

**SimBiology®**

Reference

**R2012b**

**MATLAB®**

## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

*SimBiology*<sup>®</sup> Reference

© COPYRIGHT 2005–2012 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 and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## Revision History

September 2005	Online only	New for Version 1.0 (Release 14SP3+)
March 2006	Online only	Updated for Version 1.0.1 (Release 2006a)
May 2006	Online only	Updated for Version 2.0 (Release 2006a+)
September 2006	Online only	Updated for Version 2.0.1 (Release 2006b)
March 2007	Online only	Rereleased for Version 2.1.1 (Release 2007a)
September 2007	Online only	Rereleased for Version 2.1.2 (Release 2007b)
October 2007	Online only	Updated for Version 2.2 (Release 2007b+)
March 2008	Online only	Updated for Version 2.3 (Release 2008a)
October 2008	Online only	Updated for Version 2.4 (Release 2008b)
March 2009	Online only	Updated for Version 3.0 (Release 2009a)
September 2009	Online only	Updated for Version 3.1 (Release 2009b)
March 2010	Online only	Updated for Version 3.2 (Release 2010a)
September 2010	Online only	Updated for Version 3.3 (Release 2010b)
April 2011	Online only	Updated for Version 3.4 (Release 2011a)
September 2011	Online only	Updated for Version 4.0 (Release 2011b)
March 2012	Online only	Updated for Version 4.1 (Release 2012a)
September 2012	Online only	Updated for Version 4.2 (Release 2012b)



## Function Reference

**1**

---

Modeling, Simulation, and Analysis Tools .....	1-2
Project Opening and Saving .....	1-4
SBML Model Reading and Writing .....	1-5
Object Construction .....	1-6
Pharmacokinetic Modeling .....	1-7
Units and Unit Prefixes .....	1-8

## Functions — Alphabetical List

**2**

## Method Reference

**3**

---

Objects .....	3-3
Abstract Kinetic Laws .....	3-5
Compartments .....	3-6
Configuration Sets .....	3-7

<b>Covariate Analysis</b> .....	<b>3-8</b>
<b>Events</b> .....	<b>3-9</b>
<b>Kinetic Laws</b> .....	<b>3-10</b>
<b>Models</b> .....	<b>3-11</b>
<b>Parameters</b> .....	<b>3-13</b>
<b>Pharmacokinetic Modeling</b> .....	<b>3-14</b>
<b>PKCovariate Analysis</b> .....	<b>3-15</b>
<b>Reactions</b> .....	<b>3-16</b>
<b>RepeatDose</b> .....	<b>3-17</b>
<b>Root</b> .....	<b>3-18</b>
<b>Rules</b> .....	<b>3-19</b>
<b>ScheduleDose</b> .....	<b>3-20</b>
<b>SimData</b> .....	<b>3-21</b>
<b>Species</b> .....	<b>3-22</b>
<b>Units and Unit Prefixes</b> .....	<b>3-23</b>
<b>Variants</b> .....	<b>3-24</b>
<b>Using Object Methods</b> .....	<b>3-25</b>
Constructing (Creating) Objects .....	<b>3-25</b>
Using Object Methods .....	<b>3-25</b>
Help for Objects, Methods, and Properties .....	<b>3-26</b>

**4**

Property Reference

---

**5**

Abstract Kinetic Law .....	5-3
Compartments .....	5-4
Configuration Sets .....	5-5
CovariateModel .....	5-6
Events .....	5-7
Kinetic Laws .....	5-8
Models .....	5-9
NMFileDef .....	5-10
Parameters .....	5-11
PKCompartment .....	5-12
PKData .....	5-13
PKModelDesign .....	5-14
PKModelMap .....	5-15
Reactions .....	5-16

<b>RepeatDose</b> .....	<b>5-17</b>
<b>Root</b> .....	<b>5-18</b>
<b>Rules</b> .....	<b>5-19</b>
<b>ScheduleDose</b> .....	<b>5-20</b>
<b>SimData</b> .....	<b>5-21</b>
<b>Species</b> .....	<b>5-22</b>
<b>Unit</b> .....	<b>5-23</b>
<b>Unit Prefix</b> .....	<b>5-24</b>
<b>Variant</b> .....	<b>5-25</b>
<b>Using Object Properties</b> .....	<b>5-26</b>
Entering Property Values .....	<b>5-26</b>
Retrieving Property Values .....	<b>5-26</b>
Help for Objects, Methods, and Properties .....	<b>5-27</b>

## Properties — Alphabetical List

**6**

**Index**



# Function Reference

---

Modeling, Simulation, and Analysis  
Tools (p. 1-2)

Project Opening and Saving (p. 1-4)

SBML Model Reading and Writing  
(p. 1-5)

Object Construction (p. 1-6)

Pharmacokinetic Modeling (p. 1-7)

Units and Unit Prefixes (p. 1-8)

Modeling, simulation, and analysis  
tools

Save and open projects in MATLAB®

Export and import SBML models

Create SimBiology® objects

Tools for pharmacokinetic modeling

Perform unit conversion and create  
user-defined units

## Modeling, Simulation, and Analysis Tools

<code>sbioaccelerate</code>	Prepare model object for accelerated simulations
<code>sbioconsmoiety</code>	Find conserved moieties in SimBiology model
<code>sbiodesktop</code>	Open SimBiology desktop for modeling and simulation
<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 run data
<code>sbiofittool</code>	Open SimBiology desktop for population fitting
<code>sbiogetmodel</code>	Get model object that generated simulation data
<code>sbiohelp</code>	Help for SimBiology functions
<code>sbiolasterror</code>	SimBiology last error message
<code>sbiolastwarning</code>	SimBiology last warning message
<code>sbionlinfit</code>	Perform nonlinear least-squares regression using SimBiology models
<code>sbionlmefit</code>	Estimate nonlinear mixed effects using SimBiology models
<code>sbionlmefitsa</code>	Estimate nonlinear mixed effects with stochastic EM algorithm
<code>sbioparamestim</code>	Perform parameter estimation
<code>sbioplot</code>	Plot simulation results in one figure
<code>sbioreset</code>	Delete all model objects
<code>sbioselect</code>	Search for objects with specified constraints

<code>sbiosimulate</code>	Simulate model object
<code>sbiosubplot</code>	Plot simulation results in subplots
<code>sbiotrellis</code>	Plot data or simulation results in trellis plot
<code>simbiology</code>	Open SimBiology desktop for modeling and simulation

## **Project Opening and Saving**

<code>sbioaddtolibrary</code>	Add to user-defined library
<code>sbiocopylibrary</code>	Copy library to disk
<code>sbioloadproject</code>	Load project from file
<code>sbioremovefromlibrary</code>	Remove kinetic law, unit, or unit prefix from 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 Model Reading and Writing

sbmlexport

Export SimBiology model to SBML  
file

sbmlimport

Import SBML-formatted file

## Object Construction

<code>sbioabstractkineticlaw</code>	Create kinetic law definition
<code>sbiodose</code>	Construct dose object
<code>sbiomodel</code>	Construct model object
<code>sbioroot</code>	Return SimBiology root object
<code>sbiovariant</code>	Construct variant object

## Pharmacokinetic Modeling

<code>sbiofitstatusplot</code>	Plot status of <code>sbionlmeft</code> or <code>sbionlmeftsa</code>
<code>sbionlinfit</code>	Perform nonlinear least-squares regression using SimBiology models
<code>sbionlmeft</code>	Estimate nonlinear mixed effects using SimBiology models
<code>sbionlmeftsa</code>	Estimate nonlinear mixed effects with stochastic EM algorithm
<code>sbionmfiledef</code>	NONMEM® file definition object for <code>sbionmimport</code>
<code>sbionmimport</code>	Import NONMEM-formatted data

## Units and Unit Prefixes

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



# Functions — Alphabetical List

---

# sbioabstractkineticlaw

---

**Purpose** Create kinetic law definition

**Syntax**

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

**Arguments**

<i>Name</i>	Enter a name for the kinetic law definition. 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** *abstkineticlawObj* = sbioabstractkineticlaw('Name') creates an abstract kinetic law object, with the name *Name* and returns it to *abstkineticlawObj*. Use the abstract kinetic law object to specify a *kinetic law definition*.

The *kinetic law definition* provides a mechanism for applying a specific rate law to multiple reactions. It acts as a mapping template for the reaction rate. The kinetic law definition 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.

To use the kinetic law definition, you must add it to the user-defined library with the *sbioaddtolibrary* function. To retrieve the kinetic law definitions from the user-defined library, first create a root object using *sbioroot*, then use the command `get(rootObj.UserDefinedLibrary, 'KineticLaws')`.

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

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

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

---

**Note** If you use the `sbioabstractkineticlaw` constructor function to create an object containing a reaction rate expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Method Summary

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

## Property Summary

Expression (AbstractKineticLaw, KineticLaw)	Expression to determine reaction rate equation
Name	Specify name of object

# sbioabstractkineticlaw

---

Notes	HTML text describing SimBiology object
ParameterVariables	Parameters in kinetic law definition
Parent	Indicate parent object
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Examples

- 1 Create a kinetic law definition.

```
abstkineticlawObj = sbioabstractkineticlaw('ex_myLaw1', '(k1*s)/(k2+k1+s)');
```

- 2 Assign the parameter and species variables in the expression.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined
```

```
SimBiology Abstract Kinetic Law Array
```

Index:	Library:	Name:	Expression:
1	UserDefined	ex_myLaw1	(k1*s)/(k2+k1+s)

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

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

---

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

---

## See Also

[addkineticlaw](#) | [addparameter](#) | [addreaction](#) | [sbiomodel](#)

# sbioaccelerate

---

**Purpose** Prepare model object for accelerated simulations

**Syntax**

```
sbioaccelerate(modelObj)
sbioaccelerate(modelObj, configsetObj)
sbioaccelerate(modelObj, variantObj)
sbioaccelerate(modelObj, doseObj)
sbioaccelerate(modelObj, configsetObj, variantObj)
sbioaccelerate(modelObj, configsetObj, doseObj)
sbioaccelerate(modelObj, configsetObj, variantObj, doseObj)
```

**Description** Use `sbioaccelerate` to prepare a model for accelerated simulations before running `sbiosimulate`. You must use the same model, configset, and dose input arguments when calling both functions, however, you can use any or no variant input arguments when calling `sbioaccelerate`.

---

**Note** You need to run `sbioaccelerate` again, before calling `sbiosimulate`, if you make any modifications to this model, other than:

- Changes to the variants
  - Changes to values for the `InitialAmount` of species
  - Changes to the `Capacity` of compartments
  - Changes to the `Value` of parameters
- 

`sbioaccelerate(modelObj)` prepares a model object (`modelObj`) for accelerated simulation using its active configset, active variants and active doses. A SimBiology model can contain multiple configsets with one being active at any given time. The active configset contains the settings that will be used to prepare the model for acceleration.

`sbioaccelerate(modelObj, configsetObj)` uses the configset `configsetObj`. A configset object stores simulation specific information.

`sbioaccelerate(modelObj, variantObj)` uses the variant object or array of variant objects *variantObj*. A variant object's settings supersede model property-values.

`sbioaccelerate(modelObj, doseObj)` simulates *modelObj* using the dose object or array of dose objects *doseObj*. A SimBiology Dose object defines additions that are made to species amounts or parameter values.

`sbioaccelerate(modelObj, configsetObj, variantObj)`,  
`sbioaccelerate(modelObj, configsetObj, doseObj)` uses the configset *configsetObj* and variant object or variant array *variantObj* or dose object or dose array *doseObj*. Note if the third argument is empty it is assumed to be *variantObj*.

`sbioaccelerate(modelObj, configsetObj, variantObj, doseObj)`

---

**Note** Microsoft® Visual Studio® 2010 run-time libraries must be available on any computer running accelerated models generated using Microsoft Windows® SDK. If you plan to redistribute your accelerated models to other MATLAB users, be sure they have the run-time libraries.

---

## Input Arguments

### **modelObj**

SimBiology model object.

### **configsetObj**

Specify the configuration set object to use in the simulation. For more information about configuration sets, see `Configset` object.

### **variantObj**

Specify the variant object to apply to the model during the simulation. For more information about variant objects, see `Variant` object.

### **doseObj**

# sbioaccelerate

---

Specify the dose object to apply to the model during the simulation. For more information about dose objects, see `ScheduleDose` object and `RepeatDose` object.

## Examples

This example shows how to prepare a model for accelerated simulation.

```
% Create a SimBiology model from an SBML file.
m = sbmlimport('lotka.xml');

% Prepare the model for accelerated simulation.
sbioaccelerate(m);

% Simulate the model for different initial amounts of x.
x = sbioselect(m, 'type', 'species', 'name', 'x');
for i=1:10
    x.initialAmount = i;
    sd(i) = sbiosimulate(m);
end

% Plot the results.
sbioplot(sd);
```

## See Also

`sbiosimulate`



**Purpose** Add to user-defined library

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

## Arguments

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

## Description

The function `sbioaddtolibrary` adds kinetic law definitions, units, and unit prefixes to the user-defined library.

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

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

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

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

library components become available automatically in future MATLAB sessions.

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

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

To remove the library component from the user-defined library, use the function `sbioremovefromlibrary`. You cannot remove a kinetic law definition being used by a reaction.

## Examples

This example shows how to create a kinetic law definition and add it to the user-defined library.

- 1 Create a kinetic law definition.

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

- 2 Assign the parameter and species variables in the expression.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined
```

```
SimBiology Abstract Kinetic Law Array
```

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

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

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'A + B <-> B + C');  
kineticlawObj = addkineticlaw(reactionObj, 'ex_myLaw1');
```

---

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

---

## See Also

[addkineticlaw](#) | [sbioabstractkineticlaw](#) | [sbioremovefromlibrary](#)  
| [sbioroot](#) | [sbiounit](#) | [sbiounitprefix](#)

# sbioconsmoiety

---

## Purpose

Find conserved moieties in SimBiology model

## Syntax

```
[G, Sp] = sbioconsmoiety(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> . If the species are in multiple compartments, species names are qualified with the compartment name in the form <code>compartmentName.speciesName</code> . For example, <code>nucleus.DNA</code> , <code>cytoplasm.mRNA</code> .
<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>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>	<p>Link matrix relating SI and SD. The link matrix <i>LO</i> satisfies <math>ND = LO * NR</math>. For the 'link' functionality, species with their BoundaryCondition or ConstantAmount properties set to true are treated as having stoichiometry of zero in all reactions.</p> <p><i>LO</i> is a sparse matrix. To convert it to a full matrix, use the full function.</p>
<i>NR</i>	<p>Reduced stoichiometry matrices containing one row for each independent species. The concatenated matrix [<i>NR</i>; <i>ND</i>] is a row-permuted version of the full stoichiometry matrix of <i>modelObj</i>.</p> <p><i>NR</i> is a sparse matrix. To convert it to a full matrix, use the full function.</p>
<i>ND</i>	<p>Reduced stoichiometry matrices containing one row for each dependent species. The concatenated matrix [<i>NR</i>; <i>ND</i>] is a row-permuted version of the full stoichiometry matrix of <i>modelObj</i>.</p> <p><i>ND</i> is a sparse matrix. To convert it to a full matrix, use the full function.</p>

## Description

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

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

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

- When you specify 'qr', `sbioconsmoiety` uses an algorithm based on QR factorization. From a numerical standpoint, this is the most efficient and reliable approach.
- When you specify 'rreduce', `sbioconsmoiety` uses an algorithm based on row reduction, which yields better numbers for smaller models. This is the default.
- When you specify 'semipos', `sbioconsmoiety` returns conservation relations in which all the coefficients are greater than or equal to 0, permitting a more transparent interpretation in terms of physical quantities.

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

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

`H = sbioconsmoiety(modeObj, 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(modeObj, 'link')` uses a QR-based algorithm to compute information relevant to the dimensional reduction, via conservation relations, of the reaction network in `modeObj`.

## Examples

### Example 1

This example shows conserved moieties in a cycle.

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

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

- 2 Look for conserved moieties.

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

g =

     1     1     1

sp =

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

## Example 2

Explore semipositive conservation relations in the oscillator model.

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

ans =

     'p0l + p0l_0pA + p0l_0pB + p0l_0pC'
     'OpB + p0l_0pB + pA_0pB1 + pA_0pB_pA + pA_0pB2'
     'OpA + p0l_0pA + pC_0pA1 + pC_0pA2 + pC_0pA_pC'
     'OpC + p0l_0pC + pB_0pC1 + pB_0pC2 + pB_0pC_pB'
```

## See Also

`getstoichmatrix`

## How To

- “Conserved Moiety Determination”



**Purpose** Convert unit and unit value to new unit

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

**Description** `sbioconvertunits(Obj, 'unit')` converts the current `*Units` property on SimBiology object, *Obj* to the unit, *unit*. This function configures the `*Units` property to *unit* and updates the corresponding value property. For example, `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 `compartment`, `parameter`, and `species` objects. For example, if *Obj* is a `species` object with `InitialAmount` configured to 1 and `InitialAmountUnits` configured to `mole`, after the call to `sbioconvertunits` with *unit* specified as `molecule`, `speciesObj` `InitialAmount` is 6.0221e23 and `InitialAmountUnits` is `molecule`.

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

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

At the command prompt, type:

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

SimBiology Species Array

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

# sbioconvertunits

---

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

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

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

Units are converted from molecule to mole.

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

```
ans =
```

```
mole
```

The InitialAmount value is changed.

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

```
ans =
```

```
3.8192e-023
```

## See Also

sbioshowunits

## How To

• sbioshowunits

## Purpose

Copy library to disk

## Syntax

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

## Description

`sbiocopylibrary ('kineticlaw','LibraryFileName')` copies all user-defined kinetic law definitions to the file `LibraryFileName.sbklib` and places the copied file in the current directory.

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

To get the kinetic law definitions that are in the built-in or user-defined libraries, first create a root object using `sbioroot`, then use the commands `get(rootObj.BuiltInLibrary, 'KineticLaws')` or `get(rootObj.UserDefinedLibrary, 'KineticLaws')`.

To add a kinetic law definition to the user-defined library, use `sbioaddtolibrary`.

To add a unit to the user-defined library, use `sbiounit` followed by `sbioaddtolibrary`. To add a unit prefix to the user-defined library, use `sbiounitprefix` followed by `sbioaddtolibrary`.

## Examples

Create a kinetic law definition, add it to the user-defined library, and then copy the user-defined kinetic law library to a `.sbklib` file.

- 1 Create a kinetic law definition.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

# sbiocopylibrary

---

```
sbiowhos -kineticlaw -userdefined
```

```
SimBiology Abstract Kinetic Law Array
```

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

**3** Copy the user-defined kinetic law library.

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

**4** Verify with sbiowhos.

```
sbiowhos -kineticlaw myLibFile
```

## See Also

```
sbioaddtolibrary | sbioabstractkineticlaw |  
sbioremovefromlibrary | sbiounit | sbiounitprefix
```

**Purpose** Open SimBiology desktop for modeling and simulation

**Syntax**

```
sbiodesktop
sbiodesktop(mode1Obj)
sbiodesktop(File)
```

**Input Arguments**

<i>mode1Obj</i>	SimBiology model object or an array of model objects.
<i>File</i>	String specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.

**Description**

sbiodesktop opens the SimBiology desktop, which lets you:

- Build a SimBiology model by representing reaction pathways and entering kinetic data for the reactions.
- Import or export SimBiology models to and from the MATLAB workspace or from a Systems Biology Markup Language (SBML) file.
- Modify an existing SimBiology model.
- Simulate a SimBiology model through individual or ensemble runs.
- View results from the simulation.
- Perform analysis tasks such as sensitivity analysis, parameter and species scans, and calculation of conserved moieties.
- Create and/or modify user-defined units and unit prefixes.
- Create and/or modify user-defined kinetic laws.

sbiodesktop(*mode1Obj*) opens the SimBiology desktop with *mode1Obj*, a SimBiology model object. If there is a project open in the SimBiology desktop, this command adds *mode1Obj* to the project.

sbiodesktop(*File*) opens the project specified by *File* in the SimBiology desktop. *File* is a string specifying a file name or path and file name of

an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. If a project is open in the desktop, the software replaces it with the new project, after prompting you to save any changes.

The Parent property of a SimBiology model object is set to the SimBiology root object. The root object contains a list of model objects that are accessible from the MATLAB command line and from the SimBiology desktop. Because both the command line and the desktop point to the same model object in the `Root` object, changes you make to the model at the command line are reflected in the desktop, and vice versa.

---

**Note** The `sbioreset` command removes all models from the root object. Therefore, this command also removes all models from the SimBiology desktop.

---

## Examples

Create a SimBiology model in the MATLAB workspace, and then open the SimBiology desktop with the model:

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

## See Also

`sbiroot` | `sbiofittool` | `simbiology`

**Purpose**

Construct dose object

**Syntax**

```
doseObj = sbiodose('DoseName')
doseObj = sbiodose('DoseName', 'DoseType')
doseObj = sbiodose(...'PropertyName', PropertyValue...)
```

**Inputs**

<i>DoseName</i>	Name of the dose object.
<i>DoseType</i>	Selects which type of dose object to construct. Enter either 'schedule' or 'repeat' <ul style="list-style-type: none"> <li>'schedule' creates a ScheduleDose object and defines the dose with a time array, amount array, and rate array.</li> <li>'repeat' creates a RepeatDose object and defines the dose with a dose amount, number of dose repetitions, and the time between doses.</li> </ul>

**Output Arguments**

<i>doseObj</i>	ScheduleDose or RepeatDose object.
----------------	------------------------------------

**Description**

*doseObj* = sbiodose('DoseName') constructs a SimBiology RepeatDose object (*doseObj*), assigns *DoseName* to the property Name, and assigns [] to the property Parent.

*doseObj* = sbiodose('DoseName', 'DoseType') constructs either a SimBiology ScheduleDose object or RepeatDose object (*doseObj*).

*doseObj* = sbiodose(...'PropertyName', PropertyValue...) defines dose object properties. You can enter the property name/property value pairs 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 *doseObj* properties with the get command and modify *doseObj* properties with the set command.

Before you use a dose object in a simulation, you must add the object to a SimBiology model with the method `adddose`.

## Examples

Construct a repeat dose object:

- 1 In the MATLAB Command Window, enter:

```
doseObj1 = sbiodose('dose1', 'repeat');
```

- 2 Define a repeating dose:

```
doseObj1.Amount      = 5;  
doseObj1.Repeat      = 6;  
doseObj1.Interval    = 24;  
doseObj1.TimeUnits   = 'hour';  
doseObj1.TargetName  = 'Central.x';
```

---

Construct a schedule dose object:

- 1 In the MATLAB Command Window, enter:

```
doseObj2 = sbiodose('dose2', 'schedule');
```

- 2 Define a dose schedule:

```
doseObj2.Time        = [0 24 48];  
doseObj2.Amount      = [10 5 5];  
doseObj2.TargetName  = 'Central.Drug';
```

## See Also

[adddose](#) | [getdose](#) | [removedose](#) | [copyobj](#) | [get](#) | [set](#)

## How To

- Model object
- ScheduleDose object
- RepeatDose object



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

**Syntax**

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

## Arguments

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

## Description

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

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

# sbioensembleplot

---

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

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

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

## Examples

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

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

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

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

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

- 3 Perform an ensemble of 20 runs with no interpolation.

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

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

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

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

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

## See Also

[sbioenssemblerun](#) | [sbioensemblestats](#) | [sbiomodel](#)

## Purpose

Multiple stochastic ensemble runs of SimBiology model

## Syntax

```
simdataObj = sbioensemblerun(modelObj, Numruns)  
simdataObj = sbioensemblerun(modelObj, Numruns,  
    Interpolation)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj,  
    Interpolation)  
simdataObj = sbioensemblerun(modelObj, Numruns, variantObj)  
simdataObj = sbioensemblerun(modelObj, Numruns, variantObj,  
    Interpolation)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj,  
    variantObj)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj,  
    variantObj, Interpolation)
```

## Arguments

<i>simdataObj</i>	An array of SimData objects containing simulation data generated by sbioensemblerun. All elements of <i>simdataObj</i> contain data for the same states in the same model.
<i>modelObj</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.

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

## Description

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

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

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

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

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

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

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

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

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

## Examples

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

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

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

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

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

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

---

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

---

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

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

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

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

## See Also

`addconfigset` | `getconfigset` | `sbioensemblestats` | `sbioensembleplot` | `setactiveconfigset` | SimData object

**Purpose** Get statistics from ensemble run data

**Syntax**

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

## Arguments

<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 SimData objects in <i>simDataObj</i> using <code>selectbyname</code> .
<i>simDataObj</i>	A cell array of SimData objects, where each SimData object holds data for a separate simulation run. All elements of <i>simDataObj</i> must contain data for the same states in the same model. When the time vectors of the elements of <i>simDataObj</i> are not identical, <i>simDataObj</i> is first resampled onto a common time vector (see <i>interpolation</i> below).
<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 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 names in <i>n</i> corresponds to the order of columns of <i>m</i> and <i>v</i> .

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

## Description

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

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

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

## Examples

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

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

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



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

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

- 3 Perform an ensemble of 20 runs with no interpolation.

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

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

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

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

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

## See Also

[sbioensemblerrun](#) | [sbioensembleplot](#) | [sbiomodel](#)

# sbiofitstatusplot

---

**Purpose** Plot status of sbionlmefit or sbionlmefitsa

**Syntax** `stop = sbiofitstatusplot(beta, status, state)`

**Description** `stop = sbiofitstatusplot(beta, status, state)` initializes or updates a plot with the fixed effects, *beta*, the log likelihood *status.fval*, and the variance of the random effects, `diag(status.Psi)`.

The function returns an output (*stop*) to satisfy requirements for the 'OutputFcn' option of `nlmefit` or `nlmefitsa`. For `sbiofitstatusplot`, the value of *stop* is always `false`.

Use `sbiofitstatusplot` to obtain status information about NLME fitting when using the `sbionlmefit` or `sbionlmefitsa` function. Specify `sbiofitstatusplot` as a function handle by using the `optionStruct` (options structure) input argument to `sbionlmefit` or `sbionlmefitsa`. Use `sbiofitstatusplot` or customize your own function to use in the options structure.

## Input Arguments

**beta**  
The current fixed effects.

**status**  
Structure containing several fields.

Field	Value
inner	Structure describing the current status of the inner iterations within the ALT and LAP procedures, with the fields: <ul style="list-style-type: none"><li>procedure<ul style="list-style-type: none"><li>'PNLS', 'LME', or 'none' when the procedure is 'ALT'</li><li>'PNLS', 'PLM', or 'none' when the procedure is 'LAP'</li></ul></li></ul>

Field	Value
	<ul style="list-style-type: none"> <li>• state — 'init', 'iter', 'done', or 'none'</li> <li>• iteration — Integer starting from 0, or NaN</li> </ul>
procedure	'ALT' or 'LAP'
iteration	Integer starting from 0
fval	Current log-likelihood
Psi	Current random-effects covariance matrix
theta	Current parameterization of Psi
mse	Current error variance

**state**

Either 'init', 'iter', or 'done'.

**Definitions**

**Alt**

Alternating algorithm for the optimization of the LME or RELME approximations

**FO**

First-order estimate

**FOCE**

First-order conditional estimate

**LAP**

Optimization of the Laplacian approximation for FO or FOCE

**LME**

Linear mixed-effects estimation

**NLME**

Nonlinear mixed effects

# sbiofitstatusplot

---

## **PLM**

Profiled likelihood maximization

## **PNLS**

Penalized nonlinear least squares

## **RELME**

Restricted likelihood for the linear mixed-effects model

## **Examples**

Obtain status information for NLME fitting:

```
% Create options structure with 'OutputFcn'.  
optionStruct.Options.OutputFcn = @sbiofitstatusplot;  
% Pass options structure with OutputFcn to sbionlmeft function.  
results = sbionlmeft(..., optionStruct);
```

## **See Also**

[nlmeft](#) | [sbionlinfit](#) | [sbionlmeft](#) | [sbionlmeftsa](#)

## **How To**

- “Obtaining the Status of Fitting”

<b>Purpose</b>	Open SimBiology desktop for population fitting
<b>Syntax</b>	<code>sbiofittool</code>
<b>Description</b>	<p><code>sbiofittool</code> opens the SimBiology desktop in a state designed for:</p> <ul style="list-style-type: none"><li>• Importing and plotting data for fitting</li><li>• Selecting from a library of pharmacokinetic models</li><li>• Performing population fit tasks using <code>sbionlmeffit</code> or <code>sbionlmefitsa</code></li><li>• Performing individual fit tasks using <code>sbionlinfit</code></li></ul> <p><code>sbiofittool</code> opens a simplified configuration of the SimBiology desktop. However, all desktop functionality is available.</p> <p>If you opened the SimBiology desktop using the <code>simbiology</code> function, then <code>sbiofittool</code> changes the desktop layout to optimize it for population fitting.</p>
<b>See Also</b>	<code>simbiology</code>

# sbiogetmodel

---

**Purpose** Get model object that generated simulation data

**Syntax** `modelObj = sbiogetmodel(simDataObj)`

## Arguments

<code>simDataObj</code>	SimData object returned by the function <code>sbiosimulate</code> or by <code>sbioensemblrun</code> .
<code>modelObj</code>	Model object associated with the SimData object.

## Description

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

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

## Examples

Retrieve the model object that generated the SimData object.

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

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

- 2 Get the model that generated the simulation results.

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

Model Components:

Models:	0
Parameters:	0
Reactions:	42

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

**3** Check that the two models are the same.

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

## See Also

`sbiosimulate`

# sbiohelp

---

**Purpose** Help for SimBiology functions

---

**Note** sbiohelp has been removed.

---

**Syntax**

```
sbiohelp('FunctionName')  
h = sbiohelp ('FunctionName')
```

**Description** sbiohelp('FunctionName') displays information for a SimBiology function (*FunctionName*).

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

You can get general information on the SimBiology software by specifying *FunctionName* as 'sbio'. General information about a SimBiology object can be returned by specifying *FunctionName* as one of the following:

```
'AbstractKineticLaw', 'Compartment', 'CompileOptions',  
'Configset', 'Event', 'ExplicitTauSolverOptions',  
'ImplicitTauSolverOptions', 'KineticLaw', 'ODESolverOptions',  
'Model', 'Parameter', 'PKData', 'PKCompartment', 'PKModelDeign',  
'PKModelMap', 'Reaction', 'Root', 'Rule', 'RuntimeOptions',  
'SensitivityAnalysisOptions', 'SimData', 'Species',  
'SSASolverOptions', 'Library', 'Unit', 'UnitPrefix', or  
'Variant'.
```

**Examples**

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

**See Also** help



**Purpose** SimBiology last error message

**Syntax**

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

**Arguments**

<i>diagstruct</i>	The diagnostic structure holding Type, Message ID, and Message for the errors.
-------------------	--------------------------------------------------------------------------------

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

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

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

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

**Examples** This example shows how to use `verify` and `sbiolasterror`.

1 Import a model.

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

```
SimBiology Model - RadioactiveDecay
```

```
Model Components:
Models:           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](#)

## How To

- [sbioroot](#)

**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 the software. The fields of the diagnostic structure are:

<b>Type</b>	'warning'
<b>MessageID</b>	The message ID for the warning (for example, 'SimBiology:DANotPerformedReactionRate')
<b>Message</b>	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`

**How To**

- `sbiroot`

# sbioloadproject

---

## Purpose

Load project from file

## Syntax

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

## Description

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

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

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

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

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

## See Also

`sbioaddtolibrary` | `sbioremovefromlibrary` | `sbiosaveproject` | `sbiowhos`

## How To

- `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 “Property Summary” on page 2-49.
<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/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

Simulate *modelObj* with the function `sbiosimulate`.

