

sbioconvertunits function
reference 2-13

sbiocopylibrary function
reference 2-15

sbiodesktop function
reference 2-17

sbioensembleplot function
reference 2-19

sbioensemblerrun function
reference 2-22

sbioensemblestats function
reference 2-24

sbiogetmodel function
reference 2-28

sbiogetnamedstate function
reference 2-30

sbiogetsensmatrix function
reference 2-32

sbiohelp function
reference 2-36

sbiolasterror function

reference 2-37

sbiolastwarning function
reference 2-41

sbioloadproject function
reference 2-42

sbiomodel function
reference 2-43

sbioparamestim function
reference 2-47

sbioparameter function
reference 2-52

sbioreaction function
reference 2-55

sbioregisterunit function
reference 2-60

sbioregisterunitprefix function
reference 2-62

sbioremovefromlibrary function
reference 2-63

sbioreset function
reference 2-65

sbiroot function
reference 2-68

sbiorule function
reference 2-70

sbiosaveproject function
reference 2-73

sbioselect function
reference 2-74

sbioshowunitprefixes function
reference 2-78

sbioshowunits function
reference 2-79

sbiosimulate function
reference 2-81

sbiospecies function
reference 2-85

sbiounitcalculator function
reference 2-90

sbiounregisterunit function

- reference 2-91
- sbiounregisterunitprefix function
 - reference 2-93
- sbiowhos function
 - reference 2-94
- sbmlexport function
 - reference 2-96
- sbmlimport function
 - reference 2-98
- Sensitivity Analysis
 - function for 2-32
 - properties for 6-49 6-51 6-86 6-88 6-99
- SensitivityAnalysis property
 - reference 6-86
- SensitivityAnalysisOptions property
 - reference 6-88
- setactiveconfigset function
 - reference 4-93
- setparameter function
 - reference 4-95
- setspecies function
 - reference 4-97
- SolverOptions property
 - reference 6-92
- SolverType property
 - reference 6-95
- species object
 - method summary 2-87
 - property summary 2-88
- Species property
 - reference 6-97
- SpeciesInputFactors property
 - reference 6-99
- SpeciesVariableNames property
 - reference 6-101
- SpeciesVariables property
 - reference 6-103
- StatesToLog property

- reference 6-105
- Stoichiometry property
 - reference 6-107
- StopTime property
 - reference 6-109
- StopTimeType property
 - reference 6-110

T

- Tag property
 - reference 6-112
- TimeUnits property
 - reference 6-114
- Type property
 - reference 6-115

U

- UnitConversion property
 - reference 6-116
- UserData property
 - reference 6-119
- UserDefinedKineticLaws property
 - reference 6-120
- UserDefinedUnitPrefixes property
 - reference 6-123
- UserDefinedUnits property
 - reference 6-125

V

- Value property
 - reference 6-127
- ValueUnits property
 - reference 6-128
- verify method
 - reference 4-101