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

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

**Method Summary**

<code>addcompartment (model, compartment)</code>	Create compartment object
<code>addconfigset (model)</code>	Create configuration set object and add to model object

adddose (model)	Add dose object to model
addevent (model)	Add event object to model object
addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
addreaction (model)	Create reaction object and add to model object
addrule (model)	Create rule object and add to model object
addspecies (model, compartment)	Create species object and add to compartment object within model object
addvariant (model)	Add variant to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
export (model)	Export SimBiology model
get (any object)	Get object properties
getadjacencymatrix (model)	Get adjacency matrix from model object
getconfigset (model)	Get configuration set object from model object
getdose (model)	Return SimBiology dose object
getequations	Return system equations for model object
getstoichmatrix (model)	Get stoichiometry matrix from model object
getvariant (model)	Get variant from model



removeconfigset (model)	Remove configuration set from model
removedose (model)	Add dose object to model
removevariant (model)	Remove variant from model
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties
setactiveconfigset (model)	Set active configuration set for model object
verify (model, variant)	Validate and verify SimBiology model

**Property Summary**

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

## Examples

- 1 Create a SimBiology model object.

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

- 2 List all modelObj properties and the current values.

```
get(modelObj)
```

MATLAB returns:

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

- 3 Display a summary of modelObj contents.

```
modelObj
```

```
SimBiology Model - cell
```

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

## See Also

```
addcompartment | addconfigset | addevent | addkineticlaw |
addparameter | addreaction | addrule | addspecies | copyobj |
get | sbioroot | sbiosimulate | set
```

**Purpose** Perform nonlinear least-squares regression using SimBiology models

**Syntax**

```

results = sbionlinfit(modelObj, pkModelMapObject,
    pkDataObj,
    InitEstimates)
results = sbionlinfit(modelObj, pkModelMapObject,
    pkDataObj,
    InitEstimates, Name, Value)
results = sbionlinfit(modelObj, pkModelMapObject,
    pkDataObj,
    InitEstimates, optionStruct)
[results, SimDataI] = sbionlinfit(...)

```

**Description**

---

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

---

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

*results* = sbionlinfit(*modelObj*, *pkModelMapObject*, *pkDataObj*, *InitEstimates*, *Name*, *Value*) performs least-squares regression, with additional options specified by one or more *Name*, *Value* pair arguments.

Following is an alternative to the previous syntax:

*results* = sbionlinfit(*modelObj*, *pkModelMapObject*, *pkDataObj*, *InitEstimates*, *optionStruct*) specifies *optionStruct*, a structure containing fields and values used by the options input structure to the nlinfit function.

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

## Input Arguments

### **modelObj**

SimBiology model object used to fit observed data.

---

**Note** If using a model object containing active doses (that is, containing dose objects created using the `adddose` method, and specified as active using the `Active` property of the dose object), be aware that these active doses are ignored by the `sbionlinfit` function.

---

### **pkModelMapObject**

PKModelMap object that defines the roles of the model components in the estimation. For details, see `PKModelMap` object.

---

**Note** If using a PKModelMap object that specifies multiple doses, ensure each element in the `Dosed` property is unique.

---

### **pkDataObj**

PKData object that defines the data to use in fitting, and the roles of the data columns used for estimation. For details, see `PKData` object.

---

**Note** For each subset of data belonging to a single group (as defined in the data column specified by the `GroupLabel` property), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

- These data points are not averaged, but fitted individually.
  - Different numbers of observations at different times cause some time points to be weighted more.
- 

### **InitEstimates**

Vector of initial parameter estimates for each parameter estimated in *pkModelMapObject.Estimated*. The length of *InitEstimates* must equal at least the length of *pkmodelMapObject.Estimated*. The elements of *InitEstimates* are transformed as specified by the *ParamTransform* name-value pair argument.

For details on specifying initial estimates, see “Setting Initial Estimates”.

### **optionStruct**

Structure containing fields and values used by the options input structure to the *nlinfit* function. The structure can also use the name-value pairs listed below as fields and values. Defaults for *optionStruct* are the same as for the options input structure to *nlinfit*, except for:

- *DerivStep* — Default is the lesser of  $1e-4$ , or the value of the *SolverOptions.RelativeTolerance* property of the configuration set associated with *modelObj*, with a minimum of  $\text{eps}^{(1/3)}$ .
- *FunValCheck* — Default is off.

### **Name-Value Pair Arguments**

Specify optional comma-separated pairs of *Name,Value* arguments, where *Name* is the argument name and *Value* is the corresponding value. *Name* must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as *Name1,Value1,...,NameN,ValueN*.

The *Name,Value* arguments are the same as the fields and values in the options structure accepted by *nlinfit*. For a complete list, see the options input argument in the *nlinfit* reference page in the Statistics Toolbox documentation. The defaults for *Name,Value* arguments are the same as for the options structure accepted by *nlinfit*, except for:

- *DerivStep* — Default is the lesser of  $1e-4$ , or the value of the *SolverOptions.RelativeTolerance* property of the configuration set associated with *modelObj*, with a minimum of  $\text{eps}^{(1/3)}$ .

- FunValCheck — Default is off.

Following are additional Name, Value arguments that you can use with sbionlinfit.

## ParamTransform

Vector of integers specifying a transformation function for each estimated parameter. The transformation function, *f*, takes *estimate* as an input and returns *beta*:

$\text{beta} = f(\text{estimate})$

Each element in the vector must be one of these integers specifying the transformation for the corresponding value of *estimate*:

- 0 –  $\text{beta} = \text{estimate}$
- 1 –  $\text{beta} = \log(\text{estimate})$  (default)
- 2 –  $\text{beta} = \text{probit}(\text{estimate})$
- 3 –  $\text{beta} = \text{logit}(\text{estimate})$

For details, see “Specifying Parameter Transformations”.

## ErrorModel

String specifying the form of the error term. Default is 'constant'. Each model defines the error using a standard normal (Gaussian) variable *e*, the function value *f*, and one or two parameters *a* and *b*. Choices are:

- 'constant':  $y = f + a*e$
- 'proportional':  $y = f + b*\text{abs}(f)*e$
- 'combined':  $y = f + (a+b*\text{abs}(f))*e$
- 'exponential':  $y = f*\exp(a*e)$ , or equivalently  $\log(y) = \log(f) + a*e$

If you specify an error model, the *results* output argument includes an *errorparam* property, which has the value:

- $a$  for 'constant' and 'exponential'
- $b$  for 'proportional'
- $[a\ b]$  for 'combined'

---

**Note** If you specify an error model, you cannot specify weights.

---

### Weights

Either of the following:

- A matrix of real positive weights, where the number of columns corresponds to the number of responses. That is, the number of columns must equal the number of entries in the `DependentVarLabel` property of `pkDataObj`. The number of rows in the matrix must equal the number of rows in the data set.
- A function handle that accepts a vector of predicted response values and returns a vector of real positive weights.

---

**Note** If using a function handle, the weights must be a function of the response (dependent variable).

---

Default is no weights. If you specify weights, you cannot specify an error model.

### Pooled

Logical specifying whether `sbionlinfit` does fitting for each individual (`false`) or if it pools all individual data and does one fit (`true`). If set to `true`, `sbionlinfit` uses the same model parameters for each dose level.

**Default:** `false`

## Output Arguments

### results

1-by- $N$  array of objects, where  $N$  is the number of groups in  $pkDataObj$ . There is one object per group, and each object contains these properties:

- **ParameterEstimates** — A dataset array containing fitted coefficients and their standard errors.
- **CovarianceMatrix** — Estimated covariance matrix for the fitted coefficients.
- **beta** — Vector of scalars specifying the fitted coefficients in transformed space.
- **R** — Vector of scalars specifying the residual values, where  $R(i,j)$  is the residual for the  $i$ th time point and the  $j$ th response in the group of data. If your model includes:
  - A single response, then **R** is a column vector of residual values associated with time points in the group of data.
  - Multiple responses, then **R** is a matrix of residual values associated with time points in the group of data, for each response.
- **J** — Matrix specifying the Jacobian of the model, with respect to an estimated parameter, that is

$$J(i, j, k) = \left. \frac{\partial y_k}{\partial \beta_j} \right|_{t_i}$$

where  $t_i$  is the  $i$ th time point,  $\beta_j$  is the  $j$ th estimated parameter in the transformed space, and  $y_k$  is the  $k$ th response in the group of data.

If your model includes:

- A single response, then **J** is a matrix of Jacobian values associated with time points in the group of data.
- Multiple responses, then **J** is a 3-D array of Jacobian values associated with time points in the group of data, for each response.
- **COVB** — Estimated covariance matrix for the transformed coefficients.



- `mse` — Scalar specifying the estimate of the error of the variance term.
- `errorparam` — Estimated parameters of the error model. This property is a scalar if you specify 'constant', 'exponential', or 'proportional' for the error model. This property is a two-element vector if you specify 'combined' for the error model. This property is an empty array if you specify weights using the 'Weights' name-value pair argument.

**SimData**

`SimData` object containing data from simulating the model using estimated parameter values for individuals. This object includes observed states and logged states.

**See Also**

`PKData` object | `PKModelDesign` object | `PKModelDesign` object | `PKModelMap` object | `Model` object | `sbionlmefit` | `nlinfit` | `sbionlmefitsa`

**How To**

- “Performing Data Fitting with PKPD Models”

# sbionlmeffit

---

**Purpose** Estimate nonlinear mixed effects using SimBiology models

**Syntax**

```
results = sbionlmeffit(modelObj, pkModelMapObject,  
    pkDataObject, InitEstimates)  
results = sbionlmeffit(modelObj, pkModelMapObject,  
    pkDataObject, CovModelObj)  
results = sbionlmeffit(..., Name, Value)  
results = sbionlmeffit(..., optionStruct)  
[results, SimDataI, SimDataP] = sbionlmeffit(...)
```

## Description

---

**Note** This function requires `nlmeffit` in Statistics Toolbox (Version 7.0 or later).

---

`results = sbionlmeffit(modelObj, pkModelMapObject, pkDataObject, InitEstimates)` performs nonlinear mixed-effects estimation using the SimBiology model, `modelObj`, and returns estimated results in the `results` structure.

`results = sbionlmeffit(modelObj, pkModelMapObject, pkDataObject, CovModelObj)` specifies the relationship between parameters and covariates using `CovModelObj`, a `CovariateModel` object. The `CovariateModel` object also provides the parameter transformation.

`results = sbionlmeffit(..., Name, Value)` performs nonlinear mixed-effects estimation, with additional options specified by one or more `Name, Value` pair arguments.

Following is an alternative to the previous syntax:

`results = sbionlmeffit(..., optionStruct)` specifies `optionStruct`, a structure containing fields and values, that are the name-value pair arguments accepted by `nlmeffit`. The defaults for `optionStruct` are the same as the defaults for the arguments used by `nlmeffit`, with the exceptions explained in “Input Arguments” on page 2-59.

---

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

## Input Arguments

### **modelObject**

SimBiology model object used to fit observed data.

---

**Note** If using a model object containing active doses (that is, containing dose objects created using the `adddose` method, and specified as active using the `Active` property of the dose object), be aware that these active doses are ignored by the `sbionlmeft` function.

---

### **pkModelMapObject**

PKModelMap object that defines the roles of the model components used for estimation. For details, see PKModelMap object.

---

**Note** If using a PKModelMap object that specifies multiple doses, ensure each element in the `Dosed` property is unique.

---

### **pkDataObject**

PKData object that defines the data to use in fitting, and the roles of the columns used for estimation. *pkDataObject* must define target data for at least two groups. For details, see PKData object.

---

**Note** For each subset of data belonging to a single group (as defined in the data column specified by the `GroupLabel` property), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

- These data points are not averaged, but fitted individually.
  - Different numbers of observations at different times cause some time points to be weighted more.
- 

### **InitEstimates**

Vector of initial estimates for the fixed effects. The first `P` elements of *InitEstimates* correspond to the fixed effects for each `P` element of *pkModelMapObject.Estimated*. Additional elements correspond to the fixed effects for covariate factors. The first `P` elements of *InitEstimates* are transformed as specified by the `ParamTransform` name-value pairs (log transformed by default). For details on specifying initial estimates, see “Setting Initial Estimates”.

### **CovModelObj**

`CovariateModel` object that defines the relationship between parameters and covariates. For details, see `CovariateModel` object.

---

**Tip** To simultaneously fit data from multiple dose levels, omit the random effect (`eta`) from the expressions in the `CovariateModel` object.

---

### **optionStruct**

Structure containing fields and values that are the name-value pairs accepted by the `nImeFit` function. The defaults for *optionStruct* are the same as the defaults for the arguments used by `nImeFit`, with the exceptions noted in “Name-Value Pair Arguments” on page 2-61.

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` field. This function lets you monitor the status of fitting.

---

---

**Tip** To simultaneously fit data from multiple dose levels, use the `InitEstimates` input argument and set the value of the `REParamsSelect` field to a 1-by- $n$  logical vector, with all entries set to `false`, where  $n$  equals the number of fixed effects.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name, Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

The `sbionlmeft` function uses the name-value pair arguments supported by the `nlmeft` function.

These `nlmeft` name-value pairs are hard-coded in `sbionlmeft`, and therefore, you cannot set them:

- `FEParamsSelect`
- `FEConstDesign`
- `FEGroupDesign`
- `FEObsDesign`
- `REConstDesign`
- `REGroupDesign`
- `REObsDesign`
- `Vectorization`

If you provide a `CovariateModel` object as input to `sbionlmeft`, then these `nlmeft` name-value pairs are computed from the covariate model, and therefore, you cannot set them:

- `FEGroupDesign`
- `ParamTransform`
- `REParamsSelect`

You can set all other `nlmeft` name-value pairs. For details, see the `nlmeft` reference page.

Be aware that the defaults for these `nlmeft` name-value pairs differ when used by `sbionlmeft`:

### **FEGroupDesign**

Numeric array specifying the design matrix for each group. For details, see “Specifying a Nonlinear, Mixed-Effects Model”.

**Default:** `repmat(eye(P), [1 1 nGroups])`, where `P` = the number of estimated parameters, and `nGroups` = the number of groups in the observed data.

### **ParamTransform**

Vector of integers specifying how the parameters are distributed. For details, see “Specifying Parameter Transformations”.

---

**Note** Do not use the `ParamTransform` option to specify parameter transformations when providing a `CovariateModel` object to a fitting function. The `CovariateModel` object provides the parameter transformation.

---

**Default:** Vector of ones, which specifies all parameters are log transformed.

### OptimFun

String specifying the optimization function used in maximizing the likelihood.

**Default:** `fminunc`, if you have Optimization Toolbox™ installed. Otherwise, the default is `fminsearch`.

### Options

Structure containing multiple fields, including `DerivStep`, a scalar or vector specifying the relative difference used in the finite difference gradient calculation, and `FunValCheck`, a logical specifying whether to check for invalid values, such as NaN or Inf, from `modelfun`.

**Default:** The default for `DerivStep` is the lesser of  $1e-4$ , or the value of the `SolverOptions.RelativeTolerance` property of the configuration set associated with `modelObj`, with a minimum of  $\text{eps}^{(1/3)}$ . The default for `FunValCheck` is `off`.

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` name-value pair input argument. This function lets you monitor the status of fitting.

---

---

**Tip** To simultaneously fit data from multiple dose levels, use the `InitEstimates` input argument and set the `REParamsSelect` name-value pair input argument to a 1-by- $n$  logical vector, with all entries set to `false`, where  $n$  equals the number of fixed effects.

---

## Output Arguments

### results

Structure containing these fields:

- `FixedEffects` — A dataset array containing estimated fixed effects, including standard errors.

- **RandomEffects** — A dataset array containing sampled random effects for each group in the observed data in *pkDataObject*.
- **IndividualParameterEstimates** — A dataset array containing estimated parameter values for individuals, including random effects.
- **PopulationParameterEstimates** — A dataset array containing estimated parameter values for the population, without random effects.
- **RandomEffectCovarianceMatrix** — A dataset array containing the estimated covariance matrix of the random effects.
- **EstimatedParameterNames** — Cell array of strings specifying names of the estimated parameters.
- **CovariateNames** — Cell array of strings specifying names of the covariates in *CovModelObj*.
- **FixedEffectsStruct** — Structure containing the values of the estimated fixed effects.
- **stats** — Structure containing information such as AIC, BIC, and weighted residuals. For details on the fields in this structure, see the **stats** structure in `nlmefit` in the Statistics Toolbox documentation. However, the fields in the **stats** structure returned by `sbionlmeffit` vary slightly from those returned by `nlmefit`, namely:
  - **ires**, **pres**, **iwres**, **pwres**, and **cwres** each contain a matrix of raw or weighted residuals, with the number of columns equal to the number of responses in the model.
  - The **stats** structure returned by `sbionlmeffit` includes an additional field, **Observed**. This field contains a string or cell array of strings specifying the measured responses that correspond to the columns in the matrices of the **ires**, **pres**, **iwres**, **pwres**, and **cwres** fields. The **Observed** field is the same as the **Observed** property of the `PKModelMap` input argument.

## SimData1



`SimData` object containing data from simulating the model using the estimated parameter values for individuals. This object includes observed states and logged states.

**SimDataP**

`SimData` object containing data from simulating the model using the estimated parameter values for the population. This object includes observed states and logged states.

**See Also**

`Model` object | `nlmeFit` | `PKData` object | `SimData` object | `PKModelDesign` object | `PKModelMap` object | `sbiofitstatusplot` | `sbionlinfit` | `sbionlmeFitsa`

**How To**

- “Performing Data Fitting with PKPD Models”
- “Specifying a Nonlinear, Mixed-Effects Model”
- “Specifying Parameter Transformations”

# sbionlmefitsa

---

**Purpose** Estimate nonlinear mixed effects with stochastic EM algorithm

**Syntax**

```
results = sbionlmefitsa(modelObj, pkModelMapObject,  
    pkDataObject, InitEstimates)  
results = sbionlmefitsa(modelObj, pkModelMapObject,  
    pkDataObject, CovModelObj)  
results = sbionlmefitsa(..., Name, Value)  
results = sbionlmefitsa(..., optionStruct)  
[results, SimDataI, SimDataP] = sbionlmefitsa(...)
```

## Description

---

**Note** This function requires `nlmefitsa` in Statistics Toolbox (Version 7.0 or later).

---

`results = sbionlmefitsa(modelObj, pkModelMapObject, pkDataObject, InitEstimates)` performs estimations using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm for fitting population data with the SimBiology model, `modelObj`, and returns the estimated results in the `results` structure.

`results = sbionlmefitsa(modelObj, pkModelMapObject, pkDataObject, CovModelObj)` specifies the relationship between parameters and covariates using `CovModelObj`, a `CovariateModel` object. The `CovariateModel` object also provides the parameter transformation.

`results = sbionlmefitsa(..., Name, Value)` performs estimations using the SAEM algorithm, with additional options specified by one or more `Name, Value` pair arguments.

Following is an alternative to the previous syntax:

`results = sbionlmefitsa(..., optionStruct)` specifies `optionStruct`, a structure containing fields and values, that are the name-value pair arguments accepted by `nlmefitsa`. The defaults for `optionStruct` are the same as the defaults for the name-value pair arguments used by `nlmefitsa`, with the exceptions explained in “Input Arguments” on page 2-67.

---

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

## Input Arguments

### **modelObject**

SimBiology model object used to fit observed data.

---

**Note** If using a model object containing active doses (that is, containing dose objects created using the `adddose` method, and specified as active using the `Active` property of the dose object), be aware that these active doses are ignored by the `sbionlmefitsa` function.

---

### **pkModelMapObject**

PKModelMap object that defines the roles of the model components used for estimation. For details, see `PKModelMap` object.

---

**Note** If using a PKModelMap object that specifies multiple doses, ensure each element in the `Dosed` property is unique.

---

### **pkDataObject**

PKData object that defines the data to use in fitting and the roles of the columns used for estimation. *pkDataObject* must define target data for at least two groups. For details, see `PKData` object.

---

**Note** For each subset of data belonging to a single group (as defined in the data column specified by the `GroupLabel` property), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

- These data points are not averaged, but fitted individually.
  - Different numbers of observations at different times cause some time points to be weighted more.
- 

### **InitEstimates**

Vector of initial estimates for the fixed effects. The first `P` elements of *InitEstimates* correspond to the fixed effects for each `P` element of *pkModelMapObject.Estimated*. Additional elements correspond to the fixed effects for covariate factors. The first `P` elements of *InitEstimates* are transformed as specified by the `ParamTransform` name-value pair argument (log transformed by default). For details on specifying initial estimates, see “Setting Initial Estimates”.

### **CovModelObj**

`CovariateModel` object that defines the relationship between parameters and covariates. For details, see `CovariateModel` object.

### **optionStruct**

Structure containing fields and values that are name-value pair arguments accepted by the `nlmefitsa` function. The defaults for *optionStruct* are the same as the defaults for the arguments used by `nlmefitsa`, with the exceptions noted in “Name-Value Pair Arguments” on page 2-69.

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` field. This function lets you monitor the status of fitting.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

The `sbionlmefitsa` function uses the name-value pair arguments supported by the `nlmefitsa` function.

These `nlmefitsa` name-value pair arguments are hard-coded in `sbionlmefitsa`, and therefore, you cannot set them:

- `FEParamsSelect`
- `FEConstDesign`
- `FEGroupDesign`
- `FEObsDesign`
- `REConstDesign`
- `REGroupDesign`
- `REObsDesign`
- `Vectorization`

If you provide a `CovariateModel` object as input to `sbionlmefitsa`, then these `nlmefitsa` name-value pairs are computed from the covariate model, and therefore, you cannot set them:

- `FEGroupDesign`
- `ParamTransform`
- `REParamsSelect`

You can set all other `nlmefitsa` name-value pair arguments. For details on these arguments, see the `nlmefitsa` reference page.

Be aware that the defaults for these `nlmefitsa` name-value pair arguments differ when used by `sbionlmefitsa`:

## **FEGroupDesign**

Numeric array specifying the design matrix for each group. For details, see “Specifying a Nonlinear, Mixed-Effects Model”.

**Default:** `repmat(eye(P), [1 1 nGroups])`, where `P` = the number of estimated parameters, and `nGroups` = the number of groups in the observed data.

## **ParamTransform**

Vector of integers specifying how the parameters are distributed. For details, see “Specifying Parameter Transformations”.

---

**Note** Do not use the `ParamTransform` option to specify parameter transformations when providing a `CovariateModel` object to a fitting function. The `CovariateModel` object provides the parameter transformation.

---

**Default:** Vector of ones, which specifies all parameters are log transformed.

## **OptimFun**

String specifying the optimization function used in maximizing the likelihood.

**Default:** `fminunc`, if you have Optimization Toolbox installed. Otherwise, the default is `fminsearch`.

## **Options**

Structure containing multiple fields, including `DerivStep`, a scalar or vector specifying the relative difference used in the finite difference gradient calculation, and `FunValCheck`, a logical specifying whether to check for invalid values, such as NaN or Inf, from `modelfun`.

**Default:** The default for `DerivStep` is the lesser of  $1e-4$ , or the value of the `SolverOptions.RelativeTolerance` property of the configuration set associated with `mode1Obj`, with a minimum of  $\text{eps}^{(1/3)}$ . The default for `FunValCheck` is `off`.

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` name-value pair input argument. This function lets you monitor the status of fitting.

---

## Output Arguments

### results

Structure containing these fields:

- `FixedEffects` — A dataset array containing estimated fixed effects, including standard errors.
- `RandomEffects` — A dataset array containing sampled random effects for each group in the observed data in `pkDataObject`.
- `IndividualParameterEstimates` — A dataset array containing estimated parameter values for individuals, including random effects.
- `PopulationParameterEstimates` — A dataset array containing estimated parameter values for the population, without random effects.
- `RandomEffectCovarianceMatrix` — A dataset array containing the estimated covariance matrix of the random effects.
- `EstimatedParameterNames` — Cell array of strings specifying names of the estimated parameters.
- `CovariateNames` — Cell array of strings specifying names of the covariates in `CovMode1Obj`.

- `FixedEffectsStruct` — Structure containing the values of the estimated fixed effects.
- `stats` — Structure containing information such as AIC, BIC, and weighted residuals. For details on the fields in this structure, see the `stats` structure in `nlmefitsa` in the Statistics Toolbox documentation. However, the fields in the `stats` structure returned by `sbionlmefitsa` vary slightly from those returned by `nlmefitsa`, namely:
  - `ires`, `pres`, `iwres`, `pwres`, and `cwres` each contain a matrix of raw or weighted residuals, with the number of columns equal to the number of responses in the model.
  - The `stats` structure returned by `sbionlmeffit` includes an additional field, `Observed`. This field contains a string or cell array of strings specifying the measured responses that correspond to the columns in the matrices of the `ires`, `pres`, `iwres`, `pwres`, and `cwres` fields. The `Observed` field is the same as the `Observed` property of the `PKModelMap` input argument.

## **SimDataI**

`SimData` object containing data from simulating the model using the estimated parameter values for individuals. This object includes observed states and logged states.

## **SimDataP**

`SimData` object containing data from simulating the model using the estimated parameter values for the population. This object includes observed states and logged states.

## **See Also**

`Model` object | `nlmefitsa` | `PKData` object | `SimData` object | `PKModelDesign` object | `PKModelMap` object | `sbiofitstatusplot` | `sbionlinfit` | `sbionlmeffit`

## **How To**

- “Performing Data Fitting with PKPD Models”
- “Specifying an Error Model”



- “Specifying a Nonlinear, Mixed-Effects Model”
- “Specifying Parameter Transformations”

# sbionmfiledef

---

**Purpose** NONMEM file definition object for sbionmimport

**Syntax**  
`nmdefObj = sbionmfiledef`  
`nmdefObj = sbionmfiledef('PropertyName', PropertyValue)`

**Description** `nmdefObj = sbionmfiledef` creates a NONMEM file definition object. The NONMEM file definition object contains properties for specifying the NONMEM data items such as group, time, and dependent variable. The NONMEM file definition object lets you configure the properties to the column heading or the index of the column. Use the NONMEM file definition object in conjunction with the `sbionmimport` function to import NONMEM formatted files for use in fitting.

`nmdefObj = sbionmfiledef('PropertyName', PropertyValue)` accepts one or more comma-separated property name/value pairs. Specify *PropertyName* inside single quotes. To see the default interpretations for NONMEM formatted files see “Support for Importing NONMEM Formatted Files” in the SimBiology documentation.

- Tips**
- Use `sbionmfiledef` with `sbionmimport` if you want to apply NONMEM interpretation of headers, and the data file has column header labels different from the table shown in “Support for Importing NONMEM Formatted Files”
  - Use `sbionmimport` if the data file has column header labels identical to the table shown in “Support for Importing NONMEM Formatted Files”.

**Input Arguments**

**Filename**

If *Filename* extension is `.xls` or `.xlsx` it is assumed to be an Excel® file, otherwise it is assumed to be a text file. `sbionmfiledef` file reads the file using the `dataset` constructor.

**Property Name/Value Pairs**

**CompartmentLabel**

Identifies the column in the NONMEM formatted file that contains the compartment. Specify the header name as a `char` string or specify the index number of the header. During import the `sbionmimport` function uses the information in the column to interpret which compartment receives a dose or measured an observation. The `EventIDLabel` property specifies whether the value is a dose or an observation.

**Default:** ''

### **ContinuousCovariateLabels**

Identifies the column in the NONMEM formatted file that contains continuous covariates. Specify the header name as a `char` string or specify the index number of the header.

**Default:** {}

### **DateLabel**

Identifies the column in the NONMEM formatted file that contains the date. Specify the header name as a `char` string or specify the index number of the header. During import the `sbionmimport` function uses the information in the column to interpret time information for each dose, response and covariate measurement.

**Default:** ''

### **DependentVariableLabel**

Identifies the column in the NONMEM formatted file that contains observations. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

### **DoseLabel**

Identifies the column in the NONMEM formatted file that contains the dosing information. Specify the header name as a char string or specify the index number of the header.

**Default:** ''

## **DoseIntervallLabel**

Identifies the column in the NONMEM formatted file that contains the time between doses. Specify the header name as a char string or specify the index number of the header.

**Default:** ''

## **DoseRepeatLabel**

Identifies the column in the NONMEM formatted file that contains the number of times (excluding the initial dose) that the dose is repeated. Specify the header name as a char string or specify the index number of the header.

**Default:** ''

## **EventIDLabel**

Identifies the column in the NONMEM formatted file that contains the event identification specifying whether the value is a dose, observation, or covariate. Specify the header name as a char string or specify the index number of the header.

**Default:** ''

## **GroupLabel**

Identifies the column in the NONMEM formatted file that contains the Group ID. Specify the header name as a char string or specify the index number of the header.

**Default:** ''

## **MissingDependentVariableLabel**

Identifies the column in the NONMEM formatted file that contains information about whether a row contains an observation event (0), or not (1). Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

## **RateLabel**

Identifies the column in the NONMEM formatted file that contains the rate of infusion. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

## **TimeLabel**

Identifies the column in the NONMEM formatted file that contains the time or date of observation. During import the `sbionmimport` function uses this information to interpret when a dose was given, an observation or covariate measurement recorded. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

## **Type**

Identifies the object as 'NMFileDef', (Read-only).

## **Output Arguments**

### **nmdefObj**

Defines the meanings of the file column headings. It contains properties for specifying data items such as group, time and date. `TimeLabel` and `DependentVariableLabel` must be specified.

## **Examples**

Configure a NONMEM file definition object and import data from a NONMEM formatted file.

# sbionmfiledef

---

```
% Configure a NMFileDef object.
def = sbionmfiledef;
def.CompartmentLabel      = 'CPT';
def.DoseLabel             = 'AMT';
def.DoseIntervalLabel    = 'II';
def.DoseRepeatLabel       = 'ADDL';
def.GroupLabel            = 'ID';
def.TimeLabel             = 'TIME';
def.DependentVariableLabel = 'DV';
def.EventIDLabel          = 'EVID';

filename = 'C:\work\datafiles\dose.xls';
ds = sbionmimport(filename, def);
```

**See Also**            sbionmimport

**How To**            • “Importing Data”

## Purpose

Import NONMEM-formatted data

## Syntax

```

simbioDataset = sbionmimport('Filename')
simbioDataset = sbionmimport (nmds)
simbioDataset = sbionmimport('Filename', nmdefObj)
simbioDataset = sbionmimport('Filename', nmdefObj,
    'ParameterName', ParameterValue)
simbioDataset = sbionmimport(nmds, nmdefObj)
[simbioDataset, PKDataObj] = sbionmimport(...)

```

## Description

`simbioDataset = sbionmimport('Filename')` or `simbioDataset = sbionmimport (nmds)` converts a NONMEM formatted file, and assumes that the file is configured to use the following default values for column headers: ADDL, AMT, CPT, DATE, DAT1, DAT2, or DAT3, DV, EVID, ID, II, MDV, RATE, TIME. See “Support for Importing NONMEM Formatted Files” in the SimBiology documentation for more information on each of the headers.

`simbioDataset = sbionmimport('Filename', nmdefObj)` imports a NONMEM formatted file named *Filename*, into a SimBiology formatted data set, *simbioDataset*, using the meanings of the file column headings defined in the NONMEM file definition object (*nmdefObj*).

`simbioDataset = sbionmimport('Filename', nmdefObj, 'ParameterName', ParameterValue)` accepts one or more comma-separated parameter name/value pairs that are accepted by the `dataset` function. If `dataset` requires additional information to read the file such as the delimiter, specify these as property value pairs in `sbionmimport`. See `dataset` in the Statistics Toolbox documentation for a list of supported property value pairs.

`simbioDataset = sbionmimport(nmds, nmdefObj)` converts a NONMEM formatted dataset array (*nmds*), into a SimBiology formatted data set.

`[simbioDataset, PKDataObj] = sbionmimport(...)` returns a PKData object, *PKDataObj* containing the data set *simbioDataset*. The *PKDataObj* properties show the labels specified in *simbioDataset*.

# sbionmimport

---

## Input Arguments

### Filename

If extension of *Filename* is .xls or .xlsx, sbionmimport assumes it to be an Excel file. Otherwise sbionmimport assumes *Filename* is a text file. sbionmimport reads the file using the dataset constructor.

### nmds

*nmds* is a NONMEM formatted dataset object, in other words a NONMEM formatted file imported using the dataset command. For more information see dataset in the Statistics Toolbox documentation.

### nmdefObj

*nmdefObj* defines the meanings of the file column headings. *nmdefObj* is a NONMEM file definition object created using the sbionmfiledef function. It contains properties for specifying data items such as group, time, and date. You must specify the TimeLabel and the DependentVariableLabel properties.

## Output Arguments

### simbioDataset

The SimBiology formatted data set uses a separate column for each dose and observation. The Description property of *simbioDataset* contains a list of warnings that occurred while constructing *simbioDataset*. To view the warnings enter: `get (simbioDataset, 'Description')`.

### PkDataObj

The PKData object defines the data to use in fitting and the roles of the columns used for estimation. For more information, see PKData object.

## See Also

sbionmfiledef

## How To

- “Importing Data”



## Purpose

Perform parameter estimation

## Syntax

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

## Arguments

<i>k</i>	Vector of estimated parameter values. For all optimization methods except 'fminsearch', the parameters are constrained to be greater than or equal to zero.
<i>result</i>	Structure with fields that provide information about the progress of optimization.
<i>modelObj</i>	SimBiology model object.
<i>tspan</i>	<i>n</i> -by-1 vector representing the time span of the target data <i>xtarget</i> .
<i>xtarget</i>	<i>n</i> -by- <i>m</i> matrix, where <i>n</i> is the number of time samples and <i>m</i> is the number of states to match during the simulation. The number of rows in <i>xtarget</i> must equal the number of rows in <i>tspan</i> .

*observed\_array* Either of the following:

- Array of species objects or nonconstant parameter objects in *modelObj*, whose values should be matched during the estimation process
- Cell array of names of species objects or nonconstant parameter objects in *modelObj*, whose values should be matched during the estimation process

---

**Note** If duplicate names exist for any species or parameters, ensure there are no ambiguities by specifying either an array of objects or a cell array of qualified names, such as *compartmentName.speciesName* or *reactionName.parameterName*. For example, for a species named sp1 that is in a compartment named comp2, the qualified name is comp2.sp1.

---

The length of *observed\_array* must equal the number of columns in *xtarget*. *sbioparamestim* assumes that the order of elements in *observed\_array* is the same as the order of columns in *xtarget*.

*estimated\_array* Either of the following:

- Array of objects (compartment, species, or parameter) in *modelObj* whose initial values should be estimated
- Cell array of names of objects (compartment, species, or parameter) in *modelObj* whose initial values should be estimated

---

**Note** If duplicate names exist for any compartments, species, or parameters, ensure there are no ambiguities by specifying either an array of objects or a cell array of qualified names, such as *compartmentName.speciesName* or *reactionName.parameterName*. For example, for a parameter named *param1* scoped to a reaction named *reaction1*, the qualified name is *reaction1.param1*.

---

*k0*

Numeric vector containing the initial values of compartments, species, or parameters to be estimated. The length of *k0* must equal that of *estimated\_array*. If you do not specify *k0*, or specify an empty vector for *k0*, then *sbioparamestim* takes initial values for compartments, species, or parameters from *modelObj*, or, if there are active variants, *sbioparamestim* uses any initial values specified in the active variants. For details about variants, see `Variant` object .

*method*

Optimization algorithm to use during the estimation process, specified by either of the following:

- String specifying one of the following functions:
  - 'fminsearch'
  - 'lsqcurvefit'
  - 'lsqnonlin'
  - 'fmincon'
  - 'patternsearch'
  - 'patternsearch\_hybrid'
  - 'ga'
  - 'ga\_hybrid'

For descriptions of how `sbioparamestim` uses the previous functions, see the following table.

- Two-element cell array, with the first element being one of the previous functions, and the second element being an options structure such as returned by `optimset`, `psoptimset`, or `gaoptimset`.

---

**Tip** Use a two-element cell array to provide your own options structure for the optimization algorithm.

---

Function	Description
fminsearch	<p>sbioparamestim uses the default options structure associated with fminsearch, except for:</p> <p>Display = 'off'  TolFun = 1e-6*(Initial value of objective function)</p> <hr/> <p><b>Note</b> 'fminsearch' is an unconstrained optimization method, which can result in negative values for parameters.</p> <hr/>
lsqcurvefit	<p>Requires Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with lsqcurvefit, except for:</p> <p>Display = 'off'  FinDiffRelStep = value of the SolverOptions.RelativeTolerance property of the configuration set associated with <i>mode1Obj</i>, with a minimum of <math>\text{eps}^{(1/3)}</math>  TolFun = 1e-6*(Initial value of objective function)  TypicalX = 1e-6*(Initial values of components to be estimated)</p>

## sbioparamestim

---

Function	Description
lsqnonlin	<p>Requires Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with lsqnonlin, except for:</p> <p>Display = 'off'</p> <p>FinDiffRelStep = value of the SolverOptions.RelativeTolerance property of the configuration set associated with <i>modelObj</i>, with a minimum of <math>\text{eps}^{(1/3)}</math></p> <p>TolFun = <math>1\text{e-}6 \times (\text{Initial value of objective function})</math></p> <p>TypicalX = <math>1\text{e-}6 \times (\text{Initial values of components to be estimated})</math></p>
fmincon	<p>Requires Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with fmincon, except for:</p> <p>Algorithm = 'interior-point'</p> <p>Display = 'off'</p> <p>FinDiffRelStep = value of the SolverOptions.RelativeTolerance property of the configuration set associated with <i>modelObj</i>, with a minimum of <math>\text{eps}^{(1/3)}</math></p> <p>TolFun = <math>1\text{e-}6 \times (\text{Initial value of objective function})</math></p> <p>TypicalX = <math>1\text{e-}6 \times (\text{Initial values of components to be estimated})</math></p>

Function	Description
patternsearch	<p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with patternsearch, except for:</p> <pre> Display = 'off' TolFun = 1e-6*(Initial value of objective function) TolMesh = 1.0e-3 Cache = 'on' MeshAccel = 'on' </pre>
patternsearch	<p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim calls the patternsearch function with the additional option SearchMethod = {@searchlhs,10,15}. This option adds an additional search step that uses Latin hypercube sampling.</p> <p>The sbioparamestim function uses the default options structure associated with patternsearch, except for:</p> <pre> Display = 'off' TolFun = 1e-6*(Initial value of objective function) TolMesh = 1.0e-3 Cache = 'on' MeshAccel = 'on' SearchMethod = {@searchlhs,10,15} </pre>

# sbioparamestim

Function	Description
ga	<p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with ga, except for:</p> <pre>Display = 'off' TolFun = 1e-6*(Initial value of objective function) PopulationSize = 10 Generations = 30 MutationFcn = @mutationadaptfeasible</pre>
ga_hybrid	<p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim calls the ga function with the additional option HybridFcn = {@fmincon, fminopt}, where fminopt is the same set of default options sbioparamestim uses for fmincon. This option causes an additional gradient-based minimization after the genetic algorithm step terminates.</p> <p>The sbioparamestim function uses the default options structure associated with ga, except for:</p> <pre>Display = 'off' TolFun = 1e-6*(Initial value of objective function) PopulationSize = 10 Generations = 30 MutationFcn = @mutationadaptfeasible HybridFcn = {@fmincon, struct of name/value pairs for fmincon}</pre>

## Description

[k, result]= sbioparamestim(*modelObj*, *tspan*, *xtarget*, *observed\_array*, *estimated\_array*) estimates the initial values of compartments, species, and parameters of *modelObj*, a SimBiology model object, specified in *estimated\_array*, so as to match the values of species and nonconstant parameters given by *observed\_array*



with the target state, *xtarget*, whose time variation is given by the time span *tspan*. If you have Optimization Toolbox installed, *sbioparamestim* uses the *lsqnonlin* function as the default method for the parameter estimation. If you do not have Optimization Toolbox installed, *sbioparamestim* uses the MATLAB function *fminsearch* as the default method for the parameter estimation.

`[...]= sbioparamestim(..., observed_array, estimated_array, k0)` specifies the initial values of compartments, species, and parameters listed in *estimated\_array*.

`[...]= sbioparamestim(..., observed_array, estimated_array, k0, method)` specifies the optimization method to use.

## Algorithms

*sbioparamestim* estimates parameters by attempting to minimize the discrepancy between simulation results and the data to fit. The minimization uses one of these optimization algorithms: *fminsearch* (from MATLAB); *lsqcurvefit*, *lsqnonlinfit*, or *fmincon* (from Optimization Toolbox); or *patternsearch* or *ga* (from Global Optimization Toolbox). All optimization methods require an objective function as an input. This objective function takes as input a vector of parameter values and returns an estimate of the discrepancy between simulation and data. When using *lsqcurvefit* or *lsqnonlinfit* as the optimization method, this objective function returns a vector of the residuals. For other optimization methods, the objective function returns the 2-norm of the residuals.

## Examples

Given a model and some target data, estimate all of its parameters without explicitly specifying any initial values:

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

```
sbioloadproject gprotein_norules m1;
```

- 2 Store the target data in a variable:

# sbioparamestim

---

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

**3** Store all model parameters in an array:

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

**4** Store the species that should match target:

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

**5** Estimate the parameters:

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

```
k =
```

```
0.1988  
0.0000  
0.0045  
6.2859  
0.0040  
0.9726  
0.0000  
0.1164
```

```
result =
```

```
          fval: 8.7248e+005  
        residual: [9x1 double]  
        exitflag: 2  
    iterations: 2  
        funccount: 27
```

```
algorithm: 'large-scale: trust-region reflective Newton'  
message: [1x77 char]
```

---

Estimate parameters specified in `p_array` and species specified in `sp_array` using different algorithms. This example uses data from the first example.

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

---

Estimate parameters specified in `p_array`, species specified in `sp_array`, and change default optimization options to use user-specified options. This example uses data from the first example.

```
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});
```

## References

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

## See Also

`sbiomodel` | `optimset` | `gaoptimset` | `psoptimset`

# sbioplot

---

**Purpose** Plot simulation results in one figure

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

## Arguments

<i>simDataObj</i>	A <code>SimData</code> object or an array of <code>SimData</code> objects, containing data from simulation of a model.
<i>fcnHandleValue</i>	Function handle.
<i>xArgsValue</i>	Cell array with the names of the states.
<i>yArgsValue</i>	Cell array with the names of the states.

## Description

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

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

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

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

The output argument `handles` is a two-dimensional array of handles to the lines plotted by the function. Each column corresponds to a run and each row corresponds to the lines being plotted for a state. `names` is a one-dimensional cell array that contains the names to be displayed on

the nodes which are children of a Run Node. The length of names should be equal to the number of rows in the `handles` array returned.

## Examples

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

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

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

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

## See Also

`sbiosubplot` | `SimData` object

# sbioremovefromlibrary

---

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

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

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

`sbioremovefromlibrary` does not remove a kinetic law definition that is being used in a model.

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

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

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

To create a kinetic law definition, unit, or unit prefix, use `sbioabstractkineticlaw`, `sbiounit`, or `sbiounitprefix` respectively.

To add a kinetic law definition, unit, or unit prefix to the user-defined library, use the function `sbioaddtolibrary`.

## Examples

This example shows how to remove a kinetic law definition from the user-defined library.

- 1 Create a kinetic law definition.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined
```

SimBiology Abstract Kinetic Law Array

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

**3** Remove the kinetic law definition.

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

## See Also

`sbioaddtolibrary` | `sbioabstractkineticlaw` | `sbiounit` | `sbiounitprefix`

# sbioreset

---

**Purpose** Delete all model objects

**Syntax** `sbioreset`

**Description** `sbioreset` deletes all SimBiology model objects at the root level. You cannot use a SimBiology model object after it is deleted.

---

**Tip** To remove a SimBiology model object from the MATLAB workspace, without deleting it from the root level, use the `clear` function.

---

---

**Note** If the SimBiology desktop is open, calling `sbioreset` at the command line deletes all model objects that are open in the desktop.

---

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

To add a kinetic law definition to the SimBiology root user-defined library, use the `sbioaddtolibrary` function. To add a unit to the SimBiology user-defined library on the root, use `sbiounit` followed by `sbioaddtolibrary`. To add a unit prefix to the SimBiology user-defined library on the root, use `sbiounitprefix` followed by `sbioaddtolibrary`.

## Examples

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

**1** Import a model into the workspace.

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

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

```
rootObj = sbioroot
```



SimBiology Root Contains:

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

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

```
clear all
```

```
rootObj
```

SimBiology Root Contains:

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

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

```
sbioreset
```

```
rootObj
```

SimBiology Root Contains:

Models:	0
---------	---

# sbioreset

---

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

[sbioaddtolibrary](#) | [sbioroot](#) | [sbiounit](#) | [sbiounitprefix](#)

## How To

- [sbioroot](#)

**Purpose** Return SimBiology root object

**Syntax** `rootObj = sbioroot`

**Arguments** `rootObj` Return sbioroot to this object.

**Description** `rootObj = sbioroot` returns the SimBiology root object to root. The SimBiology root object contains a list of the SimBiology model objects, available units, unit prefixes, and available kinetic laws.

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

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

The kinetic laws define the built-in kinetic laws and user-defined kinetic laws. See `AbstractKineticLaw` object for more information.

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

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

## Method Summary

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

## Property Summary

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

## See Also

`addkineticlaw` | `sbiomodel` | `sbioreset` | `Unit object` | `UnitPrefix object`

## How To

- `sbiomodel`
- `addkineticlaw`
- `sbioreset`

**Purpose** Save all models in root object

**Syntax**

```
sbiosaveproject projFilename
sbiosaveproject projFilename variableName
sbiosaveproject projFilename variableName1 variableName2 ...
```

**Description**

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

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

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

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

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

**Examples**

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

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

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

```
sbiosaveproject myprojectfile modelObj timeseriesObj
```

# sbiosaveproject

---

## See Also

[sbioaddtolibrary](#) | [sbioloadproject](#) | [sbioremovefromlibrary](#)  
| [sbiowhos](#)

## How To

- [sbioloadproject](#)
- [sbiowhos](#)
- [sbioaddtolibrary](#)
- [sbioremovefromlibrary](#)

**Purpose**

Search for objects with specified constraints

**Syntax**

```

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

```

**Arguments**

- Out*                                    Object or array of objects returned by the sbioselect function. *Out* might contain a mixture of object types (for example, species and parameters), depending on the selection you specify. If *PropertyValue* is a cell array, then the function returns all objects with the property '*PropertyName*' that matches any element of *PropertyValue*.
- Obj*                                    SimBiology object or array of objects to search. If an object is not specified, sbioselect searches the root.
- PropertyName*                        Any property of the object being searched.
- PropertyValue*                        Specify *PropertyValue* to include in the selection criteria.
- TypeValue*                            Type of object to include in the selection, for example, sbiomodel, species, reaction, or kineticlaw.
- Condition*                            The search condition. See the table under “Description” on page 2-104 for a list of conditions.

<i>PropertyNameCondition</i>	Search condition that applies only to property names (which are strings). See the table listing “Conditions for Properties Names or String Values” below.
<i>PropertyNamePattern</i>	String used to select the property name according to the condition imposed by <i>PropertyNameCondition</i> .
<i>DepthValue</i>	Specify the depth number to search. Valid numbers are positive integer values and <i>inf</i> . If <i>DepthValue</i> is <i>inf</i> , <i>sbioselect</i> searches <i>Obj</i> and all of its children. If <i>DepthValue</i> is 1, <i>sbioselect</i> only searches <i>Obj</i> and not its children. By default, <i>DepthValue</i> is <i>inf</i> .

## Description

*sbioselect* searches for objects with specified constraints.

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

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

*Out* = *sbioselect*(*Obj*, 'PropertyName', *PropertyValue*) returns the objects with the property name (*PropertyName*) and property value (*PropertyValue*) found in any object (*Obj*). If the property name in a property-value pair contains either a '?' or '\*', then the name is automatically interpreted as a wildcard expression, equivalent to the where clause ('Where', 'wildcard', 'PropertyName', '==', *PropertyValue*).

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



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

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

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

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

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

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

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

`Out = sbioselect(Obj, 'Where', 'PropertyNameCondition', 'PropertyNamePattern', 'Condition', PropertyValue)` finds objects with a property name that matches the pattern in (*PropertyNamePattern*) with the condition (*PropertyNameCondition*) and matches the value (*PropertyValue*) with the condition (*Condition*). Use this syntax when you want search conditions on both property names and property values.

# sbioselect

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

Conditions for Numeric Properties	Example Syntax
==	<p>Search in the model object (<code>modelObj</code>), and return parameter objects that have <code>Value</code> equal to 0.5. <code>sbioselect</code> returns parameter objects because only parameter objects have a property called <code>Value</code>.</p> <pre>parameterObj = sbioselect (modelObj,...     'Where', 'Value', '==', 0.5)</pre> <p>In the case of <code>==</code>, this is equivalent to omitting the condition as shown:</p> <pre>parameterObj = sbioselect (modelObj,...     'Value', 0.5)</pre> <p>Search in the model object (<code>modelObj</code>), and return parameter objects that have <code>ConstantValue</code> false (nonconstant parameters).</p> <pre>parameterObj = sbioselect (modelObj,...     'Where', 'ConstantValue', '==', false)</pre>
~=	<p>Search in the model object (<code>modelObj</code>), and return parameter objects that do not have <code>Value</code> equal to 0.5.</p> <pre>parameterObj = sbioselect (modelObj,...     'Where', 'Value', '~=', 0.5)</pre>

<b>Conditions for Numeric Properties</b>	<b>Example Syntax</b>
<p>&gt;,&lt;,&gt;=,&lt;=</p>	<p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) greater than 50.</p> <pre>speciesObj = sbioselect (modelObj, ...     'Where', 'InitialAmount', '&gt;', 50)</pre> <p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) less than or equal to 50.</p> <pre>speciesObj = sbioselect (modelObj,...     'Where', 'InitialAmount', '&lt;=', 50)</pre>
<p>between</p>	<p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) between 200 and 300.</p> <pre>speciesObj = sbioselect (modelObj,...     'Where', 'InitialAmount',...     'between', [200 300])</pre>
<p>-between</p>	<p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) that is not between 200 and 300.</p> <pre>speciesObj = sbioselect (modelObj,...     'Where', 'InitialAmount',...     '~between', [200 300])</pre>

<b>Conditions for Numeric Properties</b>	<b>Example Syntax</b>
equal_and_same_type	<p>Similar to ==, but in addition requires the property value to be of the same type. Search in the model object (modelObj), and return all objects containing a property of type double and a value equal to 0. (Using '==' would also select objects containing a property with a value of false.)</p> <pre>zeroObj = sbioselect(modelObj, ... 'Where', '*', 'equal_and_same_type', 0);</pre>
unequal_and_same_type	<p>Similar to ~=, but in addition requires the property value to be of the same type. Select all objects containing a property of type double and value not equal to 0. (Using '~=' would also select objects containing a property with a value of true.)</p> <pre>nonzeroObj = sbioselect(modelObj, ... 'Where', '*', 'unequal_and_same_type', 0);</pre>

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

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

# sbioselect

Conditions for Properties Names or String Values	Example Syntax
regexp. Supports expressions supported by the functions regexp and regexpi.	<p>Search in the model object (modelObj), and return objects that have 'ese' or 'ase' anywhere within the name.</p> <pre>Obj = sbioselect (modelObj, 'Where', ...   'Name', 'regexp', '[ea]se')</pre> <p>Search in the root, and return objects that have kinase anywhere within the name.</p> <pre>Obj = sbioselect ('Where', ...   'Name', 'regexp', 'kinase')</pre> <p>Note that this query could result in a mixture of object types (for example, species and parameters).</p>
regexpi	Same as regexp; in addition, this is case insensitive.
~regexp	<p>Search in the model object (modelObj), and return objects that do not have kinase anywhere within the name.</p> <pre>Obj = sbioselect (modelObj, 'Where', ...   'Name', '~regexp', 'kinase')</pre>
~regexpi	Same as ~regexp; in addition, this is case insensitive.
wildcard	Supports DOS-style wildcards ('?' matches any single character, '*' matches any number of characters, and the pattern must match the entire string). See <a href="#">regexptranslate</a> for more information.
wildcardi	Same as wildcard; in addition, this is case insensitive.

<b>Conditions for Properties Names or String Values</b>	<b>Example Syntax</b>
-wildcard	<p>Search in the model object (modelObj), and return objects that have names that do not begin with kin*.</p> <pre>Obj = sbioselect (modelObj, 'Where', ...     'Name', '~wildcard', 'kin*')</pre>
-wildcardi	Same as -wildcard; in addition, this is case insensitive.

# sbioselect

Use the condition type function for any property. The specified value should be a function handle that, when applied to a property value, returns a boolean indicating whether there is a match. The following table shows an example of using function.

Condition	Example Syntax
'function'	<p>Search in the model object and return reaction objects whose Stoichiometry property contains the specified stoichiometry.</p> <pre>Out = sbioselect(modelObj, 'Where', ...   'Stoichiometry', 'function', ...   @(x)any(x&gt;2))</pre> <p>Select all objects with a numeric value that is even.</p> <pre>iseven = @(x) isnumeric(x)...   &amp;&amp; isvector(x) &amp;&amp; mod(x, 2) == 0; evenValuedObj = sbioselect(modelObj, ...   'where', 'Value', 'function', iseven);</pre>

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

Condition	Example Syntax
'contains'	<p>Search in the model object and return reaction objects whose Reactant property contains the specified species.</p> <pre>Out = sbioselect(modelObj, 'Where', ...   'Reactants', 'contains', ...   modelObj.Species(1))</pre>

```
Out = sbioselect(Obj, 'Where', 'PropertyName1', 'Condition1',  
  PropertyValue1, 'Where', 'PropertyName2', 'Condition2',
```



*PropertyValue2,...*) finds objects contained by *Obj* that matches all the conditions specified.

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

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

## Examples

- 1 Import a model.

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

- 2 Find and return an object named pA.

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

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

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

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

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

SimBiology Species Array

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

- 5 Find and return objects that do not have their units set.

```
unitlessObj = sbioselect(modelObj, 'Where', 'wildcard', '*Units', '==', '');
%Alternatively,
unitlessObj = sbioselect(modelObj, '*Units', '');
```

# sbioselect

---

## See Also

`regexp`

**Purpose** Show unit prefixes in library

**Syntax**

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

## Arguments

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

**Description** sbioshowunitprefixes returns information about unit prefixes in the SimBiology library.

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

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

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

# sbioshowunitprefixes

---

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

## Examples

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

## See Also

[sbioconvertunits](#) | [sbioshowunits](#) | [sbiounitprefix](#)

**Purpose** Show units in library

**Syntax**

```

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

## Arguments

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

# sbioshowunits

---

## Description

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

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

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

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

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

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

## Examples

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

## See Also

`sbioconvertunits` | `sbioshowunitprefixes` | `sbiunit`

**Purpose**

Simulate model object

**Syntax**

```
[t,x, names] = sbiosimulate(modelObj)
simDataObj = sbiosimulate(modelObj)
... = sbiosimulate(modelObj, configsetObj)
... = sbiosimulate(modelObj, variantObj)
... = sbiosimulate(modelObj, doseObj)
... = sbiosimulate(modelObj, configsetObj, variantObj)
... = sbiosimulate(modelObj, configsetObj, doseObj)
... = sbiosimulate(modelObj, configsetObj, variantObj,
    doseObj)
```

**Description**

[t,x, names] = sbiosimulate(*modelObj*) simulates *modelObj*, a SimBiology model object, using the active configuration set associated with *modelObj*, and returns the simulation results in three outputs, described in “Output Arguments” on page 2-121.

*simDataObj* = sbiosimulate(*modelObj*) returns the simulation results to *simDataObj*, a SimData object.

... = sbiosimulate(*modelObj*, *configsetObj*) uses *configsetObj*, a configuration set object, thus overriding the active configuration set associated with *modelObj*. After the command is executed, this override does not exist. The configuration set that is defined as active is reinstated. To get the configuration sets attached to a model, use `getConfigset`. To attach a new or existing configuration set to a model, use `addconfigset`. To set the active configuration set of a model, use `setactiveconfigset`. For more information about configuration sets, see `Configset` object.

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

... = sbiosimulate(*modelObj*, *doseObj*) simulates *modelObj*, a SimBiology model object, using *doseObj*, a dose object or an array of dose objects.

# sbiosimulate

---

`... = sbiosimulate(modelObj, configsetObj, variantObj)`  
simulates *modelObj*, a SimBiology model object, using *configsetObj*, a configuration set object or [], an empty array; and *variantObj*, a variant object, array of variant objects, or [], an empty array.

`... = sbiosimulate(modelObj, configsetObj, doseObj)`  
simulates *modelObj*, a SimBiology model object, using *configsetObj*, a configuration set object or [], an empty array; and *doseObj*, a dose object, array of dose objects, or [], an empty array.

`... = sbiosimulate(modelObj, configsetObj, variantObj, doseObj)` simulates *modelObj*, a SimBiology model object, using *configsetObj*, a configuration set object or [], an empty array; *variantObj*, a variant object, array of variant objects, or [], an empty array; and *doseObj*, a dose object, array of dose objects, or [], an empty array.

## Input Arguments

### **modelObj**

SimBiology Model object.

### **configsetObj**

Configset object to use in the simulation. When there are three or more inputs, *configsetObj* can also be [], an empty array.

---

**Note** If your model contains events, the Configset object cannot specify 'exptau' or 'imptau' for the SolverType property.

---

---

**Note** If your model contains doses, the Configset object cannot specify 'ssa', 'exptau', or 'imptau' for the SolverType property.

---



---

**Tip** Set *configsetObj* to [], an empty array, if you want to specify both a variant and a dose, and use the active configuration set.

---

### **variantObj**

Variant object or array of variant objects to apply to the model during the simulation. When there are three or more inputs, *variantObj* can also be [], an empty array.

### **doseObj**

Dose object or array of dose objects to apply to the model during the simulation. For more information about dose objects, see `ScheduleDose` object and `RepeatDose` object. When there are three or more inputs, *doseObj* can also be [], an empty array.

## **Output Arguments**

### **t**

An n-by-1 vector of time points, showing the simulation time steps.

### **x**

An n-by-m data array, where n is the number of time samples and m is the number of states logged in the simulation. Each column of x describes the variation in the quantity of a state over time.

### **names**

An m-by-1 cell array of names. If the species are in multiple compartments, species names are qualified with the compartment name in the form `compartmentName.speciesName`. For example, `nucleus.DNA`, `cytoplasm.mRNA`.

Parameter names are qualified with the reaction name if the parameter is scoped to the reaction's kinetic law. For example, `Transcription.k1`, denotes that the parameter k1 is scoped to the kinetic law for the reaction `Transcription`.

## **simdataObj**

SimData object, which holds time and state data as well as metadata, such as the types and names for the logged states or the configuration set used during simulation. You can access time, data, and names stored in *simdataObj* by using properties of a SimData object.

## **Examples**

The following examples show how to change solver settings.

### **Example 1**

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

- 1** Read the file for the oscillator model.

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

- 2** Get the active configuration set for the model.

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

- 3** Set SolverType to ode23t and set StopTime to 10.

```
set(configsetObj, 'SolverType', 'ode23t');  
set(configsetObj, 'StopTime', 10);
```

- 4** Simulate the model.

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

- 5** Plot the results of the simulation.

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

### **Example 2**

Simulate the above example without dimensional analysis (DimensionalAnalysis property set to false).

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

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

- 2 Simulate the model.

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

- 3 Plot the results of the simulation.

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

## See Also

`addconfigset` | `sbioaccelerate` | `sbiomodel` | `SimData` object | `Configset` object | `getconfigset` | `setactiveconfigset` | `Variant` object | `ScheduleDose` object | `RepeatDose` object | `Model` object

# sbiosubplot

---

**Purpose** Plot simulation results in subplots

**Syntax**

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

## Arguments

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

## Description

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

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

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

## Examples

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

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

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

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

**See Also**      sbioplot

# sbiotrellis

---

## Purpose

Plot data or simulation results in trellis plot

## Syntax

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

## Arguments

<i>trellisplot</i>	Object returned by <code>sbiotrellis</code> . Use <code>trellisplot</code> together with the <code>plot</code> method to overlay trellis plots. See “Description” on page 2-127 for information about the <code>plot</code> method.
<i>Dataset</i>	A dataset array containing grouped data to plot.

---

**Tip** You create a `dataset` array from your data using the `dataset` constructor or the `sbionmimport` function.

---

<i>GroupCol</i>	String specifying a column in <i>Dataset</i> that contains groups.
<i>XCol</i>	String specifying a column in <i>Dataset</i> to plot on the <i>x</i> -axis.
<i>YCol</i>	String or cell array of strings specifying column(s) in <i>Dataset</i> to plot on the <i>y</i> -axis.
<i>fcnHandle</i>	Function handle. If '' (empty), default is <code>@plot</code> .
<i>simDataObj</i>	SimBiology data object.

**Description**

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

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

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

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

Use the `plot` method to overlay a *SimData* object or a dataset on an existing `sbiotrellis` plot. The command, `plot(trellisplot, ...)` adds a plot to the trellis plot defined by the `sbiotrellis` object, `trellisplot`. The *SimData* or dataset object that is being plotted must have the same number of elements/groups as the trellis plot. The `plot` method has the same input arguments as `sbiotrellis`. For an example, see “Performing Population Fitting” in the *SimBiology* documentation.

**See Also**

`sbioplot` | `sbiosubplot`

**Purpose** Create user-defined unit

**Syntax**

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

## Arguments

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



<i>PropertyName</i>	Name of the unit object property, for example, 'Notes'.
<i>PropertyValue</i>	Value of the unit object property, for example, 'New unit for GPCR model'.

## Description

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

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

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

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

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

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

```
sbioaddtolibrary(unitObject);
```

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

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

## Examples

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

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

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

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

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

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

```
rootObj = sbioroot;
```

```
rootObj.UserDefinedLibrary.Units
```

```
SimBiology UserDefined Units
```

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

Alternatively, use the `sbiowhos` command.

```
sbiowhos -userdefined -unit
```

```
SimBiology UserDefined Units
```

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

**See Also**

[sbioaddtolibrary](#) | [sbioshowunits](#) | [sbiunitprefix](#) | [sbiowhos](#)

# sbiunitcalculator

---

<b>Purpose</b>	Convert value between units
<b>Syntax</b>	<code>result = sbiunitcalculator('fromUnits', 'toUnits', Value)</code>
<b>Description</b>	<code>result = sbiunitcalculator('fromUnits', 'toUnits', Value)</code> converts the value, <i>Value</i> , which is defined in the units, <i>fromUnits</i> , to the value, <i>result</i> , which is defined in the units, <i>toUnits</i> .
<b>Examples</b>	<code>result = sbiunitcalculator('mile/hour', 'meter/second', 1)</code>
<b>See Also</b>	<code>sbioshowunits</code>

**Purpose** Create user-defined unit prefix

**Syntax**

```

unitprefixObject = sbiunitprefix('NameValue')
unitprefixObject = sbiunitprefix('NameValue',
    'ExponentValue')
unitprefixObject = sbiunitprefix('NameValue',
    ...'PropertyName', PropertyValue ...)

```

## Arguments

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

## Description

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

*unitprefixObject* = `sbiunitprefix('NameValue', 'ExponentValue')` creates a unit-prefix object with a multiplicative factor of  $10^{\text{ExponentValue}}$ .

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

# sbiounitprefix

---

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

```
sbioaddtolibrary(unitprefixObject);
```

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

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

## Examples

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

- 1 Create a unit prefix.

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

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

```
sbioaddtolibrary(unitprefixObj1);
```

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

```
rootObj = sbioroot;
```

```
rootObj.UserDefinedLibrary.UnitPrefixes
```

```
Unit Prefix Array
```

Index:	Library:	Name:	Exponent:
1	UserDefined	peta	15

Alternatively, use the `sbiowhos` command.

```
sbiowhos -userdefined -unitprefix
```

## SimBiology UserDefined Unit Prefixes

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

### See Also

[sbioaddtolibrary](#) | [sbioshowunits](#) | [sbiounit](#) | [sbiowhos](#)

### How To

- [sbioshowunits](#)

**Purpose** Construct variant object

**Syntax**

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

## Arguments

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

## Description

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

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

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

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

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



---

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

---

**Method  
Summary**

<code>addcontent</code> (variant)	Append content to variant object
<code>commit</code> (variant)	Commit variant contents to model
<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>get</code> (any object)	Get object properties
<code>rmcontent</code> (variant)	Remove contents from variant object
<code>set</code> (any object)	Set object properties
<code>verify</code> (model, variant)	Validate and verify SimBiology model

**Property  
Summary**

<code>Active</code>	Indicate object in use during simulation
<code>Content</code>	Contents of variant object
<code>Name</code>	Specify name of object
<code>Notes</code>	HTML text describing SimBiology object
<code>Parent</code>	Indicate parent object

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

## Examples

- 1 Create a variant object.

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

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

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

## See Also

[addvariant](#) | [copyobj](#) | [getvariant](#)

**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 kinetic laws, units, and unit prefixes.

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

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

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

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

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

**Examples**

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

# sbiowhos

---

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

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

## See Also

whos

**Purpose**

Export SimBiology model to SBML file

**Syntax**

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

**Arguments**

*modelObj* Model object. Enter a variable name for a model object.

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

**Description**

`sbmlexport(modelObj)` exports a SimBiology model object (`modelObj`) to a file with a Systems Biology Markup Language (SBML) Level 2 Version 4 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.

For more information, see “Importing from SBML Files”.

**Examples**

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

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

**References**

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

# sbmlexport

---

## See Also

[sbiomodel](#) | [sbiosaveproject](#) | [sbmlimport](#)

## How To

- [sbmlimport](#)
- [sbiomodel](#)
- [sbiosaveproject](#)

<b>Purpose</b>	Import SBML-formatted file
<b>Syntax</b>	<code>modelObj = sbmlimport(<i>File</i>)</code>
<b>Description</b>	<p><code>modelObj = sbmlimport(<i>File</i>)</code> imports <i>File</i>, a Systems Biology Markup Language (SBML)-formatted file, into MATLAB and creates a model object <code>modelObj</code>.</p> <p><i>File</i> is a string specifying a file name or a path and file name supported by your operating system. <i>File</i> extensions are <code>.sbml</code> or <code>.xml</code>. <i>File</i> can also be a URL, if you have the Java™ programming language.</p> <p><code>sbmlimport</code> supports SBML Level 2 Version 4 and earlier.</p> <p>For functional characteristics and limitations, see “Importing from SBML Files”.</p>
<b>Input Arguments</b>	<p><b>File</b></p> <p>String specifying either of the following:</p> <ul style="list-style-type: none"><li>• File name or path and file name supported by your operating system</li><li>• URL (if you have Java programming language)</li></ul>
<b>Examples</b>	<p>Import SBML model:</p> <pre>sbmlObj = sbmlimport('oscillator.xml');</pre>
<b>References</b>	<p>Finney, A., Hucka, M., (2003). <i>Systems Biology Markup Language (SBML) Level 2: Structures and facilities for model definitions</i>. SBML.org.</p>
<b>Alternatives</b>	<p>Use the SimBiology desktop to import an SBML-formatted file. For more information, see “Importing and Exporting Models, Tasks, and Data from the Desktop” and “Importing from SBML Files”.</p>
<b>See Also</b>	<code>get</code>   <code>sbiosimulate</code>   <code>sbmlexport</code>   <code>set</code>

# sbmlimport

---

## **How To**

- “Importing and Exporting Models, Tasks, and Data from the Desktop”
- “Importing from SBML Files”



<b>Purpose</b>	Open SimBiology desktop for modeling and simulation				
<b>Syntax</b>	<pre>simbiology simbiology(<i>modelObj</i>) simbiology(<i>File</i>)</pre>				
<b>Input Arguments</b>	<table> <tr> <td><i>modelObj</i></td> <td>SimBiology model object or an array of model objects.</td> </tr> <tr> <td><i>File</i></td> <td>String specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td> </tr> </table>	<i>modelObj</i>	SimBiology model object or an array of model objects.	<i>File</i>	String specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.
<i>modelObj</i>	SimBiology model object or an array of model objects.				
<i>File</i>	String specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.				
<b>Description</b>	<p>simbiology opens the SimBiology desktop, which lets you:</p> <ul style="list-style-type: none"> <li>• Build a SimBiology model by representing reaction pathways and entering kinetic data for the reactions.</li> <li>• Import or export SimBiology models to and from the MATLAB workspace or from a Systems Biology Markup Language (SBML) file.</li> <li>• Modify an existing SimBiology model.</li> <li>• Simulate a SimBiology model using individual or ensemble runs.</li> <li>• View results from the simulation.</li> <li>• Perform analysis tasks such as sensitivity analysis, parameter and species scans, and calculation of conserved moieties.</li> <li>• Import and plot data for analysis tasks.</li> <li>• Create and/or modify user-defined units and unit prefixes.</li> <li>• Create and/or modify user-defined kinetic laws.</li> </ul> <p>simbiology(<i>modelObj</i>) opens the SimBiology desktop with <i>modelObj</i>, a SimBiology model object. If a project is open in the desktop, the simbiology function adds <i>modelObj</i> to the project.</p>				

`simbiology(File)` opens the project specified by *File* in the SimBiology desktop. *File* is a string specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. If a project is open in the desktop, the software replaces it with the new project, after prompting you to save any changes.

The `Parent` property of a SimBiology model object is set to the SimBiology root object. The root object contains a list of model objects that are accessible from the MATLAB command line and from the SimBiology desktop. Because both the command line and the desktop point to the same model object in the `Root` object, changes you make to the model at the command line are reflected in the desktop, and vice versa.

---

**Note** The `sbioreset` function removes all models from the root object. Therefore, the `sbioreset` function removes all models from the SimBiology desktop.

---

## Examples

Create a SimBiology model in the MATLAB workspace, and then open the SimBiology desktop with the model:

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

## See Also

`sbiroot` | `sbiofittool`

**Superclasses** matlab.mixin.Heterogeneous

**Purpose** Exported SimBiology model dose object

**Description** SimBiology.export.Dose is the superclass for modifiable export dose objects. An export dose is either of subclass SimBiology.export.RepeatDose or SimBiology.export.ScheduleDose.

**Construction** Export dose objects are created by the `export` method for SimBiology models. By default, all active doses are export doses, but you can specify which doses to export using the optional `editdoses` input argument to `export`.

`export(model)` Export SimBiology model

## Properties

### Amount

Amount of dose, a nonnegative scalar value.

### AmountUnits

Dose amount units. This property is read only.

### DurationParameterName

Parameter specifying length of time. This property is read only.

### LagParameterName

Parameter specifying time lag for the dose. This property is read only.

### Name

Name of dose object. This property is read only.

### Notes

Text describing dose object. This property is read only.

### Parent

# SimBiology.export.Dose

---

Name of the parent export model. This property is read only.

## **Rate**

Rate of dose, a nonnegative scalar value.

## **RateUnits**

Units for dose rate. This property is read only.

## **TargetName**

Species receiving dose. This property is read only.

## **TimeUnits**

Time units for dosing. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## **Examples**

### **Exported SimBiology Model Dose Objects**

Open a sample SimBiology model project, and export the included model object.

```
sbioloadproject('AntibacterialPKPD')  
em = export(m1);
```

Get the editable doses from the exported model object.

```
doses = getdose(em);
```

```
doses =
```

```
1x4 SimBiology.export.RepeatDose handle  
Package: SimBiology.export
```

```
Properties:
```

```
Interval
```

```
RepeatCount
```

```
StartTime
TimeUnits
Amount
AmountUnits
DurationParameterName
LagParameterName
Name
Notes
Parent
Rate
RateUnits
TargetName
```

Methods, Events, Superclasses

The exported model has 4 repeated dose objects.

Display the 3rd dose object from the exported model object.

```
doses(3)
```

```
ans =
```

```
SimBiology.export.RepeatDose handle
Package: SimBiology.export
```

```
Properties:
```

```
    Interval: 12
    RepeatCount: 27
    StartTime: 0
    TimeUnits: 'hour'
    Amount: 500
    AmountUnits: 'milligram'
    DurationParameterName: 'TDose'
    LagParameterName: ''
    Name: '500 mg bid'
    Notes: ''
    Parent: 'Antibacterial'
```

# SimBiology.export.Dose

---

```
Rate: 0
RateUnits: ''
TargetName: 'Central.Drug'
```

Methods, Events, Superclasses

Change the dosing amount for this dose object.

```
doses(3).Amount = 600;
doses(3)
```

```
ans =
```

```
SimBiology.export.RepeatDose handle
Package: SimBiology.export
```

Properties:

```
Interval: 12
RepeatCount: 27
StartTime: 0
TimeUnits: 'hour'
Amount: 600
AmountUnits: 'milligram'
DurationParameterName: 'TDose'
LagParameterName: ''
Name: '500 mg bid'
Notes: ''
Parent: 'Antibacterial'
Rate: 0
RateUnits: ''
TargetName: 'Central.Drug'
```

Methods, Events, Superclasses

## See Also

```
SimBiology.export.RepeatDose | SimBiology.export.ScheduleDose
| export
```

## **Related Examples**

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”





# SimBiology.export.ExplicitTauSimulationOptions

---

Simulation time criterion to stop simulation, a nonnegative scalar value.

## **TimeUnits**

Time units for simulation. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## **See Also**

`SimBiology.export.StochasticSimulationOptions` |  
`SimBiology.export.ImplicitTauSimulationOptions`  
| `SimBiology.export.SimulationOptions` |  
`SimBiology.export.ODESimulationOptions` | `export`

# SimBiology.export.ImplicitTauSimulationOptions

---

**Superclasses** SimBiology.export.StochasticSimulationOptions

**Purpose** Settings for implicit tau-leaping stochastic simulation of exported SimBiology model

**Description** SimBiology.export.ImplicitTauSimulationOptions is the class of simulation options associated with the implicit tau-leaping solver of an export model.

**Construction** Implicit tau-leaping simulation options are created by the export method for SimBiology models with a stochastic SolverType set to 'impltau'.

export (model)

Export SimBiology model

## Properties

### ErrorTolerance

Error tolerance, a scalar value in the range (0,1).

### LogDecimation

Frequency to log stochastic simulation output, a positive integer value.

### MaxIterations

Nonlinear solver maximum number of iterations, a positive integer value.

### MaximumNumberofLogs

Maximum number of logs criterion to stop simulation, a positive scalar value.

### MaximumWallClock

Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value.

### RandomState

# SimBiology.export.ImplicitTauSimulationOptions

---

Random number generator, a positive integer value.

## **SolverType**

String indicating solver type to use for simulation, 'impltau'.  
This property is read only.

## **StopTime**

Simulation time criterion to stop simulation, a nonnegative scalar value.

## **TimeUnits**

Time units for simulation. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## **See Also**

`SimBiology.export.StochasticSimulationOptions` |  
`SimBiology.export.ExplicitTauSimulationOptions`  
| `SimBiology.export.SimulationOptions` |  
`SimBiology.export.ODESimulationOptions` | `export`



HTML text describing the exported model object. This property is read only.

## SimulationOptions

SimBiology.export.SimulationOptions object specifying simulation options.

## ValueInfo

Array of SimBiology.export.ValueInfo objects of modifiable species, parameters, and compartments.

## Methods

accelerate	Prepare exported SimBiology model for acceleration
getdose	Return exported SimBiology model dose object
getIndex	Get indices into ValueInfo and InitialValues properties
isAccelerated	Determine whether an exported SimBiology model is accelerated
simulate	Simulate exported SimBiology model

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## Examples

### Exported SimBiology Model Object

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj)
```

```
em =
```

# SimBiology.export.Model

---

```
SimBiology.export.Model
```

```
    Name: lotka  
    ExportTime: 28-Jun-2012 08:13:47  
    ExportNotes:
```

Display the editable values (compartments, species, and parameters) information.

```
em.ValueInfo
```

```
ans =
```

```
8x1 SimBiology.export.ValueInfo handle  
Package: SimBiology.export
```

```
Properties:
```

```
    Constant  
    InitialValue  
    Name  
    Parent  
    QualifiedName  
    Tag  
    Type  
    Units
```

```
Methods, Events, Superclasses
```

There are 8 editable values. Display the names of the editable values.

```
{em.ValueInfo.Name}
```

```
ans =
```

```
'unnamed'    'x'    'y1'    'y2'    'z'    'c1'    'c2'    'c3'
```

Display the exported model simulation options.

`em.SimulationOptions`

`ans =`

```
SimBiology.export.ODESimulationOptions handle  
Package: SimBiology.export
```

Properties:

```
AbsoluteTolerance: 1.0000e-06  
AbsoluteToleranceScaling: 1  
AbsoluteToleranceStepSize: []  
MaxStep: []  
OutputTimes: []  
RelativeTolerance: 1.0000e-03  
SolverType: 'ode15s'  
MaximumNumberOfLogs: Inf  
MaximumWallClock: Inf  
StopTime: 10  
TimeUnits: 'second'
```

Methods, Events, Superclasses

The exported model has a deterministic solver.

## See Also

[SimBiology.export.Dose](#) | [SimBiology.export.SimulationOptions](#)  
| [SimBiology.export.ValueInfo](#) |

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

## Concepts

- Class Attributes
- Property Attributes

# SimBiology.export.ODESimulationOptions

---

<b>Superclasses</b>	SimBiology.export.SimulationOptions
<b>Purpose</b>	Settings for deterministic, ordinary differential equation simulation of exported SimBiology model
<b>Description</b>	SimBiology.export.ODESimulationOptions is the class of simulation options associated with deterministic, ordinary differential equation (ODE) solvers.
<b>Construction</b>	Deterministic simulation options are created by the <code>export</code> method for SimBiology models with a deterministic SolverType (for example, <code>sundials</code> or <code>ode15s</code> ).  <code>export (model)</code> Export SimBiology model
<b>Properties</b>	<b>AbsoluteTolerance</b> Absolute error tolerance applied to state value during simulation, a positive scalar value. <b>AbsoluteToleranceScaling</b> Control scaling of absolute error tolerance, a logical value. <b>AbsoluteToleranceStepSize</b> Initial guess for time step size for scaling of absolute error tolerance, [] or a scalar value. <b>MaximumNumberofLogs</b> Maximum number of logs criterion to stop simulation, a positive scalar value. <b>MaximumWallClock</b> Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value. <b>MaxStep</b>



Upper bound on ODE solver step size, [ ] or a positive scalar value.

## **OutputTimes**

Times to log in simulation output, a vector of sorted nonnegative values.

## **RelativeTolerance**

Allowable error tolerance relative to state value during simulation, a scalar value in the range (0,1).

## **SolverType**

String indicating solver type to use for simulation. Possible deterministic solver types are:

- 'sundials'
- 'ode15s'
- 'ode23t'
- 'ode45'

## **StopTime**

Simulation time criterion to stop simulation, a nonnegative scalar value.

## **TimeUnits**

Time units for simulation. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## **See Also**

`SimBiology.export.SimulationOptions` |  
`SimBiology.export.StochasticSimulationOptions` | `export`

# SimBiology.export.RepeatDose

---

**Superclasses** SimBiology.export.Dose

**Purpose** Repeated doses for exported SimBiology model

**Description** SimBiology.export.RepeatDose is the class for export repeat doses.

**Construction** Export repeat dose objects are created by the `export` method for SimBiology models. By default, all active repeat doses are export repeat doses, but you can specify which repeat doses to export using the optional `editdoses` input argument to `export`.

`export (model)` Export SimBiology model

## Properties

### Amount

Amount of dose, a nonnegative scalar value.

### AmountUnits

Dose amount units. This property is read only.

### DurationParameterName

Parameter specifying length of time. This property is read only.

### Interval

Time between doses, a nonnegative scalar value.

### LagParameterName

Parameter specifying time lag for the dose. This property is read only.

### Name

Name of dose object. This property is read only.

### Notes

Text describing dose object. This property is read only.

### Parent

Name of the parent export model. This property is read only.

**Rate**

Rate of dose, a nonnegative scalar value.

**RateUnits**

Units for dose rate. This property is read only.

**RepeatCount**

Dose repetitions, a nonnegative integer value.

**StartTime**

Start time for initial dose, a nonnegative scalar value.

**TargetName**

Species receiving dose. This property is read only.

**TimeUnits**

Time units for dosing. This property is read only.

**Copy  
Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

**See Also**

`SimBiology.export.Dose` | `SimBiology.export.ScheduleDose` | `export`

# SimBiology.export.ScheduleDose

---

**Superclasses** SimBiology.export.Dose

**Purpose** Schedule dose for exported SimBiology model

**Description** SimBiology.export.ScheduleDose is the class for export schedule doses.

**Construction** Export schedule dose objects are created by the `export` method for SimBiology models. By default, all active schedule doses are export schedule doses, but you can specify which schedule doses to export using the optional `editdoses` input argument to `export`.

`export (model)`

Export SimBiology model

## Properties

### Amount

Amount of dose, a nonnegative scalar value.

### AmountUnits

Dose amount units. This property is read only.

### DurationParameterName

Parameter specifying length of time. This property is read only.

### LagParameterName

Parameter specifying time lag for the dose. This property is read only.

### Name

Name of dose object. This property is read only.

### Notes

Text describing dose object. This property is read only.

### Parent

Name of the parent export model. This property is read only.

**Rate**

Rate of dose, a nonnegative scalar value.

**RateUnits**

Units for dose rate. This property is read only.

**TargetName**

Species receiving dose. This property is read only.

**Time**

Schedule dose times, a vector of nonnegative values.

**TimeUnits**

Time units for dosing. This property is read only.

**Copy  
Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

**See Also**

`SimBiology.export.Dose` | `SimBiology.export.RepeatDose` | `export`

# SimBiology.export.SimulationOptions

---

**Purpose** Simulation settings for exported SimBiology model

**Description** `SimBiology.export.SimulationOptions` is the superclass of simulation options for exported models. Simulation options are either of subclass `SimBiology.export.ODESimulationOptions` for deterministic solvers, or `SimBiology.export.StochasticSimulationOptions` for stochastic solvers.

**Construction** Simulation options are created by the `export` method for SimBiology models.

`export (model)`

Export SimBiology model

## Properties

### MaximumNumberofLogs

Maximum number of logs criterion to stop simulation, a positive scalar value.

### MaximumWallClock

Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value.

### StopTime

Simulation time criterion to stop simulation, a nonnegative scalar value.

### TimeUnits

Time units for simulation. This property is read only.

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## See Also

`SimBiology.export.ODESimulationOptions` |  
`SimBiology.export.StochasticSimulationOptions` | `export`

# SimBiology.export.StochasticSimulationOptions

---

<b>Superclasses</b>	SimBiology.export.SimulationOptions
<b>Purpose</b>	Settings for stochastic simulation of exported SimBiology model
<b>Description</b>	SimBiology.export.StochasticSimulationOptions is the superclass of simulation options associated with stochastic solvers. The subclasses of SimBiology.export.StochasticSimulationOptions are SimBiology.export.ExplicitTauSimulationOptions and SimBiology.export.ImplicitTauSimulationOptions.
<b>Construction</b>	<p>Stochastic simulation options are created by the export method for SimBiology models with a stochastic SolverType (ssa, expltau, or impltau).</p> <pre>export (model)                                Export SimBiology model</pre>
<b>Properties</b>	<p><b>LogDecimation</b></p> <p>Frequency to log stochastic simulation output, a positive integer value.</p> <p><b>MaximumNumberofLogs</b></p> <p>Maximum number of logs criterion to stop simulation, a positive scalar value.</p> <p><b>MaximumWallClock</b></p> <p>Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value.</p> <p><b>RandomState</b></p> <p>Random number generator, a positive integer value.</p> <p><b>SolverType</b></p> <p>String indicating solver type to use for simulation. This property is read only. The stochastic solver type is one of:</p>

# SimBiology.export.StochasticSimulationOptions

---

- 'ssa'
- 'exptau'
- 'impltau'

## StopTime

Simulation time criterion to stop simulation, a nonnegative scalar value.

## TimeUnits

Time units for simulation. This property is read only.

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## See Also

[SimBiology.export.SimulationOptions](#) |  
[SimBiology.export.ODESimulationOptions](#) |  
[SimBiology.export.ExplicitTauSimulationOptions](#) |  
[SimBiology.export.ImplicitTauSimulationOptions](#) | [export](#)



<b>Purpose</b>	Modifiable species, compartments, or parameters in exported SimBiology model
<b>Description</b>	SimBiology.export.ValueInfo is the class that describes the modifiable value components in a SimBiology.export.Model, including species, parameters, and compartments.
<b>Construction</b>	<p>ValueInfo objects are created by the export method for SimBiology models. By default, all model species, parameters, and compartments are ValueInfo objects, but you can specify which value components to export using the optional editvals input argument to export.</p> <pre>export (model)                Export SimBiology model</pre>
<b>Properties</b>	<p><b>Constant</b></p> <p>Display whether value is constant or time-varying. This property is read only.</p> <p><b>InitialValue</b></p> <p>Initial value for the component, a scalar value.</p> <p><b>Name</b></p> <p>Name of species, compartment, or parameter. This property is read only.</p> <p><b>Parent</b></p> <p>Name of parent model, compartment, or reaction. This property is read only.</p> <p><b>QualifiedName</b></p> <p>Qualified name of species, compartment, or parameter. This property is read only.</p> <ul style="list-style-type: none"><li>• For compartments and model-scoped parameters, the qualified name is the same as the name.</li></ul>

# SimBiology.export.ValueInfo

---

- For species, the qualified name is `CompartmentName.SpeciesName`.
- For reaction-scoped parameters, the qualified name is `ReactionName.ParameterName`.

## Tag

Label for species, compartment, or parameter. This property is read only.

## Type

Type of value (species, parameter, or compartment). This property is read only.

## Units

Value units. This property is read only

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

## See Also

`SimBiology.export.Model` | `export`

# Method Reference

---

Objects (p. 3-3)	SimBiology objects
Abstract Kinetic Laws (p. 3-5)	Work with abstract kinetic law objects
Compartments (p. 3-6)	Work with compartment objects
Configuration Sets (p. 3-7)	Work with configuration set objects
Covariate Analysis (p. 3-8)	Work with CovariateModel objects
Events (p. 3-9)	Work with event objects
Kinetic Laws (p. 3-10)	Create parameter objects and work with kinetic law objects
Models (p. 3-11)	Create SimBiology objects and work with model objects
Parameters (p. 3-13)	Work with parameter objects
Pharmacokinetic Modeling (p. 3-14)	Create and work with objects for PK modeling and analysis
PKCovariate Analysis (p. 3-15)	Work with PKCovariateModel objects
Reactions (p. 3-16)	Create kinetic law and species objects and work with reaction objects
RepeatDose (p. 3-17)	Methods for RepeatDose objects
Root (p. 3-18)	Work with the root object
Rules (p. 3-19)	Work with rule objects
ScheduleDose (p. 3-20)	Methods for ScheduleDose objects

SimData (p. 3-21)

Species (p. 3-22)

Units and Unit Prefixes (p. 3-23)

Variants (p. 3-24)

Using Object Methods (p. 3-25)

Methods for SimData objects

Methods for species objects

Methods for unit and prefix objects

Methods for variant objects

Command-line syntax for using  
methods with SimBiology objects

## Objects

AbstractKineticLaw object	Kinetic law information in library
Compartment object	Options for compartments
Configset object	Solver settings information for model simulation
CovariateModel object	Define relationship between parameters and covariates
Event object	Store event information
KineticLaw object	Kinetic law information for reaction
Model object	Model and component information
Parameter object	Parameter and scope information
PKCompartment object	Used by PKModelDesign to create SimBiology model
PKData object	Define roles of data set columns
PKModelDesign object	Helper object to construct pharmacokinetic model
PKModelMap object	Define SimBiology model components' roles
Reaction object	Options for model reactions
RepeatDose object	Define drug dosing protocol
Root object	Hold models, unit libraries, and abstract kinetic law libraries
Rule object	Hold rule for species and parameters
ScheduleDose object	Define drug dosing protocol
SimData object	Simulation data storage
Species object	Options for compartment species
Unit object	Hold information about user-defined unit

UnitPrefix object

Hold information about user-defined  
unit prefix

Variant object

Store alternate component values

## Abstract Kinetic Laws

delete (any object)

display (any object)

get (any object)

set (any object)

Delete SimBiology object

Display summary of SimBiology  
object

Get object properties

Set object properties

## Compartments

<code>addcompartment (model, compartment)</code>	Create compartment object
<code>addspecies (model, compartment)</code>	Create species object and add to compartment object within 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>get (any object)</code>	Get object properties
<code>rename (compartment, parameter, species)</code>	Rename object and update expressions
<code>reorder (model, compartment)</code>	Reorder component lists
<code>set (any object)</code>	Set object properties



## Configuration Sets

copyobj (any object)

Copy SimBiology object and its children

delete (any object)

Delete SimBiology object

display (any object)

Display summary of SimBiology object

set (any object)

Set object properties

## Covariate Analysis

<code>constructDefaultFixedEffectValues</code> (covmodel)	Create initial estimate vector needed for fit
CovariateModel object	Define relationship between parameters and covariates
<code>verify</code> (covmodel)	Check covariate model for errors

## Events

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

## Kinetic Laws

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

## Models

<code>addcompartment (model, compartment)</code>	Create compartment object
<code>addconfigset (model)</code>	Create configuration set object and add to model object
<code>adddose (model)</code>	Add dose object to model
<code>addevent (model)</code>	Add event 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, compartment)</code>	Create species object and add to compartment object within model object
<code>addvariant (model)</code>	Add variant to model
<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>export (model)</code>	Export SimBiology model
<code>get (any object)</code>	Get object properties
<code>getadjacencymatrix (model)</code>	Get adjacency matrix from model object
<code>getconfigset (model)</code>	Get configuration set object from model object
<code>getdose (model)</code>	Return SimBiology dose object
<code>getequations</code>	Return system equations for model object

<code>getstoichmatrix (model)</code>	Get stoichiometry matrix from model object
<code>getvariant (model)</code>	Get variant from model
<code>removeconfigset (model)</code>	Remove configuration set from model
<code>removedose (model)</code>	Add dose object to model
<code>removevariant (model)</code>	Remove variant from model
<code>reorder (model, compartment)</code>	Reorder component lists
<code>set (any object)</code>	Set object properties
<code>setactiveconfigset (model)</code>	Set active configuration set for model object
<code>verify (model, variant)</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
get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

## Pharmacokinetic Modeling

<code>addCompartment (PKModelDesign)</code>	Add compartment to <code>PKModelDesign</code> object
<code>construct (PKModelDesign)</code>	Construct <code>SimBiology</code> model from <code>PKModelDesign</code> object
<code>get (any object)</code>	Get object properties
<code>getCovariateData (pkdata)</code>	Create design matrix needed for fit
<code>PKCompartment</code> object	Used by <code>PKModelDesign</code> to create <code>SimBiology</code> model
<code>PKData</code> object	Define roles of data set columns
<code>PKModelDesign</code> object	Helper object to construct pharmacokinetic model
<code>PKModelMap</code> object	Define <code>SimBiology</code> model components' roles
<code>set (any object)</code>	Set object properties



## **PKCovariate Analysis**

## Reactions

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

## RepeatDose

copyobj (any object)

Copy SimBiology object and its children

get (any object)

Get object properties

set (any object)

Set object properties

## **Root**

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

## Rules

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

## **ScheduleDose**

copyobj (any object)

Copy SimBiology object and its children

get (any object)

Get object properties

set (any object)

Set object properties

## SimData

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

## **Species**

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



## Units and Unit Prefixes

delete (any object)

display (any object)

get (any object)

set (any object)

Delete SimBiology object

Display summary of SimBiology  
object

Get object properties

Set object properties

## Variants

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

## Using Object Methods

Command-line syntax for using methods with SimBiology objects

Constructing (Creating) Objects  
(p. 3-25)

Using Object Methods (p. 3-25)

Help for Objects, Methods, and  
Properties (p. 3-26)

### Constructing (Creating) Objects

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

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

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

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

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

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

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

### Using Object Methods

Using MATLAB function notation:

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

Using object dot notation:

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

## **Help for Objects, Methods, and Properties**

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

<code>help sbio</code>	Display a list of functions and methods.
<code>help FunctionName</code>	Display function information.

# Methods — Alphabetical List

---

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

# SimBiology.export.Model.accelerate

---

**Purpose** Prepare exported SimBiology model for acceleration

**Syntax** `accelerate(model)`

**Description** `accelerate(model)` prepares the exported model for acceleration on the current type of computer.

---

**Note** Microsoft Visual Studio 2010 run-time libraries must be available on any computer running accelerated models generated using Microsoft Windows SDK. If you plan to redistribute your accelerated models to other MATLAB users, be sure they have the run-time libraries.

---

## Input Arguments

### **model**

SimBiology.export.Model object.

## Examples

### **Accelerate Exported SimBiology Model**

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj)
```

```
em =
```

```
SimBiology.export.Model
```

```
    Name: lotka
```

```
    ExportTime: 28-Jun-2012 08:13:47
```

```
    ExportNotes:
```

Accelerate the exported model.

```
accelerate(em);  
em.isAccelerated
```

```
ans =
```

```
1
```

The logical value 1 indicates that the exported model is accelerated.

## See Also

[SimBiology.export.Model](#) |  
[SimBiology.export.Model.isAccelerated](#) |

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# AbstractKineticLaw object

---

**Purpose** Kinetic law information in library

**Description** The abstract kinetic law object represents a *kinetic law definition*, 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 kinetic law definition specifies 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 `Expression` property. 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 kinetic law definition relates to the kinetic law object, see `KineticLaw` object.

Create your own kinetic law definition and add it to the kinetic law library with the `sbioaddtolibrary` function. You can then use the kinetic law to define a reaction rate. To retrieve a kinetic law definition from the user-defined library, first create a root object using `sbiroot`, then use the command `get(rootObj.UserDefinedLibrary, 'KineticLaws')`.

See “Property Summary” on page 4-5 for links to abstract kinetic law object property reference pages.

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

**Constructor Summary** `sbioabstractkineticlaw` Create kinetic law definition



## Method Summary

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

## Property Summary

Expression (AbstractKineticLaw, KineticLaw)	Expression to determine reaction rate equation
Name	Specify name of object
Notes	HTML text describing SimBiology object
ParameterVariables	Parameters in kinetic law definition
Parent	Indicate parent object
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## See Also

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

# addcompartment (model, compartment)

---

**Purpose** Create compartment object

**Syntax**

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

## Arguments

<i>modelObj</i>	Model object.
<i>owningCompObj</i>	Compartment object that contains the newly created compartment object.
<i>NameValue</i>	Name for a compartment object. Enter a character string unique to the model object. For information on naming compartments, see Name.
<i>CapacityValue</i>	Capacity value for the compartment object. Enter double. Positive real number, default = 1.
<i>PropertyName</i>	Enter the name of a valid property. Valid property names are listed in “Property Summary” on page 4-8.
<i>PropertyValue</i>	Enter the value for the property specified in <i>PropertyName</i> . Valid property values are listed on each property reference page.

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

## addcompartment (model, compartment)

---

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

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

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

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

`compartmentObj = addcompartment(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). “Property Summary” on page 4-8 lists the properties. The Owner property is one exception; you cannot set the Owner property in the `addcompartment` syntax because, `addcompartment` requires the owning model or compartment to be specified as the first argument and uses this information to set the Owner property. After adding a compartment, you can change the owner using the function `set`.

# addcompartment (model, compartment)

---

## Method Summary

Methods for compartment objects

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

## Property Summary

Properties for compartment objects

Capacity	Compartment capacity
CapacityUnits	Compartment capacity units
Compartments	Array of compartments in model or compartment
ConstantCapacity	Specify variable or constant compartment capacity
Name	Specify name of object
Notes	HTML text describing SimBiology object
Owner	Owning compartment

# addcompartment (model, compartment)

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

## Examples

- 1 Create a model object (modelObj).

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

- 2 Add two compartments to the model object.

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

- 3 Add a compartment to one of the compartment objects.

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

- 4 Display the Compartments property in the model object.

```
get(modelObj, 'Compartments')
```

```
SimBiology Compartment Array
```

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

- 5 Display the Compartments property in the compartment object.

```
get(compartmentObj2, 'Compartments')
```

## addcompartment (model, compartment)

---

SimBiology Compartment - matrix

Compartment Components:

Capacity:	1
CapacityUnits:	
Compartments:	0
ConstantCapacity:	true
Owner:	mitochondrion
Species:	0

### **See Also**

addproduct, addreactant, addreaction, addspecies, get, set

# addCompartment (PKModelDesign)

---

**Purpose** Add compartment to PKModelDesign object

**Syntax**

```
PKCompartmentObj = addCompartment(PKModelDesignObj,  
    CompObjName)  
PKCompartmentObj = addCompartment(PKModelDesignObj,  
    CompObjName, Name, Value)
```

**Description**

*PKCompartmentObj = addCompartment(PKModelDesignObj, CompObjName)* constructs a PK compartment with the specified name and adds it to *PKModelDesignObj*, a PKModelDesign object.

*PKCompartmentObj = addCompartment(PKModelDesignObj, CompObjName, Name, Value)* constructs a PK compartment with the specified name, and with additional options specified by one or more Name, Value pair arguments.

**Input Arguments**

<i>PKModelDesignObj</i>	PKModelDesign object to which you want to add a compartment
<i>CompObjName</i>	Name of the PKCompartment object that is constructed

## Name-Value Pair Arguments

Optional comma-separated pairs of *Name*, *Value* arguments, where *Name* is the argument name and *Value* is the corresponding value. *Name* must appear inside single quotes (' '). You can specify several name-value pair arguments in any order as *Name1,Value1, ,NameN,ValueN*.

## addCompartment (PKModelDesign)

---

DosingType	<p>String specifying the mechanism for drug absorption. Choices are:</p> <ul style="list-style-type: none"><li>• 'Bolus'</li><li>• 'Infusion'</li><li>• 'ZeroOrder'</li><li>• 'FirstOrder'</li><li>• '' (default)</li></ul>
EliminationType	<p>For more information, see “Dosing Types”.</p> <p>String specifying the mechanism for drug elimination. Choices are:</p> <ul style="list-style-type: none"><li>• 'Linear'</li><li>• 'Linear-Clearance'</li><li>• 'Enzymatic'</li><li>• '' (default)</li></ul> <p>For more information, see “Elimination Types”.</p>
HasResponseVariable	<p>Logical indicating if the drug concentration in this compartment is reported. Multiple compartments in a model can have this property set to true. Default is false.</p>



# addCompartment (PKModelDesign)

---

---

**Note** If you perform a parameter fit on a model, at least one compartment in the model must have a `HasResponseVariable` property set to `true`.

---

`HasLag` Logical indicating if any dose targeting this compartment have a lag associated with them. Default is `false`.

These optional name-value pair arguments set the corresponding property of the `PKCompartment` object. You can also set these properties after creating the `PKCompartment` object by using the following syntax:

```
PKCompartmentObj.PropertyName = Value
```

For example:

```
PKCompartmentObj.DosingType = 'Bolus'
```

## Output Arguments

*PKCompartmentObj* PKCompartment object

## Method Summary

<code>get (any object)</code>	Get object properties
<code>set (any object)</code>	Set object properties

## Property Summary

<code>DosingType</code>	Drug dosing type in compartment
<code>EliminationType</code>	Drug elimination type from compartment
<code>HasLag</code>	Lag associated with dose targeting compartment

## addCompartment (PKModelDesign)

---

HasResponseVariable	Compartment drug concentration reported
Name	Specify name of object

### See Also

“Creating Pharmacokinetic Model Using the Command Line”, HasLag, HasResponseVariable, PKCompartment object, PKModelDesign object

**Purpose** Create configuration set object and add to model object

**Syntax**

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

## Arguments

*modelObj* Model object. Enter a variable name.

*NameValue* Descriptive name for a configuration set object. Reserved words 'active' and 'default' are not allowed.

*configsetObj* Configuration set object.

## Description

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

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

*configsetObj* = addconfigset(..., 'PropertyName', *PropertyValue*, ...) constructs a configuration set object, *configsetObj*, and configures *configsetObj* with property value pairs. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs). The *configsetObj* properties are listed in “Property Summary” on page 4-16.

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

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

# addconfigset (model)

---

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

## Method Summary

Methods for configuration set objects

<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>set</code> (any object)	Set object properties

## Property Summary

Properties for configuration set objects

<code>Active</code>	Indicate object in use during simulation
<code>CompileOptions</code>	Dimensional analysis and unit conversion options
<code>MaximumNumberOfLogs</code>	Maximum number of logs criteria to stop simulation
<code>MaximumWallClock</code>	Maximum elapsed wall clock time to stop simulation
<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

SolverType	Select solver type for simulation
StartTime	Start time for initial dose time
StopTime	Simulation time criteria to stop simulation
TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type

## Examples

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

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

- 2 Configure the simulation stop criteria by setting the `StopTime`, `MaximumNumberOfLogs`, and `MaximumWallClock` properties of the `Configset` object. Set the stop criteria to a simulation time of 3000 seconds, 50 logs, or a wall clock time of 10 seconds, whichever comes first.

```
set(configsetObj, 'StopTime', 3000, 'MaximumNumberOfLogs', 50,...  
    'MaximumWallClock', 10)  
get(configsetObj)
```

```
Active: 0  
CompileOptions: [1x1 SimBiology.CompileOptions]  
Name: 'myset'  
Notes: ''  
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]  
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]  
SolverOptions: [1x1 SimBiology.ODESolverOptions]  
SolverType: 'ode15s'  
StopTime: 3000  
MaximumNumberOfLogs: 50  
MaximumWallClock: 10
```

## addconfigset (model)

---

```
TimeUnits: 'second'  
Type: 'configset'
```

- 3 Set the new Configset object to be active, simulate the model using the new Configset object, and plot the result.

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

### See Also

get, getconfigset, removeconfigset, set, setactiveconfigset

**Purpose** Append content to variant object

**Syntax**  
addcontent(*variantObj*, *contents*)  
addcontent(*variantObj1*, *variantObj2*)

**Arguments**

*variantObj* Specify the variant object to which you want to append data. The Content property is modified to add the new data.

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

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

# addcontent (variant)

---

## Description

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

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

---

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

---

## Examples

- 1 Create a model containing one species.

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

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

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

## See Also

`addvariant`, `rmcontent`, `sbiovariant`



**Purpose** Add dose object to model

**Syntax**

```
doseObj2 = adddose(mode1Obj, 'DoseName')  
doseObj2 = adddose(mode1Obj, 'DoseName', 'DoseType')  
doseObj2 = adddose(mode1Obj, doseObj)
```

## Arguments

<i>mode1Obj</i>	Model object to which you add a dose object.
<i>DoseName</i>	Name of a dose object to construct and add to a model object. <i>DoseName</i> is the value of the dose object property Name.
<i>DoseType</i>	Type of dose object to construct and add to a model object. Enter either 'schedule' or 'repeat'.
<i>doseObj</i>	Dose object to add to a model object. Created with the constructor sbiodose.

## Outputs

<i>doseObj2</i>	ScheduleDose or RepeatDose object. A RepeatDose or ScheduleDose object defines an increase (dose) to a species amount during a simulation.
-----------------	--------------------------------------------------------------------------------------------------------------------------------------------

## Description

Before using a dose object in a simulation, use the `adddose` method to add the dose object to a SimBiology model object. Then, set the `Active` dose object property to `true`.

`doseObj2 = adddose(mode1Obj, 'DoseName')` constructs a SimBiology RepeatDose object (*doseObj2*), assigns *DoseName* to the property `Name`, adds the dose object to a SimBiology model object (*mode1Obj*), and assigns *mode1Obj* to the property `Parent`.

# adddose (model)

---

`doseObj2 = adddose(modeObj, 'DoseName', 'DoseType')` constructs either a SimBiology ScheduleDose object or RepeatDose object (`doseObj`).

`doseObj2 = adddose(modeObj, doseObj)` adds a SimBiology dose object (`doseObj`) to a SimBiology model object (`modeObj`), copies the dose object to a second dose object (`doseObj2`), and assigns `modeObj` to the property Parent.

## Example

Add a dose object to a model object.

- 1 Create a model then add a dose to the model.

```
modelObj = sbiomodel('mymodel');  
doseObj = adddose(modelObj, 'dose1');
```

- 2 Define properties of the dose object.

```
doseObj.Amount = 5;  
doseObj.Repeat = 6;  
doseObj.Interval = 24;  
doseObj.TimeUnits = 'hour'
```

## See Also

Model object methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

ScheduleDose object and RepeatDose object methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

**Purpose** Add event object to model object

**Syntax**

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

**Arguments**

<i>modelObj</i>	Model object.
<i>TriggerValue</i>	Required property to specify a trigger condition. Must be a MATLAB expression that evaluates to a logical value. Use the keyword 'time' to specify that an event occurs at a specific time during the simulation. For more information, see Trigger.
<i>EventFcnsValue</i>	A string or a cell array of strings, each of which specifies an assignment of the form ' <i>objectname</i> = <i>expression</i> ', where <i>objectname</i> is the name of a valid object. Defines what occurs when the event is triggered. For more information, see EventFcns.
<i>PropertyName</i>	Property name for an event object from “Property Summary” on page 4-24.
<i>PropertyValue</i>	Property value. For more information on property values, see the property reference for each property listed in “Property Summary” on page 4-24.

**Description**

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

# addevent (model)

---

object (*modelObj*) to the property *Parent*. In the model object, this method appends the event object to the property *Events*.

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

For details on how events are handled during a simulation, see “Event Object”.

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

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

## Method Summary

<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>get</code> (any object)	Get object properties
<code>set</code> (any object)	Set object properties

## Property Summary

<code>Active</code>	Indicate object in use during simulation
<code>EventFcns</code>	Event expression
<code>Name</code>	Specify name of object
<code>Notes</code>	HTML text describing SimBiology object

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

## Examples

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

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

- 2 Get a list of properties for an event object.

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

Or

```
get(eventObj)
```

MATLAB displays a list of event properties.

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

## See Also

Event object

# addkineticlaw (reaction)

---

**Purpose** Create kinetic law object and add to reaction object

**Syntax**

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

**Arguments**

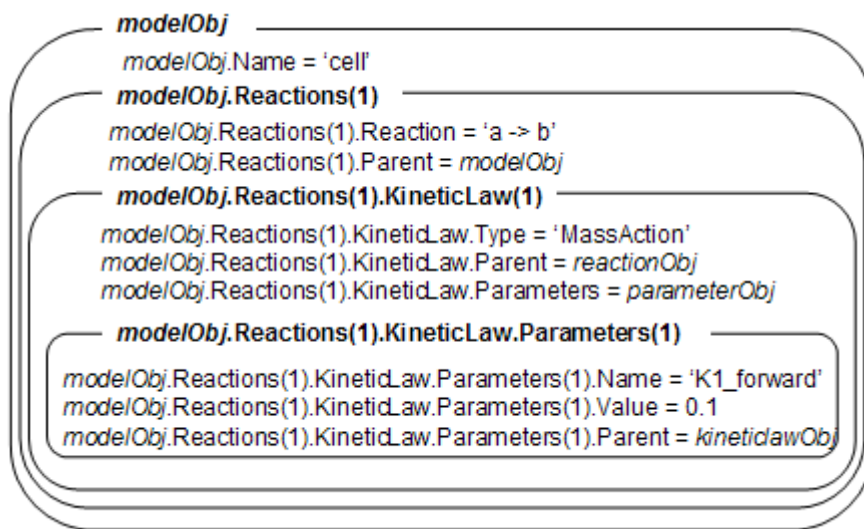
<i>reactionObj</i>	Reaction object. Enter a variable name for a reaction object.
<i>KineticLawNameValue</i>	Property to select the type of kinetic law object to create. For built-in kinetic law, valid values are:  'Unknown', 'MassAction', 'Henri-Michaelis-Menten', 'Henri-Michaelis-Menten-Reversible', 'Hill-Kinetics', 'Iso-Uni-Uni', 'Ordered-Bi-Bi', 'Ping-Pong-Bi-Bi', 'Competitive-Inhibition', 'NonCompetitive-Inhibition', and 'UnCompetitive-Inhibition'.  Find valid <i>KineticLawNameValues</i> by using <i>sbioroot</i> to create a SimBiology root object, then query the object with the commands <code>get(rootObj.BuiltInLibrary, 'KineticLaws')</code> and <code>get(rootObj.UserDefinedLibrary, 'KineticLaws')</code> .  <code>sbiowhos -kineticlaw</code> lists kinetic laws in the SimBiology root, which includes kinetic laws from both the <code>BuiltInLibrary</code> and the <code>UserDefinedLibrary</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 kinetic law definition. See “Kinetic Law Definition” on page 6-65 for a definition of kinetic laws and more information about how they are used to get the reaction rate expression.

*kineticlawObj* = addkineticlaw(..., '*PropertyName*', *PropertyValue*, ...) constructs a kinetic law object, *kineticlawObj*, and configures

## addkineticlaw (reaction)

---

*kineticlawObj* with property value pairs. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The *kineticlawObj* properties are listed in “Property Summary” on page 4-29.

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(reactiObj, '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</code> (model, kineticlaw)	Create parameter object and add to model or kinetic law 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>get</code> (any object)	Get object properties
<code>getparameters</code> (kineticlaw)	Get specific parameters in kinetic law object
<code>getspecies</code> (kineticlaw)	Get specific species in kinetic law object
<code>set</code> (any object)	Set object properties
<code>setparameter</code> (kineticlaw)	Specify specific parameters in kinetic law object
<code>setspecies</code> (kineticlaw)	Specify species in kinetic law object



## Property Summary

Properties for kinetic law objects

Expression (AbstractKineticLaw, KineticLaw)	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 kinetic law definition
Parent	Indicate parent object
SpeciesVariableNames	Cell array of species in reaction rate equation
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Examples

### Example 1

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

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

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

## addkineticlaw (reaction)

---

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

- 2 Define a kinetic law for the reaction object and view the parameters to be set.

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

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

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

The Henri-Michaelis-Menten kinetic law has two parameters (*V<sub>m</sub>* and *K<sub>m</sub>*) and one species (*S*). You need to enter values for these parameters by first creating parameter objects, and then adding the parameter objects to the kinetic law object.

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

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

The `addparameter` method creates two parameter objects with values that are associated with the kinetic law parameters.

- 4 Associate kinetic law parameters with the parameters in the kinetic law definition.

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

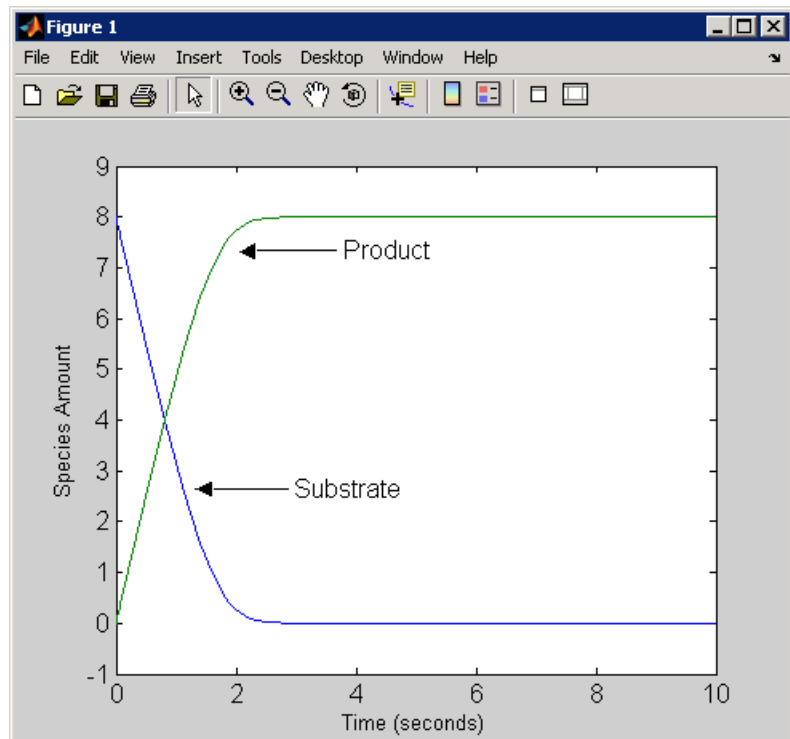
This method associates the parameters in the property `ParameterVariableNames` with the parameters in the property `ParameterVariables` using a one-to-one mapping in the order given.

- 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

This example uses the built-in kinetic law `MassAction`.

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

```
modelObj = sbiomodel ('Cell');  
reactionObj = addreaction (modelObj, 'a -> b');
```

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');  
get(kineticlawObj, 'Expression')  
ans =  
    MassAction
```

Notice, the property `Expression` for `MassAction` kinetic law does not show the parameters and species in the reaction rate.

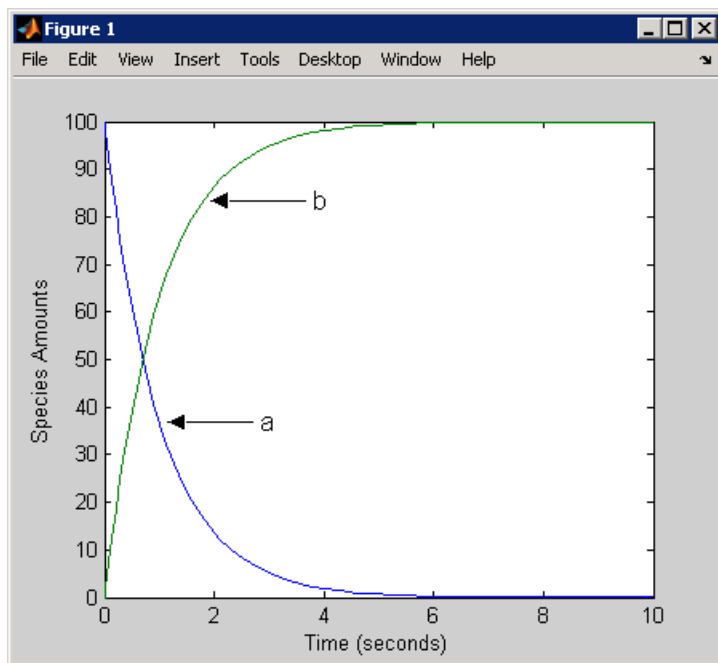
- 3 Assign the rate constant for the reaction.

```
parameterObj = addparameter(kineticlawObj, 'k_forward');  
set (kineticlawObj, 'ParameterVariablenames', 'k_forward');  
get (reactionObj, 'ReactionRate')  
  
ans =  
    k_forward*a
```

- 4 Enter an initial value for the substrate and simulate.

```
modelObj.Species(1).InitialAmount = 100;  
[T, X] = sbiosimulate(modelObj);plot(T,X)
```

The value used for `k_forward` is the default value = 1.0.



**See Also** `addreaction`, `setparameter`

# 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. Since objects can use this property to reference a parameter, a parameter object must have a unique name at the level it is created. For example, a kinetic law object cannot contain two parameter objects named kappa. However, the model object that contains the kinetic law object can contain a parameter object named kappa along with the kinetic law object.  For information on naming parameters, see Name.
<i>ValueValue</i>	Property for a parameter object. Enter a number.

## Description

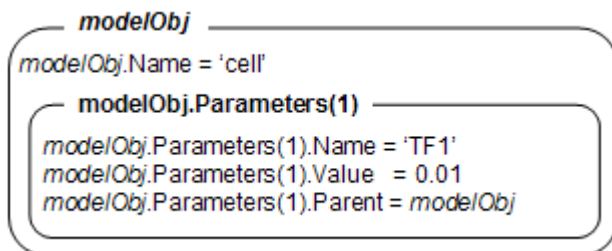
`parameterObj = addparameter(Obj, 'NameValue')` creates a parameter object and returns the object (*parameterObj*). In the parameter object, this method assigns a value (*NameValue*) to the property Name, assigns a value 1 to the property Value, and assigns the model or kinetic law object to the property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

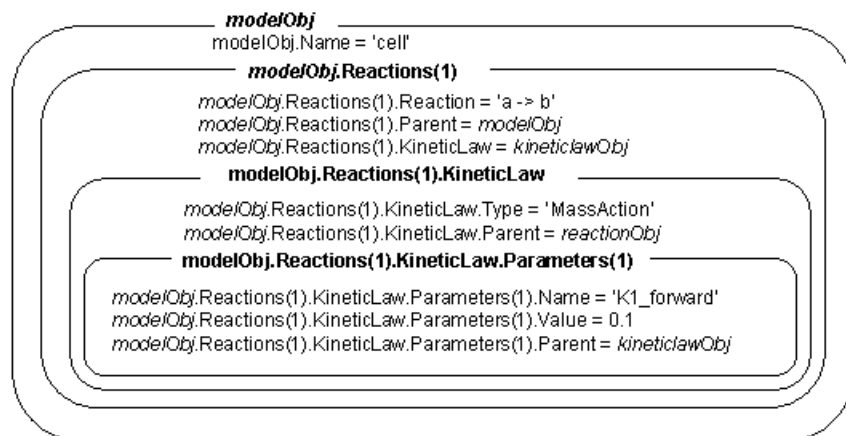
## addparameter (model, kineticlaw)

parameter. If a parameter object is defined with a model object as its parent, then all objects within the model (including all rules, events and kinetic laws) can use the parameter.

```
modelObj = sbiomodel('cell')
parameterObj = addparameter(modelObj, 'TF1', 0.01)
```



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



## addparameter (model, kineticlaw)

---

`parameterObj = addparameter(Obj, 'NameValue', ValueValue)` creates a parameter object, assigns a value (*NameValue*) to the property *Name*, assigns the value (*ValueValue*) to the property *Value*, and assigns the model object or the kinetic law object to the property *Parent*. In the model or kinetic law object (*Obj*), this method assigns the parameter object to the property *Parameters*, and returns the parameter object to a variable (*parameterObj*).

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

**Scope of a parameter** — A parameter can be *scoped* to either a model or a kinetic law.

- When a kinetic law searches for a parameter in its expression, it first looks in the parameter list of the kinetic law. If the parameter isn't found there, it moves to the model that the kinetic law object is in and looks in the model parameter list. If the parameter isn't found there, it moves to the model parent.
- When a rule searches for a parameter in its expression, it looks in the parameter list for the model. If the parameter isn't found there, it moves to the model parent. A rule cannot use a parameter that is scoped to a kinetic law. So for a parameter to be used in both a reaction rate equation and a rule, the parameter should be *scoped* to a model.

Additional parameter object properties can be viewed with the `get` command. Additional parameter object properties can be modified with the `set` command. The parameters of *Obj* can be viewed with `get(Obj, 'Parameters')`.

A SimBiology parameter object can be copied to a SimBiology model or kinetic law object with `copyobj`. A SimBiology parameter object can be removed from a SimBiology model or kinetic law object with `delete`.



# addparameter (model, kineticlaw)

---

## Method Summary

Methods for parameter objects

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

## Property Summary

Properties for parameter objects

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 SimBiology object type
UserData	Specify data to associate with object
Value	Assign value to parameter object
ValueUnits	Parameter value units

# addparameter (model, kineticlaw)

---

## Example

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

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

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

- 3 Add a parameter and assign it to the kinetic law object (kineticlawObj); add another parameter and assign to the model object (modelObj).

```
% Add parameter to kinetic law object  
parameterObj1 = addparameter (kineticlawObj, 'K1');
```

```
get (kineticlawObj, 'Parameters')
```

MATLAB returns:

SimBiology Parameter Array

Index:	Name:	Value:	ValueUnits:
1	K1	1	

```
% Add parameter with value 0.9 to model object  
parameterObj1 = addparameter (modelObj, 'K2', 0.9);
```

```
get (modelObj, 'Parameters')
```

MATLAB returns:

SimBiology Parameter Array

Index:	Name:	Value:	ValueUnits:
1	K2	1	

## See Also

[addreaction](#)

## 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>sbiselect</code> to find an object with a specific <i>NameValue</i> .
<i>speciesObj</i>	Species object.
<i>Stoichcoefficient</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*. In the reaction object, this method assigns the species object to the property *Products*, modifies the reaction equation in the property *Reaction* to include the new species, and adds the stoichiometric coefficient 1 to the property *Stoichiometry*.

When you define a reaction with a new species:

## addproduct (reaction)

---

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (*compObj*) exists in the model, the method creates a species object in that compartment.
- If there is more than one compartment object (*compObj*) in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Create and add a species object to a compartment object with the method `addspecies`.

`speciesObj = addproduct(reactionObj, speciesObj)`, in the species object (*speciesObj*), assigns the parent object of the *reactionObj* to the species property `Parent`. In the reaction object (*reactionObj*), it assigns the species object to the property `Products`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient 1 to the property `Stoichiometry`.

`speciesObj = addproduct(reactionObj, 'NameValue', Stoichcoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`. If `NameValue` is a cell array of species names, then `Stoichcoefficient` must be a vector of doubles with the same length as `NameValue`.

`speciesObj = addproduct(reactionObj, speciesObj, Stoichcoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species the reaction also uses the new name. You must however configure all other applicable elements such as rules that use the species, and the kinetic law object.

### Examples

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

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'A + C -> U');
```

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

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

### See Also

addspecies

# addreactant (reaction)

---

**Purpose** Add species object as reactant to reaction object

**Syntax**

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

**Arguments**

<i>reactionObj</i>	Reaction object.
<i>NameValue</i>	Name property of a species object. Enter a unique character string, for example, 'glucose'. A species object can be referenced by other objects using this property. You can use the function <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>StoichCoefficient</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*. In the reaction object, this method assigns the species object to the property *Reactants*, modifies the reaction equation in the property *Reaction* to include the new species, and adds the stoichiometric coefficient -1 to the property *Stoichiometry*.

When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (*compObj*) exists in the model, the method creates a species object in that compartment.

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Create and add a species object to a compartment object with the method `addspecies`.

`addreactant(reactionObj, speciesObj, StoichCoefficient)`, in the species object (`speciesObj`), assigns the parent object to the `speciesObj` property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `-1` to the property `Stoichiometry`. If `speciesObj` is a cell array of species objects, then `StoichCoefficient` must be a vector of doubles with the same length as `speciesObj`.

`addreactant(reactionObj, 'NameValue', StoichCoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`StoichCoefficient`) to the property `Stoichiometry`. If `NameValue` is a cell array of species names, then `StoichCoefficient` must be a vector of doubles with the same length as `NameValue`.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species the reaction also uses the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object.

See for more information on species names.

### Example

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

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'A -> U');
```

## addreactant (reaction)

---

**2** Modify the reaction of the `reactionObj` from  $A \rightarrow U$  to be  $A + 3C \rightarrow U$ .

```
speciesObj = addreactant(reactionObj, 'C', 3);
```

### See Also

`addspecies`



## Purpose

Create reaction object and add to model object

## Syntax

```
reactionObj = addreaction(modelObj, 'ReactionValue')
reactionObj = addreaction(modelObj, 'ReactantsValue',
    'ProductsValue')
reactionObj = addreaction(modelObj, 'ReactantsValue',
    RStoichCoefficients, 'ProductsValue', PStoichCoefficients)
reactionObj = addreaction(...'PropertyName', PropertyValue...)
```

## Arguments

*modelObj*

SimBiology model object.

*ReactionValue*

Specify the reaction equation. Enter a character string. A hyphen preceded by a space and followed by a right angle bracket (->) indicates reactants going forward to products. A hyphen with left and right angle brackets (<->) indicates a reversible reaction. Coefficients before reactant or product names must be followed by a space.

Examples are 'A -> B', 'A + B -> C', '2 A + B -> 2 C', and 'A <-> B'. Enter reactions with spaces between the species.

If there are multiple compartments, or to specify the compartment name, use *compartmentName.speciesName*.

Examples are 'cytoplasm.A -> cytoplasm.B', 'cytoplasm.A -> nucleus.A', and 'cytoplasm.A + cytoplasm.B -> nucleus.AB'.

*ReactantsValue*

A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments.

# addreaction (model)

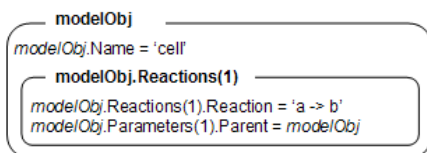
---

<i>ProductsValue</i>	A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments.
<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')
```



When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (*compObj*) exists in the model, the method creates a species object in that compartment.

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

You can manually add a species to a compartment object with the method `addspecies`.

You can add species to a reaction object using the methods `addreactant` or `addproduct`. You can remove species from a reaction object with the methods `rmreactant` or `rmproduct`. The property `Reaction` is modified by adding or removing species from the reaction equation.

You can copy a SimBiology reaction object to a model object with the function `copyobj`. You can remove the SimBiology reaction object from a SimBiology model object with the function `delete`.

You can view additional reaction object properties with the `get` command. For example, the reaction equation of `reactionObj` can be viewed with the command `get(reactionObj, 'Reaction')`. You can modify additional reaction object properties with the command `set`.

`reactionObj = addreaction(modelObj, 'ReactantsValue', 'ProductsValue')` creates a reaction object, assigns a value to the property `Reaction` using the reactant (`ReactantsValue`) and product (`ProductsValue`) names, assigns the species objects to the properties `Reactants` and `Products`, and assigns the model object to the property `Parent`. In the model object (`modelObj`), this method assigns the reaction object to the property `Reactions`, and returns the reaction object (`reactionObj`). The stoichiometric values are assumed to be 1.

`reactionObj = addreaction(modelObj, 'ReactantsValue', RStoichCoefficients, 'ProductsValue', PStoichCoefficients)` adds stoichiometric coefficients (`RStoichCoefficients`) for reactant species, and stoichiometric coefficients (`PStoichCoefficients`) for product species to the property `Stoichiometry`. The length of `Reactants`

## addreaction (model)

---

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).

---

**Note** If you use the `addreaction` method to create a reaction rate expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

### 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 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>get (any object)</code>	Get object properties
<code>rmproduct (reaction)</code>	Remove species object from reaction object products

rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

## Property Summary

Properties for reaction objects

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

# addreaction (model)

---

## Examples

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

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

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

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');  
  
reactionObj.KineticLaw property is configured to kineticlawObj.
```

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables ( $V_m$  and  $K_m$ ) and one species variable (S) that should be set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names  $V_m_d$ , and  $K_m_d$ , 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 =
```

$$V_{m,d} \cdot a / (K_m \cdot d + a)$$

**See Also**

addkineticlaw, addproduct, addreactant, rmproduct, rmreactant

# addrule (model)

---

**Purpose** Create rule object and add to model object

**Syntax**

```
ruleObj = addrule(modelObj, Rule)
ruleObj = addrule(modelObj, Rule, RuleType)
ruleObj = addrule(..., 'PropertyName', PropertyValue,...)
```

## Arguments

<i>modelObj</i>	Model object to which to add the rule.
<i>Rule</i>	String specifying the rule. For example, enter the algebraic rule 'Va*Ea + Vi*Ei - K2'.
<i>RuleType</i>	String specifying the type of rule. Choices are: <ul style="list-style-type: none"><li>• 'algebraic'</li><li>• 'initialAssignment'</li><li>• 'repeatedAssignment'</li><li>• 'rate'</li></ul> For more information, see RuleType

## 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, Rule)` constructs and returns *ruleObj*, a rule object. In *ruleObj*, the rule object, this method assigns the *modelObj* input argument to the Parent property, assigns the *Rule* input argument to the Rule property, and assigns 'initialAssignment' or 'algebraic' to the RuleType property. (This method assigns 'initialAssignment' for all assignment rules and 'algebraic' for all other rules.) In *modelObj*, the model object, this method assigns *ruleObj*, the rule object, to the Rules property.

*ruleObj* = `addrule(modelObj, Rule, RuleType)` in addition to the assignments above, assigns the *RuleType* input argument to the



`RuleType` property. For more information on the 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`.

---

**Note** If you use the `addrule` method to create an algebraic rule, rate rule, or repeated assignment rule, and the rule expression is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## 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
<code>get</code> (any object)	Get object properties
<code>set</code> (any object)	Set object properties

# addrule (model)

---

## Property Summary

Properties for rule objects

Active	Indicate object in use during simulation
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 SimBiology object type
UserData	Specify data to associate with object

## Examples

Add a rule with the default RuleType.

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

```
modelObj = sbiomodel('cell');  
ruleObj = addrule(modelObj, '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 a rule with the RuleType property set to rate.

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

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

- 2 Add a rule which defines that the quantity of a species c. In the rule expression, k is the rate constant for a -> b.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

- 3 Change the RuleType from default ('algebraic') to 'rate', and verify using the get command.

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
Active: 1
Annotation: ''
Name: ''
Notes: ''
Parent: [1x1 SimBiology.Model]
Rule: 'c = k*(a+b)'
RuleType: 'rate'
Tag: ''
Type: 'rule'
UserData: []
```

## addrule (model)

---

**See Also**      `copyobj, delete, sbiomodel`

# addspecies (model, compartment)

---

## Purpose

Create species object and add to compartment object within model object

## Syntax

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

## Arguments

<i>compObj</i>	Compartment object.
<i>modelObj</i>	Model object containing zero or one compartment.
<i>NameValue</i>	Name for a species object. Enter a character string unique among species within <i>modelObj</i> or <i>compObj</i> . Species objects are identified by name within Event, ReactionRate, and Rule property strings. For information on naming species, see Name.  You can use the function <code>sbiiselect</code> 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.
<i>PropertyName</i>	Enter the name of a valid property. Valid property names are listed in “Property Summary” on page 4-60.
<i>PropertyValue</i>	Enter the value for the property specified in <i>PropertyName</i> . Valid property values are listed on each property reference page.

# addspecies (model, compartment)

---

## Description

`speciesObj = addspecies(compObj, 'NameValue')` creates `speciesObj`, a species object, and adds it to `compObj`, a compartment object. In the species object, this method assigns `NameValue` to the `Name` property, assigns `compObj` to the `Parent` property, and assigns 0 to the `InitialAmount` property. In the compartment object, this method adds the species object to the `Species` property.

`speciesObj = addspecies(compObj, 'NameValue', InitialAmountValue)`, in addition to the above, assigns `InitialAmountValue` to the `InitialAmount` property for the species object.

`speciesObj = addspecies(modelObj, 'NameValue')` creates `speciesObj`, a species object, and adds it to `compObj`, the compartment object in `modelObj`, a model object. If `modelObj` does not contain any compartments, it creates `compObj` with a `Name` property of 'unnamed'. In the species object, this method assigns `NameValue` to the `Name` property, assigns `compObj` to the `Parent` property, and assigns 0 to the `InitialAmount` property. In the compartment object, this method adds the species object to the `Species` property.

`speciesObj = addspecies(modelObj, 'NameValue', InitialAmountValue)`, in addition to the above, assigns `InitialAmountValue` to the `InitialAmount` property for the species object.

You can also add a species to a reaction using the methods `addreactant` and `addproduct`.

A species object must have a unique name at the level at which it is created. For example, a compartment object cannot contain two species objects named H2O. However, another compartment can have a species named H2O.

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

# addspecies (model, compartment)

---

`speciesObj = addspecies(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The property summary on this page shows the list of properties.

If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name. For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

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

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

Species names are automatically updated for reactions that use `MassAction` kinetic law.

## Method Summary

Methods for species objects

<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>get</code> (any object)	Get object properties

# addspecies (model, compartment)

---

rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

## Property Summary

Properties for species objects	
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 SimBiology object type
UserData	Specify data to associate with object

## Examples

Add two species to a model, where one is a reactant and the other is the enzyme catalyzing the reaction.

**1** Create a model object named `my_model` and add a compartment object.

```
modelObj = sbiomodel ('my_model');  
compObj = addcompartment(modelObj, 'comp1');
```



## addspecies (model, compartment)

---

- 2** Add two species objects named `glucose_6_phosphate` and `glucose_6_phosphate_dehydrogenase`.

```
speciesObj1 = addspecies (compObj, 'glucose_6_phosphate');
speciesObj2 = addspecies (compObj, ...
                        'glucose_6_phosphate_dehydrogenase');
```

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

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

MATLAB returns:

```
ans =

    100
```

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

```
get(compObj, 'Species')
```

MATLAB returns:

SimBiology Species Array

Index:	Name:	InitialAmount:	InitialAmountUnits:
1	glucose_6_phosphate	100	
2	glucose_6_phosphate_dehydrogenase	0	

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

```
get(compObj.Species(1))
    Annotation: ''
    BoundaryCondition: 0
    ConstantAmount: 0
    InitialAmount: 100
    InitialAmountUnits: ''
    Name: 'glucose_6_phosphate'
```

## addspecies (model, compartment)

---

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

**See Also**      addcompartment, addproduct, addreactant, addreaction, get, set

**Purpose** Add variant to model

**Syntax**  
`variantObj = addvariant(modelObj, 'NameValue')`  
`variantObj2 = addvariant(modelObj, variantObj)`

## Arguments

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

## Description

`variantObj = addvariant(modelObj, 'NameValue')` creates a SimBiology variant object (*variantObj*) with the name *NameValue* and adds the variant object to the SimBiology model object *modelObj*. The variant object Parent property is assigned the value of *modelObj*.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see [Variant object](#).

`variantObj2 = addvariant(modelObj, variantObj)` adds a SimBiology variant object (*variantObj*) to the SimBiology model object and returns another variant object *variantObj2*. The variant object *variantObj2* Parent property is assigned the value of *modelObj*.

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

---

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

---

## addvariant (model)

---

To view the variants stored on a model object, use the `getvariant` method. To copy a variant object to another model, use `copyobj`. To remove a variant object from a SimBiology model, use the `delete` method.

### Examples

- 1 Create a model containing one species.

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

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

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

### See Also

`addcontent`, `commit`, `copyobj`, `delete`, `getvariant`

**Purpose** Commit variant contents to model

**Syntax** `commit(variantObj, modelObj)`

## Arguments

<i>modelObj</i>	Specify the model object to which you want to commit a variant.
<i>variantObj</i>	Variant object to commit to the model object.

## Description

`commit(variantObj, modelObj)` commits the `Contents` property of a SimBiology variant object (*variantObj*) to the model object *modelObj*. The property values stored in the variant object replace the values stored in the model.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant` object.

The `Contents` are set on the model object in order of occurrence, with duplicate entries overwriting. If the `commit` method finds an incorrectly specified entry, an error occurs and the remaining properties defined in the `Contents` property are not set.

## Examples

- 1 Create a model containing one species.

```
modelObj = sbiomodel('mymodel');  
compObj = addcompartment(modelObj, 'comp1');  
speciesObj = addspecies(compObj, 'A', 10);
```

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

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

- 3 Commit the contents of the variant (*variantObj*).

## commit (variant)

---

```
commit (variantObj, modelObj);
```

**See Also**      addvariant, Variant object

## Purpose

Options for compartments

## Description

The SimBiology compartment object represents a container for species in a model. Compartment size can vary or remain constant during a simulation. All models must have at least one compartment and all species in a model must be assigned to a compartment. Compartment names must be unique within a model.

Compartments allow you to define the size (**Capacity**) of physically isolated regions that may affect simulation, and associate pools of species within those regions. You can specify or change **Capacity** using rules, events, and variants, similar to species amounts or parameter values.

The model object stores compartments as a flat list. Each compartment stores information on its own organization; in other words a compartment has information on which compartment it lives within (**Owner**) and who it contains (**Compartments**).

The flat list of compartments in the model object lets you vary the way compartments are organized in your model without invalidating any expressions.

To add species that participate in reactions, add the reaction to the model using the `addreaction` method. When you define a reaction with a new species:

- If no compartment objects exist in the model, the `addreaction` method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object exists in the model, the method creates a species object in that compartment.
- If there is more than one compartment object in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

# Compartment object

---

Alternatively, create and add a species object to a compartment object, using the `addspecies` method at the command line.

When you use the SimBiology desktop to create a new model, it adds an empty compartment (*unnamed*), to which you can add species.

You can specify reactions that cross compartments using the syntax `compartment1Name.species1Name -> compartment2Name.species2Name`. If you add a reaction that contains species from different compartments, and the reaction rate dimensions are concentration/time, all reactants should be from the same compartment.

In addition, if the reaction is reversible then there are two cases:

- If the kinetic law is `MassAction`, and the reaction rate dimensions are concentration/time, then the products must be from the same compartment.
- If the kinetic law is not `MassAction`, then both reactants and products must be in the same compartment.

See “Property Summary” on page 4-69 for links to compartment property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

`addcompartment (model, compartment)`

Create compartment object



## Method Summary

Methods for compartment objects

<code>addcompartment (model, compartment)</code>	Create compartment object
<code>addspecies (model, compartment)</code>	Create species object and add to compartment object within 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>get (any object)</code>	Get object properties
<code>rename (compartment, parameter, species)</code>	Rename object and update expressions
<code>reorder (model, compartment)</code>	Reorder component lists
<code>set (any object)</code>	Set object properties

## Property Summary

Properties for compartment objects

<code>Capacity</code>	Compartment capacity
<code>CapacityUnits</code>	Compartment capacity units
<code>Compartments</code>	Array of compartments in model or compartment
<code>ConstantCapacity</code>	Specify variable or constant compartment capacity
<code>Name</code>	Specify name of object
<code>Notes</code>	HTML text describing SimBiology object
<code>Owner</code>	Owning compartment

# Compartment object

---

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

## See Also

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

## Purpose

Solver settings information for model simulation

## Description

The SimBiology configset object, also known as the configuration set object, contains the options that the solver uses during simulation of the model object. The configuration set object contains the following options for you to choose:

- Type of solver
- Stop time for the simulation
- Solver error tolerances, and for ode solvers — the maximum time step the solver should take
- Whether to perform sensitivity analysis during simulation
- Whether to perform dimensional analysis and unit conversion during simulation
- Species and parameter input factors for sensitivity analysis

A SimBiology model can contain multiple configsets with one being active at any given time. The active configset contains the settings that are used during the simulation. Use the method `setactiveconfigset` to define the active configset. Use the method `getConfigset` to return a list of configsets contained by a model. Use the method `addconfigset` to add a new configset to a model.

See “Property Summary” on page 4-72 for links to configset object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

## Constructor Summary

`addconfigset (model)`

Create configuration set object and add to model object

# Configset object

---

## Method Summary

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

## Property Summary

Active	Indicate object in use during simulation
CompileOptions	Dimensional analysis and unit conversion options
MaximumNumberOfLogs	Maximum number of logs criteria to stop simulation
MaximumWallClock	Maximum elapsed wall clock time to stop simulation
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
StartTime	Start time for initial dose time
StopTime	Simulation time criteria to stop simulation

TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type

## See Also

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

# construct (PKModelDesign)

---

**Purpose** Construct SimBiology model from PKModelDesign object

**Syntax**

```
[modelObj, pkModelMapObject] =  
construct(pkModelDesignObject)  
[modelObj, pkModelMapObject,  
CovModelObj] = construct(pkModelDesignObject)
```

**Arguments**

<i>modelObj</i>	SimBiology model object specifying a pharmacokinetic model.
<i>pkModelMapObject</i>	Defines the roles of the components in <i>modelObj</i> . For details, see PKModelMap object.
<i>CovModelObj</i>	Defines the relationship between parameters and covariates. For details, see CovariateModel object.

**Description**

[*modelObj*, *pkModelMapObject*] = `construct(pkModelDesignObject)` constructs a SimBiology model object, *modelObj*, containing the model components (such as compartments, species, reactions, and rules) required to represent the pharmacokinetic model specified in *pkModelDesignObject*. It also constructs *pkModelMapObject*, a PKModelMap object, which defines the roles of the model components.

The newly constructed model object, *modelObj*, is named 'Generated Model' (which you can change). It contains one compartment for each compartment specified in the PKCompartment property of *pkModelDesignObject*. Each compartment contains a species that represents a drug concentration. The compartments are connected with reversible reactions that models flux between compartments.

[*modelObj*, *pkModelMapObject*, *CovModelObj*] = `construct(pkModelDesignObject)` constructs *CovModelObj*, a CovariateModel object, which defines the relationship between parameters and covariates. Within the Expression property of

*CovModelObj*, each parameter being estimated has an expression of the form  $\text{parameterName} = \exp(\text{theta1} + \text{eta1})$  (without covariate dependencies), where *theta1* is a fixed effect, and *eta1* is a random effect. You can modify the expressions to add covariate dependencies. For details, see *CovariateModel* object.

### See Also

*PKModelDesign* object | *PKModelMap* object | *CovariateModel* object

### How To

- “Creating Pharmacokinetic Model Using the Command Line”
- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”

# constructDefaultFixedEffectValues (covmodel)

---

**Purpose** Create initial estimate vector needed for fit

**Syntax** `FEInitEstimates =  
constructDefaultFixedEffectValues(CovModelObj  
j)`

**Description** `FEInitEstimates =  
constructDefaultFixedEffectValues(CovModelObj)` creates `FEInitEstimates`, a structure containing the initial estimates for the fixed effects in `CovModelObj`, a `CovariateModel` object. These initial estimates are set to a default of zero, but you can edit these estimates. The number and names of the fields in the `FEInitEstimates` structure matches the number and names of fixed effects (theta values) in the Expression property of `CovModelObj`.

---

**Tip** After creating the `FEInitEstimates` structure, you can edit it and use it to change the `FixedEffectValues` property of `CovModelObj`, before using the object as an input argument to `sbionlmeFit` or `sbionlmeFitSA`.

---

**See Also** `CovariateModel` | `Expression` | `FixedEffectValues` | `sbionlmeFit` | `sbionlmeFitSA`

**How To**

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”



**Purpose** Copy SimBiology object and its children

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

## Arguments

*Obj* Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, reaction, rule, species, or variant object.

*parentObj*

<b>If <i>copiedObj</i> is...</b>	<b><i>parentObj</i> must be...</b>
configuration set, event, reaction, rule, or variant object	model object
compartment object	compartment or model object
species object	compartment object
parameter object	model or kinetic law object
kinetic law object	reaction object
model object or abstract kinetic law object	sbioroot

*modelObj* Model object to be copied.

*copiedObj* Output returned by the `copyobj` method with the parent set as specified in input argument (*parentObj*).

## Description

`copiedObj = copyobj(Obj, parentObj)` makes a copy of a SimBiology object (*Obj*) and returns a pointer to the copy (*copiedObj*). In the copied object (*copiedObj*), this method assigns a value (*parentObj*) to the property `Parent`.

## copyobj (any object)

---

`copiedObj = copyobj(modelObj)` makes a copy of a model object (`modelObj`) and returns the copy (`copiedObj`). In the copied model object (`copiedObj`), this method assigns the root object to the property `Parent`.

---

**Note** When the `copyobj` method copies a model, it resets the `StatesToLog` property to the default value. Similarly, the `Inputs` and `Outputs` properties are not copied but rather left empty. Thus, when you simulate a copied model you see results for the default states, unless you manually update these properties.

---

### Examples

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

- 1 Create a model object and add a reaction object.

```
modelObj1 = sbiomodel('cell1');  
reactionObj = addreaction(modelObj1, 'a -> b');
```

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

```
modelObj2 = sbiomodel('cell2');  
reactionObjCopy = copyobj(reactionObj, modelObj2);  
modelObj2.Reactions
```

SimBiology Reaction Array

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

### See Also

`sbiomodel`, `sbioroot`

**Purpose** Define relationship between parameters and covariates

**Description** CovariateModel defines the relationship between estimated parameters and covariates.

---

**Tip** Use a CovariateModel object as an input argument to `sbionlmeFit` or `sbionlmeFitsa` to fit a model with covariate dependencies. Before using the CovariateModel object with either fitting function, set the `FixedEffectValues` property to specify the initial estimates for the fixed effects.

---

**Construction** `CovModelObj = CovariateModel` creates an empty CovariateModel object.

`CovModelObj = CovariateModel(Expression)` creates a CovariateModel object with its `Expression` property set to *Expression*, a string or cell array of strings, where each string represents the relationship between a parameter being estimated and one or more covariates. *Expression* must denote fixed effects with the prefix `theta`, and random effects with the prefix `eta`. Each string in *Expression* must be in the form:

```
parameterName = relationship
```

This example of an expression string defines the relationship between a parameter (`volume`) and a covariate (`weight`), with fixed effects, but no random effects:

```
Expression = {'volume = theta1 + theta2*weight'};
```

This table illustrates expression formats for some common parameter-covariate relationships.

# CovariateModel object

---

Parameter-Covariate Relationship	Expression Format
Linear with random effect	$C1 = \text{theta1} + \text{theta2} * \text{WEIGHT} + \text{eta1}$
Exponential without random effect	$C1 = \exp(\text{theta\_C1} + \text{theta\_C1\_WT} * \text{WEIGHT})$
Exponential, WEIGHT centered by mean, has random effect	$C1 = \exp(\text{theta1} + \text{theta2} * (\text{WEIGHT} - \text{mean}(\text{WEIGHT})) + \text{eta1})$
Exponential, $\log(\text{WEIGHT})$ , which is equivalent to power model	$C1 = \exp(\text{theta1} + \text{theta2} * \log(\text{WEIGHT}) + \text{eta1})$
Exponential, dependent on WEIGHT and AGE, has random effect	$C1 = \exp(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$

---

**Tip** To simultaneously fit data from multiple dose levels, use a CovariateModel object as an input argument to `sbionlmeFit`, and omit the random effect (`eta`) from the Expression property in the CovariateModel object.

---

---

**Tip** Use the `getCovariateData` method of a PKData object to view the covariate data when writing equations for the *Expression* input argument.

---

---

**Note** You can also construct a `CovariateModel` object using the `construct` method of a `PKModelDesign` object. However, the `Expression` property of the `CovariateModel` object does not include covariate dependencies. You can modify the expressions to add covariate dependencies. For details, see `Expression`.

---

## Method Summary

<code>constructDefaultFixedEffectValues (covmodel)</code>	Create initial estimate vector needed for fit
<code>verify (covmodel)</code>	Check covariate model for errors

## Property Summary

<code>CovariateLabels (CovariateModel)</code>	Labels for covariates in <code>CovariateModel</code> object
<code>Expression (CovariateModel)</code>	Define relationship between parameters and covariates
<code>FixedEffectDescription (CovariateModel)</code>	Descriptions of fixed effects in <code>CovariateModel</code> object
<code>FixedEffectNames (CovariateModel)</code>	Names of fixed effects in <code>CovariateModel</code> object
<code>FixedEffectValues (CovariateModel)</code>	Values for initial estimates of fixed effects in <code>CovariateModel</code> object
<code>ParameterNames (CovariateModel)</code>	Names of parameters in <code>CovariateModel</code> object
<code>RandomEffectNames (CovariateModel)</code>	Names of random effects in <code>CovariateModel</code> object

## Examples

Create a `CovariateModel` object and set the `Expression` property to define the relationship between two parameters (clearance and volume)

# CovariateModel object

---

and two covariates (weight and age) using fixed effects (thetas) and random effects (etas):

```
covModelObj = CovariateModel  
covModelObj.Expression = {'CL = theta1 + theta2*WT + eta1', 'V = theta3 +
```

## See Also

`construct` | `getCovariateData` | PKData object | PKModelDesign object | `sbionlmeFit` | `sbionlmeFitSA`

## How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”

**Purpose** Delete SimBiology object

**Syntax** `delete(Obj)`

**Arguments**

*Obj* abstract kinetic law, configuration set, event, kinetic law, model, parameter, reaction, rule, SimData, species, unit, unit prefix, or variant object.

**Description**

`delete(Obj)` removes an object (*Obj*) from its parent.

- If *Obj* is a model object, the model is deleted from the root object. `delete` removes all references to the model both at the command line and in the SimBiology desktop.
- 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, an error occurs because the parameter cannot be found.
- If *Obj* is a reaction object, this method deletes the object, but the species objects that were being used by the reaction object are not deleted.
- If *Obj* is an abstract kinetic law object and there is a kinetic law object referencing it, this method returns an error.
- If *Obj* is a SimBiology configuration set object, and it is the active configuration set object, this method, after deleting the object, makes the default configuration set object active. Note that you cannot delete the default configuration set.
- You cannot delete the SimBiology root.

## delete (any object)

---

You can also delete all model objects from the root with one call to the `sbioreset` function.

### 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, compartment, event, kinetic law, model, parameter, reaction, rule, species, or unit.

**Description**

Display the SimBiology object array. `display(Obj)` is called for the SimBiology object, *Obj* when the semicolon is not used to terminate a statement. The display of *Obj* gives a brief summary of the *Obj* configuration. You can view a complete list of *Obj* properties with the command `get`. You can modify all *Obj* properties that can be changed, with the command `set`.

**Examples**

```
modelObj = sbiomodel('cell')
reactionObj = addreaction(modelObj, 'A + B -> C')
```

# Event object

---

**Purpose** Store event information

**Description** Events are used to describe sudden changes in model behavior. An event lets you specify discrete transitions in model component values that occur when a user-specified condition become true. You can specify that the event occurs at a particular time, or specify a time-independent condition.

For details on how events are handled during a simulation, see “Event Object”.

See “Property Summary” on page 4-87 for links to event property reference pages.

Properties define the characteristics of an object. For example, an event object includes properties that allow you to specify the conditions to trigger an event (Trigger), and what to do after the event is triggered (EventFcn). Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

<b>Constructor Summary</b>	addevent (model)	Add event object to model object
----------------------------	------------------	----------------------------------

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

## Property Summary

Active	Indicate object in use during simulation
EventFcns	Event expression
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Trigger	Event trigger
Type	Display SimBiology object type
UserData	Specify data to associate with object

## See Also

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

# export (model)

---

**Purpose** Export SimBiology model

**Syntax**

```
exportedModel = export(model)
exportedModel = export(model,editvals)
exportedModel = export(model,editvals,editdoses)
```

## Arguments

model	Model object.
editvals	Vector of species, parameter, and compartment objects that are editable in the exported model.
editdoses	Vector of dose objects that are editable in the exported model.

## Description

`exportedModel = export(model)` exports a model, including all its active doses, and returns an exported model object, `SimBiology.export.Model`. By default, all species, parameters, compartments, and doses are editable in the exported model.

The exported SimBiology model object is a subclass of `hgsetget`. Thus, you can view properties for an exported model object with the `get` command, and modify properties for an exported model object with the `set` command.

`exportedModel = export(model,editvals)` specifies a vector of species, parameter, and compartment objects that are editable in the exported model. All doses in the exported model are editable.

`exportedModel = export(model,editvals,editdoses)` additionally specifies a vector of dose objects that are editable in the exported model.

## Method Summary

Methods for exported model objects

accelerate	Prepare exported SimBiology model for acceleration
getdose	Return exported SimBiology model dose object
getIndex	Get indices into ValueInfo and InitialValues properties
isAccelerated	Determine whether an exported SimBiology model is accelerated
simulate	Simulate exported SimBiology model

## Examples

Export a SimBiology model object.

```
modelObj = sbmlimport('lotka');  
exportedModel = export(modelObj)
```

```
exportedModel =
```

```
    SimBiology.export.Model
```

```
        Name: lotka
```

```
        ExportTime: 27-Jun-2012 14:49:46
```

```
        ExportNotes:
```

Display the editable values (species, parameters, and compartments) information for the exported model object.

```
{exportedModel.ValueInfo.Name}
```

```
ans =
```

```
    'unnamed'    'x'    'y1'    'y2'    'z'    'c1'    'c2'    'c3'
```

## export (model)

---

There are 8 editable values in the exported model. Export the model again, allowing only the parameters (c1, c2, and c3) to be editable.

```
parameters = sbioselect(modelObj, 'Type', 'parameter');
exportedModelParam = export(modelObj, parameters);
{exportedModelParam.ValueInfo.Name}
```

```
ans =
```

```
    'c1'    'c2'    'c3'
```

Export the model a third time, allowing the parameters and species to be editable.

```
PS = sbioselect(modelObj, 'Type', {'species', 'parameter'});
exportedModelPS = export(modelObj, PS);
{exportedModelPS.ValueInfo.Name}
```

```
ans =
```

```
    'x'    'y1'    'y2'    'z'    'c1'    'c2'    'c3'
```

**See Also** [SimBiology.export.Model](#) |

### Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

## Purpose

Get object properties

## Syntax

```
PropertyValue = get(Obj, 'PropertyName')  
objProperties = get(Obj)
```

## Arguments

*PropertyValue* Value defined for '*PropertyName*'

*Obj* Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, PKCompartment, PKData, PKModelDesign, PKModelMap, reaction, rule, SimData, species, or variant object.

'*PropertyName*' Name of the property to get.

*objProperties* Struct containing properties and values for the object, *Obj*.

## Description

*PropertyValue* = get(*Obj*, '*PropertyName*') gets the value '*PropertyValue*' of the object, *Obj*'s *PropertyName* property.

*objProperties* = get(*Obj*) gets the properties for the object, *Obj*, and returns it to *objProperties*.

## Examples

1 Create a model object.

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

## get (any object)

---

**2** Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

**3** Set the ConstantValue property of the parameter object to false and verify.

MATLAB returns 1 for true and 0 for false.

```
set (parameterObj, 'ConstantValue', false);  
get(parameterObj, 'ConstantValue')
```

MATLAB returns

```
ans =
```

```
0
```

### See Also

getadjacencymatrix, getconfigset, getdata, getparameters,  
getsensmatrix, getspecies, getstoichmatrix, set



**Purpose** Get adjacency matrix from model object

**Syntax**

```
M = getadjacencymatrix(modelObj)  
[M, Headings] = getadjacencymatrix(modelObj)  
[M, Headings, Mask] = getadjacencymatrix(modelObj)
```

## Arguments

<i>M</i>	Adjacency matrix for <i>modelObj</i> .
<i>modelObj</i>	Specify the model object.
<i>Headings</i>	Return row and column headings. If species are in multiple compartments, species names are qualified with the compartment name in the form <code>compartmentName.speciesName</code> . For example, <code>nucleus.DNA</code> , <code>cytoplasm.mRNA</code> .
<i>Mask</i>	Return 1 for the species object and 0 for the reaction object to <i>Mask</i> .

## Description

`getadjacencymatrix` returns the adjacency matrix for a model object.

`M = getadjacencymatrix(modelObj)` returns an adjacency matrix for the model object (*modelObj*) to *M*.

An adjacency matrix is defined by listing all species contained by *modelObj* and all reactions contained by *modelObj* column-wise and row-wise in a matrix. The reactants of the reactions are represented in the matrix with a 1 at the location of [row of species, column of reaction]. The products of the reactions are represented in the matrix with a 1 at the location of [row of reaction, column of species]. All other locations in the matrix are 0.

`[M, 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*.

## getadjacencymatrix (model)

---

`[M, Headings, Mask] = getadjacencymatrix(modelObj)` returns an array of 1s and 0s to *Mask*, where a 1 represents a species object and a 0 represents a reaction object.

### Examples

**1** Read in *m1*, a model object, using `sbmlimport`:

```
m1 = sbmlimport('lotka.xml');
```

**2** Get the adjacency matrix for *m1*:

```
[M, Headings] = getadjacencymatrix(m1)
```

### See Also

`getstoichmatrix`

**Purpose** Get configuration set object from model object

**Syntax**

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

## Arguments

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

## Description

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

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

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

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

Use the setactiveconfigset function to define the active configset. *modelObj* always contains at least one configset object with the name configured to 'default'. Additional configset objects can be added to *modelObj* with the method addconfigset.

## Examples

1 Retrieve the default configset object from the *modelObj*.

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

## getConfigset (model)

---

```
Configuration Settings - default (active)
  SolverType:          ode15s
  StopTime:           10

SolverOptions:
  AbsoluteTolerance:  1.000000e-06
  RelativeTolerance:  1.000000e-03
  SensitivityAnalysis: false

RuntimeOptions:
  StatesToLog:        all

CompileOptions:
  UnitConversion:     false
  DimensionalAnalysis: true

SensitivityAnalysisOptions:
  Inputs:             0
  Outputs:            0
```

### 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]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
  SolverOptions: [1x1 SimBiology.SSASolverOptions]
  SolverType: 'ssa'
  StopTime: 10
MaximumNumberOfLogs: Inf
MaximumWallClock: Inf
TimeUnits: 'second'
```

Type: 'configset'

**See Also**      addconfigset, removeconfigset, setactiveconfigset

# getCovariateData (pkdata)

---

**Purpose** Create design matrix needed for fit

**Syntax** `CovData = getCovariateData(PKDataObj)`

**Description** `CovData = getCovariateData(PKDataObj)` creates *CovData*, a dataset array containing only the covariate data from the data set in *PKDataObj*, a PKData object. *CovData* contains one row for each individual and one column for each covariate.

---

**Tip** Use the `getCovariateData` method to view the covariate data when writing equations for the `Expression` property of a `CovariateModel` object.

---

**See Also** `CovariateModel` | `Expression` | `PKData`

**How To**

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”

**Purpose** Get data from SimData object array

**Syntax**  
`[t, x, names] = getdata(simDataObj)`  
`[Out] = getdata(simDataObj, 'FormatValue')`

## Arguments

### Output Arguments

- |                 |                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>t</i>        | An n-by-1 vector of time points.                                                                                                                                                                                                                                                                                                                                                    |
| <i>x</i>        | An n-by-m data array. <i>t</i> and <i>names</i> label the rows and columns of <i>x</i> respectively.                                                                                                                                                                                                                                                                                |
| <i>names</i>    | An m-by-1 cell array of names.                                                                                                                                                                                                                                                                                                                                                      |
| <i>Metadata</i> | When used with the 'nummetadata' input argument, <i>Metadata</i> contains a cell array of metadata structures. The elements of <i>Metadata</i> label the columns of <i>x</i> .                                                                                                                                                                                                      |
| <i>Out</i>      | Data returned in the format specified in 'FormatValue', shown in "Input Arguments" on page 4-99. Depending on the specified 'FormatValue', <i>Out</i> contains one of the following: <ul style="list-style-type: none"><li>• Structure array</li><li>• SimData object</li><li>• Time series object</li><li>• Combined time series object from an array of SimData objects</li></ul> |

### Input Arguments

- |                    |                                                             |
|--------------------|-------------------------------------------------------------|
| <i>simDataObj</i>  | SimData object. Enter a variable name for a SimData object. |
| <i>FormatValue</i> | Choose a format from the following table.                   |

## getdata (SimData)

---

<b>FormatValue</b>	<b>Description</b>
'num'	Specifies the format that lets you return data in numeric arrays. This is the default when <code>getdata</code> is called with multiple output arguments.
'nummetadata'	Specifies the format that lets you return a cell array of metadata structures in <i>metadata</i> instead of names. The elements of <i>metadata</i> label the columns of <i>x</i> .
'numqualnames'	Specifies the format that lets you return qualified names in <i>names</i> to resolve ambiguities.
'struct'	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
'simdata'	Specifies the format that lets you return data in a new <code>SimData</code> object. This format is more useful for <code>SimData</code> methods other than <code>getdata</code> .



<b>FormatValue</b>	<b>Description</b>
'ts'	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in <code>simDataObj</code> .
'tslumped'	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

## Description

`[t, x, names] = getdata(simDataObj)` gets simulation time and state data from the SimData object `simDataObj`. When `simDataObj` contains more than one element, the outputs `t`, `x`, `names` are cell arrays in which each cell contains data for the corresponding element of `simDataObj`.

`[Out] = getdata(simDataObj, 'FormatValue')` returns the data in the specified format. Valid formats are listed in “Input Arguments” on page 4-99.

## Examples

### Simulating and Retrieving Data

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

```
sbioloadproject('radiodecay');
simDataObj = sbiosimulate(m1);
```

- 2 Get all the simulation data from the SimData object.

```
[t x names] = getdata(simDataObj);
```

# getdata (SimData)

---

## Retrieving Data for Ensemble Runs

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

```
sbioloadproject('radiodecay');
```

- 2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj = getconfigset(m1);  
set(csObj, 'SolverType', 'ssa');  
simDataObj = sbioenssemblerun(m1, 10);
```

- 3 Get all the simulation data from the `SimData` object.

```
tsObjs = getdata(simDataObj(1:5), 'ts');
```

## See Also

`display`, `get`, `resample`, `select`, `selectbyname`, `setactiveconfigset`  
MATLAB function struct

<b>Purpose</b>	Return exported SimBiology model dose object
<b>Syntax</b>	<pre>doses = getdose(model) doses = getdose(model,doseName)</pre>
<b>Description</b>	<p><code>doses = getdose(model)</code> returns all the <code>SimBiology.export.Dose</code> objects associated with the exported model.</p> <p><code>doses = getdose(model,doseName)</code> returns the export dose object with the <code>Name</code> property matching <code>doseName</code>.</p>
<b>Input Arguments</b>	<p><b>model</b> SimBiology.export.Model object.</p> <p><b>doseName</b> String containing a dose name to match against the <code>Name</code> property of the export dose objects in <code>model</code>.</p> <p><b>Default:</b> All dose objects.</p>
<b>Output Arguments</b>	<p><b>doses</b> Export dose objects in <code>model</code>, or the export dose object with <code>Name</code> property <code>doseName</code>.</p>
<b>Examples</b>	<p><b>Exported SimBiology Model Dose Objects</b></p> <p>Open a sample SimBiology model project, and export the included model object.</p> <pre>sbioloadproject('AntibacterialPKPD') em = export(m1);</pre> <p>Display the editable doses in the exported model object.</p> <pre>doses = getdose(em);</pre> <pre>doses =</pre>

# SimBiology.export.Model.getdose

---

```
1x4 SimBiology.export.RepeatDose handle
Package: SimBiology.export
```

```
Properties:
  Interval
  RepeatCount
  StartTime
  TimeUnits
  Amount
  AmountUnits
  DurationParameterName
  LagParameterName
  Name
  Notes
  Parent
  Rate
  RateUnits
  TargetName
```

Methods, Events, Superclasses

The exported model has 4 repeated dose objects. Display the dose names.

```
{doses.Name}
```

```
ans =
```

```
'250 mg bid'    '250 mg tid'    '500 mg bid'    '500 mg tid'
```

Extract only the 3rd dose object from the exported model object.

```
dose3 = getdose(em,'500 mg bid')
```

```
dose3 =
```

```
SimBiology.export.RepeatDose handle
```

Package: SimBiology.export

Properties:

```
Interval: 12
RepeatCount: 27
StartTime: 0
TimeUnits: 'hour'
Amount: 500
AmountUnits: 'milligram'
DurationParameterName: 'TDose'
LagParameterName: ''
Name: '500 mg bid'
Notes: ''
Parent: 'Antibacterial'
Rate: 0
RateUnits: ''
TargetName: 'Central.Drug'
```

Methods, Events, Superclasses

## See Also

[SimBiology.export.Model](#) | [SimBiology.export.Dose](#) |

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# getdose (model)

---

**Purpose** Return SimBiology dose object

**Syntax**  
`doseObj = getdose(modelObj)`  
`doseObj = getdose(modelObj, 'DoseName')`

**Arguments**

<i>modelObj</i>	Selects a model object that contains a dose object.
<i>DoseName</i>	Name of a dose object contained in a model object. <i>DoseName</i> is from the dose object property, Name.

**Outputs**

<i>doseObj</i>	ScheduleDose or RepeatDose object retrieved from a model object. A RepeatDose or ScheduleDose object defines an increase (dose) to a species amount during a simulation.
----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Description**

`doseObj = getdose(modelObj)` returns a Simbiology dose object (*doseObj*) contained in a Simbiology model object (*modelObj*).

`doseObj = getdose(modelObj, 'DoseName')` returns a SimBiology dose object (*modelObj*) with the name *DoseName*.

**Examples** Get a dose object from a model object.

**1** Create a model object, and then add a dose object to the model object.

```
modelObj = sbiomodel('mymodel');  
doseObj = adddose(modelObj, 'dose1');
```

**2** Get the dose object from a model object.

```
myModelDose = getdose(modelObj);
```

### See Also

Model object methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

`ScheduleDose` object and `RepeatDose` object methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

# Model.getequations

---

**Purpose** Return system equations for model object

**Syntax**

```
equations = getequations(modelobj)
equations = getequations(modelobj,configsetobj,variantobj,
    doseobj)
```

**Description** `equations = getequations(modelobj)` returns `equations`, a string containing the system of equations that represent `modelobj`, a `Model` object. The active `Configset` object is used to generate the equations and must specify a deterministic solver.

```
equations =
getequations(modelobj,configsetobj,variantobj,doseobj)
```

returns the system of equations that represent the model specified by a `Model` object, `Variant` objects, and dose objects (`RepeatDose` or `ScheduleDose`). The `Configset` object, `configsetobj`, is used to generate the equations and must specify a deterministic solver.

**Tips** Use `getequations` to see the system of equations that represent a model for:

- Publishing purposes
- Model debugging

## Input Arguments

### **modelobj**

Object of the `Model` class.

---

**Note** If using `modelobj` as the only input argument, the active `Configset` object must specify a deterministic solver.

---

### **configsetobj**

Object of the `Configset` class. This object must specify a deterministic solver.



**Default:** [ ] (Empty, which specifies the active Configset object for modelobj)

## **variantobj**

Object or array of objects of the Variant class.

**Default:** [ ] (Empty, which specifies no variant object)

## **doseobj**

Object or array of objects of the RepeatDose or ScheduleDose class.

**Default:** [ ] (Empty, which specifies no dose object)

## **Output Arguments**

### **equations**

String containing the system of equations that represent a model. This string includes equations for reactions, rules, events, variants, and doses.

## **Examples**

### **View System of Equations for Simple Model**

View system of equations that represent a simple model, containing only reactions.

Import the lotka model, included with SimBiology, into a variable named model1:

```
model1 = sbmlimport('lotka');
```

View all equations that represent the model1 model and its active configset:

```
m1equations = getequations(model1)
```

```
m1equations =
```

```
ODEs:
```

## Model.getequations

---

```
d(y1)/dt = 1/unnamed*(ReactionFlux1 - ReactionFlux2)
d(y2)/dt = 1/unnamed*(ReactionFlux2 - ReactionFlux3)
d(z)/dt = 1/unnamed*(ReactionFlux3)
```

```
Fluxes:
ReactionFlux1 = c1*y1*x
ReactionFlux2 = c2*y1*y2
ReactionFlux3 = c3*y2
```

```
Parameter Values:
c1 = 10
c2 = 0.01
c3 = 10
unnamed = 1
```

```
Initial Conditions:
x = 1
y1 = 900
y2 = 900
z = 0
```

MATLAB displays the ODEs, fluxes, parameter values, and initial conditions for the reactions in `model1`.

### **View System of Equations for Model and Dose**

View system of equations that represent a model, containing only reactions, and a repeated dose.

Import the `lotka` model, included with SimBiology, into a variable named `model1`:

```
model1 = sbmlimport('lotka');
```

Add a repeated dose to the model:

```
doseObj1 = adddose(model1, 'dose1', 'repeat');
```

Set the properties of the dose to administer 3 mg, at a rate of 10 mg/hour, 6 times, at an interval of every 24 hours, to species y1:

```
doseObj1.Amount = 0.003;
doseObj1.AmountUnits = 'gram';
doseObj1.Rate = 0.010;
doseObj1.RateUnits = 'gram/hour';
doseObj1.Repeat = 6;
doseObj1.Interval = 24;
doseObj1.TimeUnits = 'hour';
doseObj1.TargetName = 'y1';
```

View all equations that represent the model1 model, its active configset, and the repeated dose:

```
m1_with_dose_equations = getequations (model1,[],[],doseObj1)
```

```
m1_with_dose_equations =
```

ODEs:

```
d(y1)/dt = 1/unnamed*(ReactionFlux1 - ReactionFlux2) + dose1
d(y2)/dt = 1/unnamed*(ReactionFlux2 - ReactionFlux3)
d(z)/dt = 1/unnamed*(ReactionFlux3)
```

Fluxes:

```
ReactionFlux1 = c1*y1*x
ReactionFlux2 = c2*y1*y2
ReactionFlux3 = c3*y2
```

Parameter Values:

```
c1 = 10
c2 = 0.01
c3 = 10
unnamed = 1
```

Initial Conditions:

```
y1 = 900
y2 = 900
```

# Model.getequations

---

```
z = 0
x = 1
```

```
Doses:
Variable          Type          Units
dose1             repeatdose   gram
```

MATLAB displays the ODEs, fluxes, parameter values, and initial conditions for the reactions and the dose in `model1`.

## See Also

[Model object](#) | [Configset object](#) | [Variant object](#) | [RepeatDose object](#) | [ScheduleDose object](#)

## Purpose

Get indices into ValueInfo and InitialValues properties

## Syntax

```
indices = getIndex(model,name)
indices = getIndex(model,name,type)
```

## Description

`indices = getIndex(model,name)` returns the indices of all ValueInfo objects in a SimBiology.export.Model object that have a QualifiedName or Name property that match the specified name input argument.

- `getIndex` first tries to match the QualifiedName property. If there are matches, then `getIndex` returns their indices.
- If there are no matches based on QualifiedName, then `getIndex` tries to match the Name property. If there are matches, then `getIndex` returns their indices.
- If there are no matches based on QualifiedName or Name, then `getIndex` returns [].

`indices = getIndex(model,name,type)` returns indices for only the ValueInfo objects with a Type property that matches the type input argument.

## Input Arguments

### model

SimBiology.export.Model object.

### name

String containing a name to match against the QualifiedName, then Name, properties of the ValueInfo objects in model.

### type

String containing a name to match against the Type property of the ValueInfo objects in model.

**Default:** All types.

# SimBiology.export.Model.getIndex

---

## Output Arguments

### indices

Vector of indices indicating which `ValueInfo` objects in a `SimBiology.export.Model` object match on the specified name and type.

## Examples

### Index Exported SimBiology Editable Values

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj);
```

Get the index of the editable value with name `y1`.

```
ix = getIndex(em,'y1')
```

```
ix =
```

```
3
```

Display the type of value.

```
em.ValueInfo(ix).Type
```

```
ans =
```

```
species
```

The name `y1` corresponds to an editable species.

## See Also

`SimBiology.export.Model` | `SimBiology.export.ValueInfo` |

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

## Purpose

Get specific parameters in kinetic law object

## Syntax

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

## Arguments

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

## Description

*parameterObj* = `getparameters(kineticlawObj)` returns the parameters used by the kinetic law object *kineticlawObj* to *parameterObj*.

*parameterObj* = `getparameters(kineticlawObj, 'ParameterVariablesValue')` returns the parameter in the ParameterVariableNames property that corresponds to the parameter specified in the ParameterVariables property of *kineticlawObj*, to *parameterObj*. ParameterVariablesValue is the name of the parameter as it appears in the ParameterVariables property of *kineticlawObj*. ParameterVariablesValue can be a cell array of strings.

If you change the name of a parameter, you must configure all applicable elements such as rules that use the parameter, any user-specified ReactionRate, or the kinetic law object property ParameterVariableNames. Use the method `setParameter` to configure ParameterVariableNames.

## Examples

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

## getparameters (kineticlaw)

---

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

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

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

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

- 3 Add two parameter objects.

```
parameterObj1 = addparameter(kineticlawObj, 'Va');  
parameterObj2 = addparameter(kineticlawObj, 'Ka');
```

- 4 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables ( $V_m$  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:

SimBiology Parameter Array

Index:	Name:	Value:	ValueUnits:
1	Va	1	

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

### See Also

addparameter, getspecies, setparameter



## Purpose

Get 3-D sensitivity matrix from SimData array

## Syntax

```
[T, R, Outputs, InputFactors] = getsensmatrix(simDataObj)
[T, R, Outputs, InputFactors] = getsensmatrix(simDataObj,
    OutputNames, InputFactorNames)
```

## Arguments

<i>T</i>	<i>T</i> is an <i>m</i> -by-1 array specifying time points for the sensitivity data in <i>R</i> .
<i>R</i>	<i>R</i> is an <i>m</i> -by- <i>n</i> -by- <i>p</i> array of sensitivity data with times, outputs, and input factors corresponding to its first, second, and third dimensions respectively. <i>R</i> (:, <i>i</i> , <i>j</i> ) is the time course for the sensitivity of state <i>Outputs</i> { <i>i</i> } to the input factor <i>InputFactors</i> { <i>j</i> }.
<i>Outputs</i>	Name of the output factors, where output factors are the names of the states for which you want to calculate sensitivity.
<i>InputFactors</i>	Name of the input factors, where input factors are the names of the states with respect to which you want to calculate sensitivity.

## Description

`[T, R, Outputs, InputFactors] = getsensmatrix(simDataObj)` gets time and sensitivity data from the SimData object (*simDataObj*).

When *simDataObj* contains more than one element, the output arguments are cell arrays in which each cell contains data for the corresponding element of *simDataObj*.

The `getsensmatrix` method can only return sensitivity data that is contained in the SimData object. The sensitivity data that is logged in a SimData object is set at simulation time by the configuration set used during the simulation. This is typically the model's active configuration set. For an explanation of how to set up a sensitivity calculation using the configuration set, see "Sensitivity Analysis". Note in particular that

## getsensmatrix (SimData)

---

the sensitivity data *R* returned by `getsensmatrix` may be normalized, as specified at simulation time.

```
[T, R, Outputs, InputFactors] =  
getsensmatrix(simDataObj, OutputNames, InputFactorNames) gets  
sensitivity data for the outputs specified by OutputNames and the input  
factors specified by InputFactorNames.
```

*OutputNames* and *InputFactorNames* can both be any one of the following:

- Empty array
- Single name
- Cell array of names

Pass an empty array for *OutputNames* or *InputFactorNames* to ask for sensitivity data on all output factors or input factors contained in *simDataObj*, respectively. You can also use qualified names such as '*CompartmentName.SpeciesName*' or '*ReactionName.ParameterName*' to resolve ambiguities.

### Examples

This example shows how to retrieve sensitivity data from a `SimData` object.

**1** Set up the simulation:

- a** Import the radio decay model from SimBiology examples.

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

- b** Retrieve the configuration settings and the sensitivity analysis options from the `modelObj`.

```
configsetObj = getconfigset(modelObj);  
sensitivityObj = get(configsetObj, 'SensitivityAnalysisOptions');
```

- c** Specify the species for which you want sensitivity data in the `Outputs` property. All model species are selected in this example.

Use the `sbioselect` function to retrieve the species objects from the model.

```
allSpeciesObj = sbioselect(modelObj, 'Type', 'species');  
set(sensitivityObj, 'Outputs', allSpeciesObj);
```

- d** Specify species and parameters with respect to which you want to calculate the sensitivities in the `Inputs` property.
- e** Enable `SensitivityAnalysis`.

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

```
ans =
```

```
1
```

- f** Simulate and return the results in a `SimData` object.

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

- 2** Extract and plot sensitivity data from the `SimData` object.

- a** Use `getsensmatrix` to retrieve sensitivity data.

```
[t R outs ifacs] = getsensmatrix(simDataObj);
```

- b** Plot sensitivity values.

```
plot(t, R(:, :, 2));  
legend(outs);  
title(['Sensitivities of species relative to ' ifacs{2}]);
```

### See Also

`display`, `get`, `getdata`, `resample`, `selectbyname`

MATLAB function struct

# getspecies (kineticlaw)

---

**Purpose** Get specific species in kinetic law object

**Syntax**

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

## Arguments

<i>kineticlawObj</i>	Retrieve species used by the kinetic law object.
<i>SpeciesVariablesValue</i>	Retrieve species used by the kinetic law object corresponding to the specified species in the <code>SpeciesVariables</code> property of the kinetic law object.

## Description

`speciesObj = getspecies(kineticlawObj)` returns the species used by the kinetic law object `kineticlawObj` to `speciesObj`.

`speciesObj = getspecies(kineticlawObj, 'SpeciesVariablesValue')` returns the species in the `SpeciesVariableNames` property to `speciesObj`.

`SpeciesVariablesValue` is the name of the species as it appears in the `SpeciesVariables` property of `kineticlawObj`. `SpeciesVariablesValue` can be a cell array of strings.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the name of a species, the reaction updates to use the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object `SpeciesVariableNames`. Use the method `setspecies` to configure `SpeciesVariableNames`.

## Examples

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

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

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

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

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

reactionObj KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has one species variable (S) that should to be set. To set this variable:

```
setspecies(kineticlawObj, 'S', 'a');
```

- 4 Retrieve the species variable using getspecies.

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

MATLAB returns:

SimBiology Species Array

Index:	Compartment:	Name:	InitialAmount:	InitialAmountUnits:
1	unnamed	a	0	

### See Also

[addspecies](#), [getparameters](#), [setparameter](#), [setspecies](#)

# getstoichmatrix (model)

---

**Purpose** Get stoichiometry matrix from model object

**Syntax**

```
M = getstoichmatrix(modelObj)
[M,objSpecies] = getstoichmatrix(modelObj)
[M,objSpecies,objReactions] = getstoichmatrix(modelObj)
```

## Arguments

<i>M</i>	Adjacency matrix for <i>modelObj</i> .
<i>modelObj</i>	Specify the model object <i>modelObj</i> .
<i>objSpecies</i>	Return the list of <i>modelObj</i> species by Name property of the species. If the species are in multiple compartments, species names are qualified with the compartment name in the form compartmentName.speciesName. For example, nucleus.DNA, cytoplasm.mRNA.
<i>objReactions</i>	Return the list of <i>modelObj</i> reactions by the Name property of reactions.

## Description

getstoichmatrix returns a stoichiometry matrix for a model object.

*M* = getstoichmatrix(*modelObj*) returns a stoichiometry matrix for a SimBiology model object (*modelObj*) to *M*.

A stoichiometry matrix is defined by listing all reactions contained by *modelObj* column-wise and all species contained by *modelObj* row-wise in a matrix. The species of the reaction are represented in the matrix with the stoichiometric value at the location of [row of species, column of reaction]. Reactants have negative values. Products have positive values. All other locations in the matrix are 0.

For example, if *modelObj* is a model object with two reactions with names R1 and R2 and Reaction values of 2 A + B -> 3 C and B + 3 D -> 4 A, the stoichiometry matrix would be defined as:

	R1	R2
A	-2	4
B	-1	-1
C	3	0
D	0	-3

`[M,objSpecies] = getstoichmatrix(modelObj)` returns the stoichiometry matrix to *M* and the species to *objSpecies*. *objSpecies* is defined by listing all Name property values of species contained by *Obj*. In the above example, *objSpecies* would be {'A', 'B', 'C', 'D'}.

`[M,objSpecies,objReactions] = getstoichmatrix(modelObj)` returns the stoichiometry matrix to *M* and the reactions to *objReactions*. *objReactions* is defined by listing all Name property values of reactions contained by *modelObj*. In the above example, *objReactions* would be {'R1', 'R2'}.

## Examples

**1** Read in *m1*, a model object, using `sbmlimport`:

```
m1 = sbmlimport('lotka.xml');
```

**2** Get the stoichiometry matrix for the *m1*:

```
[M,objSpecies,objReactions] = getstoichmatrix(m1)
```

## See Also

`getadjacencymatrix`, “Determining the Stoichiometry Matrix for a Model”

# getvariant (model)

---

**Purpose** Get variant from model

**Syntax**  
`variantObj = getvariant(modelObj)`  
`variantObj = getvariant(modelObj, 'NameValue')`

## Arguments

<code>variantObj</code>	Variant object returned by the <code>getvariant</code> method.
<code>modelObj</code>	Model object from which to get the variant.
<code>'NameValue'</code>	Name of the variant to get from the model object <code>modelObj</code> .

## Description

`variantObj = getvariant(modelObj)` returns SimBiology variant objects contained by the SimBiology model object `modelObj` to `variantObj`.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see [Variant object](#).

`variantObj = getvariant(modelObj, 'NameValue')` returns the SimBiology variant object with the name `NameValue`, contained by the SimBiology model object, `modelObj`.

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

---

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

---

To copy a variant object to another model, use `copyobj`. To remove a variant object from a SimBiology model, use the `delete` method.



## Examples

- 1 Create a model containing several variants.

```
modelObj = sbiomodel('mymodel');  
variantObj1 = addvariant(modelObj, 'v1');  
variantObj2 = addvariant(modelObj, 'v2');
```

- 2 Get all variants in the model.

```
vObjs = getvariant(modelObj)
```

SimBiology Variant Array

Index:	Name:	Active:
1	v1	false
2	v2	false

- 3 Get the variant object named 'v2' from the model.

```
vObjv2 = getvariant(modelObj, 'v2');
```

## See Also

addvariant, removevariant

# SimBiology.export.Model.isAccelerated

---

**Purpose** Determine whether an exported SimBiology model is accelerated

**Syntax**  
`tf = isAccelerated(model)`  
`tf = isAccelerated(model, computerType)`

**Description**  
`tf = isAccelerated(model)` returns true if `model` is accelerated for the current type of computer, and false otherwise.  
`tf = isAccelerated(model, computerType)` returns true if `model` is accelerated for the specified computer type.

**Input Arguments**

**model**  
SimBiology.export.Model object.

**computerType**  
String specifying a computer type. You can specify any valid archstr supported by the function `computer`.

**Output Arguments**

**tf**  
Logical value true if `model` is accelerated for the current computer type, or computer type specified by `computerType`.  
Logical value false if the exported model is not accelerated for the specified computer type.

## Examples Accelerate Exported SimBiology Model

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj)
```

```
em =
```

```
SimBiology.export.Model
```

```
    Name: lotka
```

```
ExportTime: 28-Jun-2012 08:13:47  
ExportNotes:
```

Accelerate the exported model.

```
accelerate(em);  
em.isAccelerated
```

```
ans =
```

```
1
```

The logical value 1 indicates that the exported model is accelerated.

## See Also

```
SimBiology.export.Model | SimBiology.export.Model.accelerate  
| computer
```

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# KineticLaw object

---

**Purpose** Kinetic law information for reaction

**Description** The kinetic law object holds information about the abstract kinetic law applied to a reaction and provides a template for the reaction rate. In the model, the SimBiology software uses the information you provide in a fully defined kinetic law object to determine the `ReactionRate` property in the reaction object.

When you first create a kinetic law object, you must specify the name of the abstract kinetic law to use. The SimBiology software fills in the `KineticLawName` property and the `Expression` property in the kinetic law object with the name of the abstract kinetic law you specified and the mathematical expression respectively. The software also fills in the `ParameterVariables` property and the `SpeciesVariables` property of the kinetic law object with the values found in the corresponding properties of the abstract kinetic law object.

To obtain the reaction rate, you must fully define the kinetic law object:

- 1** In the `ParameterVariableNames` property, specify the parameters from the model that you want to substitute in the expression (`Expression` property).
- 2** In the `SpeciesVariableNames` property, specify the species from the model that you want to substitute in the expression.

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

The software then shows the substituted expression as the reaction rate in the `ReactionRate` property of the reaction object. If the kinetic law object is not fully defined, the `ReactionRate` property remains ' ' (empty).

For links to kinetic law object property reference pages, see “Property Summary” on page 4-133.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

For an explanation of how relevant properties relate to one another, see “Command Line” on page 4-129.

The following sections use a kinetic law example to show how you can fully define your kinetic law object to obtain the reaction rate in the SimBiology desktop and at the command line.

The Henri-Michaelis-Menten kinetic law is expressed as follows:

$$V_m * S / (K_m + S)$$

In the SimBiology software Henri-Michaelis-Menten is a built-in abstract kinetic law, where  $V_m$  and  $K_m$  are defined in the `ParameterVariables` property of the abstract kinetic law object, and  $S$  is defined in the `SpeciesVariables` property of the abstract kinetic law object.

## SimBiology Desktop

To fully define a kinetic law in the SimBiology desktop, define the names of the species variables and parameter variables that participate in the reaction rate.

## 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 ( $V_m$  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  $V_m = V_a$ 
%  $V_m$  is listed first in 'ParameterVariables',
% therefore list  $V_a$  first in 'ParameterVariableNames'.

set(kineticlawObj, 'ParameterVariableNames', {'Va' 'Ka'});
set(kineticlawObj, 'SpeciesVariableNames', {'A'});
```

The rate equation is assigned in the reaction object as follows:

$$V_a * A / (K_a + A)$$

For a detailed procedure, see “Examples” on page 4-134.

The following table summarizes the relationships between the properties in the abstract kinetic law object and the kinetic law object in the context of the above example.

Property	Property Purpose	Abstract Kinetic Law Object	Kinetic Law Object
Name (abstract kinetic law object) KineticLawName (kinetic law object)	Name of abstract kinetic law applied to a reaction. For example:  Henri-Michaelis-Menten	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only
Expression	Mathematical expression used to determine the reaction rate equation. For example:  $V_m * S / (K_m + S)$	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only; depends on abstract kinetic law applied to reaction.
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



display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getparameters (kineticlaw)	Get specific parameters in kinetic law object
getspecies (kineticlaw)	Get specific species in kinetic law object
set (any object)	Set object properties
setParameter (kineticlaw)	Specify specific parameters in kinetic law object
setspecies (kineticlaw)	Specify species in kinetic law object

## Property Summary

Expression (AbstractKineticLaw, KineticLaw)	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 kinetic law definition
Parent	Indicate parent object

# KineticLaw object

---

SpeciesVariableNames	Cell array of species in reaction rate equation
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Examples

This example shows how to define the reaction rate for a reaction.

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

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'A -> B');
```

- 2 Define a kinetic law for the reaction object.

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

- 3 Query the parameters and species variables defined in the kinetic law.

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

```
ans =
```

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

```
get(kineticlawObj, 'SpeciesVariables')
```

```
ans =
```

```
    'S'
```

- 4 Define  $V_a$  and  $K_a$  as `ParameterVariableNames`, which correspond to the `ParameterVariables`  $V_m$  and  $K_m$ . To set these variables, first create the parameter variables as parameter objects (`parameterObj1`, `parameterObj2`) with the names  $V_a$  and  $K_a$ , and then add them to `kineticlawObj`. The species object with Name  $A$  is created when `reactionObj` is created and need not be redefined.

```
parameterObj1 = addparameter(kineticlawObj, 'Va');  
parameterObj2 = addparameter(kineticlawObj, 'Ka');
```

- 5 Set the variable names for the kinetic law object.

```
set(kineticlawObj, 'ParameterVariableNames', {'Va' 'Ka'});  
set(kineticlawObj, 'SpeciesVariableNames', {'A'});
```

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

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
Va*A/(Ka+A)
```

### See Also

`AbstractKineticLaw` object, `Configset` object, `Model` object, `Parameter` object, `Reaction` object, `Root` object, `Rule` object, `Species` object

`SimBiology` property `Expression`(`AbstractKineticLaw`, `KineticLaw`)

# Model object

---

**Purpose** Model and component information

**Description** The SimBiology model object represents a *model*, which is a collection of interrelated reactions and rules that transform, transport, and bind species. The model includes model components such as compartments, reactions, parameters, rules, and events. Each of the components is represented as a property of the model object. A model object also has a default configuration set object to define simulation settings. You can also add more configuration set objects to a model object.

See “Property Summary” on page 4-138 for links to model property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

You can retrieve SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object. The root object contains a list of model objects that are accessible from the MATLAB command line and from the SimBiology desktop. Because both the command line and the desktop point to the same model object in the `Root` object, any changes you make to the model at the command line are reflected in the desktop, and vice versa.

## Constructor Summary

<code>sbiomodel</code>	Construct model object
------------------------	------------------------

## Method Summary

<code>addcompartment (model, compartment)</code>	Create compartment object
--------------------------------------------------	---------------------------

<code>addconfigset (model)</code>	Create configuration set object and add to model object
-----------------------------------	---------------------------------------------------------

<code>adddose (model)</code>	Add dose object to model
------------------------------	--------------------------

addevent (model)	Add event object to model object
addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
addreaction (model)	Create reaction object and add to model object
addrule (model)	Create rule object and add to model object
addspecies (model, compartment)	Create species object and add to compartment object within model object
addvariant (model)	Add variant to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
export (model)	Export SimBiology model
get (any object)	Get object properties
getadjacencymatrix (model)	Get adjacency matrix from model object
getconfigset (model)	Get configuration set object from model object
getdose (model)	Return SimBiology dose object
getequations	Return system equations for model object
getstoichmatrix (model)	Get stoichiometry matrix from model object
getvariant (model)	Get variant from model

# Model object

---

removeconfigset (model)	Remove configuration set from model
removedose (model)	Add dose object to model
removevariant (model)	Remove variant from model
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties
setactiveconfigset (model)	Set active configuration set for model object
verify (model, variant)	Validate and verify SimBiology model

## Property Summary

Compartments	Array of compartments in model or compartment
Events	Contain all event objects
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parameters	Array of parameter objects
Parent	Indicate parent object
Reactions	Array of reaction objects
Rules	Array of rules in model object
Tag	Specify label for SimBiology object
Type	Display 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

# Parameter object

---

**Purpose** Parameter and scope information

**Description** The parameter object represents a *parameter*, which is a quantity that can change or can be constant. SimBiology parameters are generally used to define rate constants. You can add parameter objects to a model object or a kinetic law object. The scope of a parameter depends on where you add the parameter object: If you add the parameter object to a model object, the parameter is available to all reactions in the model and the `Parent` property of the parameter object is `SimBiology.Model`. If you add the parameter object to a kinetic law object, the parameter is available only to the reaction for which you are using the kinetic law object and the `Parent` property of the parameter object is `SimBiology.KineticLaw`.

See “Property Summary” on page 4-141 for links to parameter object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

**Constructor Summary**

<code>addparameter (model, kineticlaw)</code>	Create parameter object and add to model or 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
<code>get (any object)</code>	Get object properties



rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

## Property Summary

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 SimBiology object type
UserData	Specify data to associate with object
Value	Assign value to parameter object
ValueUnits	Parameter value units

## See Also

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

# PKCompartment object

---

**Purpose** Used by PKModelDesign to create SimBiology model

**Description** The PKCompartment object is used by the PKModelDesign object to construct a SimBiology model for pharmacokinetic modeling. PKCompartment holds the following information:

- Name of the compartment
- Dosing type
- Elimination type
- Whether the drug concentration in this compartment is reported

The PKCompartment class is a subclass of the hgsetget class which is a subclass of the handle class. For more information on the inherited methods, see hgsetget, and handle.

**Construction**

addCompartment (PKModelDesign)	Add compartment to PKModelDesign object
-----------------------------------	--------------------------------------------

**Method Summary**

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

**Property Summary**

DosingType	Drug dosing type in compartment
EliminationType	Drug elimination type from compartment
HasLag	Lag associated with dose targeting compartment
HasResponseVariable	Compartment drug concentration reported
Name	Specify name of object

**See Also**

“Creating Pharmacokinetic Models” in the SimBiology User’s Guide,  
PKModelDesign object

# PKData object

---

**Purpose** Define roles of data set columns

**Description** The properties of the PKData object specify what each column in the data represents. The PKData object specifies which columns in the data set represent the following:

- The grouping variable
- The independent and dependent variables
- The dose
- The rate (only if infusion is the dosing type)
- The covariates

This information is used by the fitting functions, `sbionlmefit` and `sbionlinfit`.

To create the PKData object specify:

```
pkDataObject = PKData(data);
```

Where `data` is the imported data set.

The PKData class is a subclass of the `hgsetget` class, which is a subclass of the `handle` class. For more information on the inherited methods, see `hgsetget` and `handle`.

**Construction** PKData Create PKData object

**Method Summary**

<code>get</code> (any object)	Get object properties
<code>getCovariateData</code> (pkdata)	Create design matrix needed for fit
<code>set</code> (any object)	Set object properties

## Property Summary

CovariateLabels	Identify covariate columns in data set
DataSet	Dataset object containing imported data
DependentVarLabel	Identify dependent variable column in data set
DependentVarUnits	Response units in PKData object
DoseLabel	Dose column in data set
DoseUnits	Dose units in PKData object
GroupID	Integer identifying each group in data set
GroupLabel	Identify group column in data set
GroupNames	Unique values from GroupLabel in data set
IndependentVarLabel	Identify independent variable column in data set
IndependentVarUnits	Time units in PKData object
RateLabel	Rate of infusion column in data set
RateUnits	Units for dose rate

## See Also

“Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, PKModelDesign object

# PKModelDesign object

---

**Purpose** Helper object to construct pharmacokinetic model

**Description** Use the PKModelDesign object to construct a SimBiology model for PK modeling. The PKModelDesign object lets you specify the number of compartments, the type of dosing, and method of elimination which you then use to construct the SimBiology model object with the necessary compartments, species, reactions, rules, and events.

```
pkm = PKModelDesign;
```

Use the `addCompartment` method to add a compartment with a specified dosing and elimination. `addCompartment` adds each subsequent compartment and connects it to the previous compartment using a reversible reaction. This reaction models the flux between compartments in a PK model.

The `construct` method uses the PKModelDesign object to create a SimBiology model object.

The PKModelDesign class is a subclass of the `hgsetget` class, which is a subclass of the `handle` class. For more information on the inherited methods see `hgsetget` and `handle`.

**Construction** PKModelDesign Create PKModelDesign object

<b>Method Summary</b>	<code>addCompartment</code> (PKModelDesign)	Add compartment to PKModelDesign object
	<code>construct</code> (PKModelDesign)	Construct SimBiology model from PKModelDesign object
	<code>get</code> (any object)	Get object properties
	<code>set</code> (any object)	Set object properties

## Property Summary

PKCompartments

Hold compartments in PK model

## See Also

“Creating Pharmacokinetic Models” in the SimBiology User’s Guide,  
PKCompartment object

# PKModelMap object

---

<b>Purpose</b>	Define SimBiology model components' roles	
<b>Description</b>	<p>The PKModelMap object holds information about the dosing type, and defines which components of a SimBiology model represent the observed response, the dose, and the estimated parameters.</p> <p>The PKModelMap class is a subclass of the hgsetget class which is a subclass of the handle class. For more information on the inherited methods see, hgsetget, and handle.</p>	
<b>Construction</b>	PKModelMap	Create PKModelMap object
<b>Method Summary</b>	get (any object) set (any object)	Get object properties Set object properties
<b>Property Summary</b>	Dosed DosingType Estimated LagParameter  Observed ZeroOrderDurationParameter	Dosed object name Drug dosing type in compartment Names of parameters to estimate Parameter specifying time lag for doses  Measured response object name Zero-order dose absorption duration
<b>See Also</b>	“Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, PKModelDesign 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 4-150 for links to reaction object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

<code>addreaction (model)</code>	Create reaction object and add to model 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 reactant to reaction object
<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>delete (any object)</code>	Delete SimBiology object

# Reaction object

---

display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmproduct (reaction)	Remove species object from reaction object products
rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

## Property Summary

Active	Indicate object in use during simulation
KineticLaw	Show kinetic law used for ReactionRate
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Products	Array of reaction products
Reactants	Array of reaction reactants
Reaction	Reaction object reaction
ReactionRate	Reaction rate equation in reaction object
Reversible	Specify whether reaction is reversible or irreversible
Stoichiometry	Species coefficients in reaction
Tag	Specify label for SimBiology object

Type	Display SimBiology object type
UserData	Specify data to associate with object

## See Also

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

# removeconfigset (model)

---

**Purpose** Remove configuration set from model

**Syntax**  
`removeconfigset(modelObj, 'NameValue')`  
`removeconfigset(modelObj, configsetObj)`

## Arguments

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

## Description

`removeconfigset(modelObj, 'NameValue')` removes the configset object with the name *NameValue* from the SimBiology model object *modelObj*. A configuration set object stores simulation-specific information. A SimBiology model can contain multiple configuration sets with one being active at any given time. The active configuration set contains the settings that are used during the simulation. *modelObj* always contains at least one configuration set object with name configured to 'default'. You cannot remove the default configuration set from *modelObj*. If the active configuration set is removed from *modelObj*, then the default configuration set will be made active.

`removeconfigset(modelObj, configsetObj)` removes the configuration set object, *configsetObj*, from the SimBiology model, *modelObj*. The configuration set is not deleted; if you want to delete *configsetObj*, use the `delete` method.

If however, there is no MATLAB variable holding the configset, `removeconfigset(modelObj, 'NameValue')` removes the configset from the model and deletes it.

## Examples

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

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

```
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

# removedose (model)

---

**Purpose** Add dose object to model

**Syntax**  
`doseObj2 = removedose(modelObj, 'DoseName')`  
`doseObj2 = removedose(modelObj, doseObj)`

## Arguments

<i>modelObj</i>	Model object from which you remove a dose object.
<i>DoseName</i>	Name of the dose object to remove from a model object. <i>DoseName</i> is the value of the dose object property Name.
<i>doseObj</i>	Dose object to remove from a model object.

## Outputs

<i>doseObj2</i>	ScheduleDose or RepeatDose object.
-----------------	------------------------------------

## Description

`doseObj2 = removedose(modelObj, 'DoseName')` removes a SimBiology ScheduleDose or RepeatDose object with the name *DoseName* from a model object (*modelObj*). returns the dose object (*doseObj*), and assigns [] to the dose object property Parent.

You can add a removed dose object back to a model object using the method `adddose`.

`doseObj2 = removedose(modelObj, doseObj)` removes a SimBiology ScheduleDose or RepeatDose object *doseObj*.

## Examples

Remove a dose object from a model object.

- 1 Create model and dose objects, and then add dose to model.

```
modelObj = sbiomodel('mymodel');  
doseObj = adddose(modelObj, 'dose1');
```

- 2 Remove dose object from model object.

```
removedose(mymodel, 'dose1');
```

Get all dose objects from a model object, and then remove the second dose object.

```
AllDoseObjects = getdose(mymodel);  
removedose(mymodel, AllDoseObjects(2));
```

### See Also

Model object methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

`ScheduleDose` object and `RepeatDose` object methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

# removevariant (model)

---

**Purpose** Remove variant from model

**Syntax**  
`variantObj = removevariant(modelObj, 'NameValue')`  
`variantObj = removevariant(modelObj, variantObj)`

## Arguments

*modelObj* Specify the model object from which you want to remove the variant.

*variantObj* Specify the variant object to return from the model object.

## Description

`variantObj = removevariant(modelObj, 'NameValue')` removes a SimBiology variant object with the name *NameValue* from the model object *modelObj* and returns the variant object to *variantObj*. The variant object `Parent` property is assigned `[]` (empty).

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant` object.

`variantObj = removevariant(modelObj, variantObj)` removes a SimBiology variant object (*variantObj*) and returns the variant object *variantObj*.

To view the variants stored on a model object, use the `getvariant` method. To copy a variant object to another model, use `copyobj`. To add a variant object to a SimBiology model, use the `addvariant` method.

## Examples

**1** Create a model containing several variants.

```
modelObj = sbiomodel('mymodel');  
variantObj1 = addvariant(modelObj, 'v1');  
variantObj2 = addvariant(modelObj, 'v2');  
variantObj3 = addvariant(modelObj, 'v3');
```

**2** Remove a variant object using its name.



```
removevariant(modelObj, 'v1');
```

**3** Remove a variant object using its index number.

**a** Get the index number of the variant in the model.

```
vObjs = getvariant(modelObj)
```

```
SimBiology Variant Array
```

Index:	Name:	Active:
1	v2	false
2	v3	false

**b** Remove the variant object.

```
removevariant(modelObj, vObjs(2));
```

### See Also

addvariant, getvariant

# rename (compartment, parameter, species)

---

**Purpose** Rename object and update expressions

**Syntax** `rename(Obj, 'NewNameValue')`

## Arguments

*Obj* Compartment, parameter, or species object.  
'NewNameValue' Specify the new name.

## Description

`rename(Obj, 'NewNameValue')`, changes the Name property of the object, *Obj* to *NewNameValue* and updates any expressions in the model (such as Rule or ReactionRate) to use the new name.

If the new name is already being used by another model component, the new name will be qualified to ensure that it is unique. For example if you change a species named A to K, and a parameter with the name K exists, the species will be qualified as *CompartmentName.K* to indicate that the reference is to the species. If you are referring to an object by its qualified name, for example *CompartmentName.A* and you change the species name, the reference will contain the qualified name in its updated form, for example, *CompartmentName.K*

When you want to change the name of a compartment, parameter, or species object, use this method instead of `set`. The `set` method only changes the Name property of the object, except for species objects where the species object's Name property and any reaction strings which refer to species are updated to use the new name.

## Examples

**1** Create a model object that contains a species A in a rule.

```
m = sbiomodel('cell');  
s = addspecies(m, 'A');  
r = addrule(m, 'A = 4');
```

**2** Rename the species to Y

```
rename(s, 'Y');
```

## rename (compartment, parameter, species)

---

**3** See that the rule expression is now updated.

r

SimBiology Rule Array

Index:	RuleType:	Rule:
1	initialAssignment	Y = 4

### See Also

set

# reorder (model, compartment)

---

**Purpose** Reorder component lists

**Syntax** `modelObj = reorder(Obj, NewOrder)`

## Arguments

<i>Obj</i>	Model object or compartment. Enter a variable name.
<i>NewOrder</i>	Object vector in the new order. If <i>Obj</i> is a model object, <i>NewOrder</i> can be an array of compartments, events, parameters, reactions or rules objects. If <i>Obj</i> is a compartment object, <i>NewOrder</i> must be an array of species objects.

## Description

`modelObj = reorder(Obj, NewOrder)` reorders the component vector *NewOrder*, to be in the order specified.

You can use this method to reorder any of the component vectors, such as events, parameters, rules, and species. The vector of components, when reordered, must contain the same objects as the original list of objects but they can be in a different order.

## Examples

**1** Import a model.

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

**2** Display the order of the reactions in the model.

```
get(modelObj.Reactions);
```

```
SimBiology Reaction Array
```

```
Index:   Reaction:
1        x + y1 -> 2 y1 + x
2        y1 + y2 -> 2 y2
3        y2 -> z
```

## reorder (model, compartment)

---

- 3 Reverse the order of the reactions in the model.

```
reorder(modelObj, modelObj.Reactions([3 2 1]))
```

# RepeatDose object

---

**Purpose** Define drug dosing protocol

**Description** A RepeatDose object defines a series of doses to the amount of a species during a simulation. The TargetName property of a dose object defines the species that receives the dose.

Each dose is the same amount, as defined by the Amount property, and given at equally spaced times, as defined by the Interval property. The RepeatCount property defines the number of injections in the series, excluding the initial injection. The Rate property defines how fast each dose is given.

To use a dose object in a simulation you must add the dose object to a model object and set the Active property of the dose object to true. Set the Active property to true if you always want the dose to be applied before simulating the model.

When there are multiple active RepeatDose objects on a model and if there are duplicate specifications for a property value, the last occurrence for the property value in the array of dose, is used during simulation. You can find out which dose is applied last by looking at the indices of the variant objects stored on the model.

See “Property Summary” on page 4-213 for links to species property reference pages. Properties define the characteristics of an object. Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

sbiodose

Construct dose object

## Method Summary

Methods for RepeatDose objects

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

## Property Summary

Properties for RepeatDose objects

Active	Indicate object in use during simulation
Amount	Amount of dose
AmountUnits	Dose amount units
DurationParameterName	arameter length of time
Interval	Time between doses
Name	Specify name of object
Notes	HTML text describing SimBiology object
ParameterName	arameter
Parent	Indicate parent object
Rate	of dose
RateUnits	Units for dose rate
RepeatCount	Dose repetitions
StartTime	Start time for initial dose time
Tag	Specify label for SimBiology object
TargetName	Species receiving dose

# RepeatDose object

---

TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type
UserData	Specify data to associate with object

## See Also

Model object, ScheduleDose object, sbiodose, sbiosimulate



**Purpose** Resample SimData object array onto new time vector

**Syntax**

```
newSimDataObj = resample(simDataObj)  
newSimDataObj = resample(simDataObj, timevector)  
newSimDataObj = resample(simDataObj, timevector, method)
```

## Arguments

*newSimDataObj* Resampled SimData object array.

*simDataObj* SimData object array that you want to resample.

*timevector* Real numeric array of time points onto which you want to resample the data.

*method* Method to use during resampling. Can be one of the following:

- 'interp1q' — Uses the MATLAB function `interp1q`.
- — To use the MATLAB function `interp1`, specify one of the following methods:
  - 'nearest'
  - 'linear'
  - 'spline'
  - 'pchip'
  - 'cubic'
  - 'v5cubic'
- 'zoh' — specifies zero-order hold.

**Description** `newSimDataObj = resample(simDataObj)` resamples the simulation data contained in every element of the SimData object array `simDataObj` onto a common time vector, producing a new SimData array `newSimDataObj`. By default, the common time vector is taken from the element of `simDataObj` with the earliest stopping time.

# resample (SimData)

---

`newSimDataObj = resample(simDataObj, timevector)` resamples the SimData array `simDataObj` onto the time vector `timevector`. `timevector` must either be a real numeric array or the empty array `[]`. If you use an empty array, `resample` uses the default time vector as described above.

`newSimDataObj = resample(simDataObj, timevector, method)` uses the interpolation method specified in `method`.

If the specified `timevector` includes time points outside the time interval encompassed by one or more SimData objects in `simDataObj`, the resampling will involve extrapolation and you will see a warning. See the help for the MATLAB function corresponding to the interpolation method in use for information on how the function performs the extrapolation.

## Examples

### Simulating and Resampling Data

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

```
sbioloadproject('radiodecay');  
simDataObj = sbiosimulate(m1);
```

- 2 Resample the data.

```
newSimDataObj = resample(simDataObj, [1:5], 'linear');
```

### Resampling Data for Ensemble Runs

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

```
sbioloadproject('radiodecay');
```

- 2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj = getconfigset(m1);
```

```
set(csObj, 'SolverType', 'ssa');  
simDataObj = sbioensemblerun(m1, 10);
```

**3** Interpolate the time steps.

```
newSimDataObj = resample(simDataObj, [1:10], 'linear');
```

**4** View the time steps in the SimData object arrays.

```
newSimDataObj(1).Time  
simDataObj(1).Time
```

### See Also

sbioensemblerun, sbioensemblestats, sbiosimulate, SimData  
object

MATLAB functions interp1, interp1q

# reset (root)

---

**Purpose** Delete all model objects from root object

**Syntax** `reset(sbioroot)`

**Description** `reset(sbioroot)` deletes all SimBiology model objects contained by the root object. This call is equivalent to `sbioreset`.

The root object contains a list of model objects, available units, unit prefixes, and kinetic laws.

To add a kinetic law to the user-defined library, use the `sbioaddtolibrary` function. To add a unit to the user-defined library, use `sbiounit` followed by `sbioaddtolibrary`. To add a unit prefix to the user-defined library, use `sbiounitprefix` followed by `sbioaddtolibrary`.

**Examples** **1** Query `sbioroot`, which has two model objects.

```
sbioroot
```

```
SimBiology Root Contains:
```

```
Models: 2
Builtin Abstract Kinetic Laws: 3
User Abstract Kinetic Laws: 1
Builtin Units: 54
User Units: 0
Builtin Unit Prefixes: 13
User Unit Prefixes: 0
```

**2** Call `reset`.

```
sbioroot
```

```
SimBiology Root Contains:
```

```
Models: 0
Builtin Abstract Kinetic Laws: 3
```

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

### See Also

`sbioaddtolibrary`, `sbiohelp`, `sbioreset`, `sbiroot`, `sbiunit`,  
`sbiunitprefix`

# rmcontent (variant)

---

**Purpose** Remove contents from variant object

**Syntax** `rmcontent(variantObj, contents)`  
`rmcontent(variantObj, idx)`

## Arguments

*variantObj* Specify the variant object from which you want to remove data. The Content property is modified to remove the new data.

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

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

*idx* Specify the ContentIndex or indices of the data to be removed. To display the ContentIndex, enter the object name and press **Enter**.

## Description

`rmcontent(variantObj, contents)` removes the data stored in the variable `contents` from the variant object (`variantObj`).

`rmcontent(variantObj, idx)` removes the data specified by the indices `idx` (also called ContentIndex) from the Content property of the variant object.

## Examples

- 1 Create a model containing three species in one compartment.

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

- 2 Add a variant object that varies the species' InitialAmount property.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {'species','A', 'InitialAmount', 5}, ...
{'species', 'B', 'InitialAmount', 10}, ...
{'species', 'C', 'InitialAmount', 15});% Display the variant
variantObj
```

SimBiology Variant - v1 (inactive)

ContentIndex:	Type:	Name:	Property:	Value:
1	species	A	InitialAmount	5
2	species	B	InitialAmount	10
3	species	C	InitialAmount	15

- 3 Use the ContentIndex number to remove a species from the Content property of the variant object.

```
rmcontent(variantObj, 2);
variantObj
```

SimBiology Variant - v1 (inactive)

## rmcontent (variant)

---

ContentIndex:	Type:	Name:	Property:	Value:
1	species	A	InitialAmount	5
2	species	C	InitialAmount	15

- 4** (Alternatively) Remove a species from the contents of the variant object using detailed reference to the species.

```
rmcontent(variantObj, {'species','A', 'InitialAmount', 5});  
% Display variant object  
variantObj  
SimBiology Variant - v1 (inactive)
```

ContentIndex:	Type:	Name:	Property:	Value:
1	species	C	InitialAmount	15

### See Also

addvariant, rmcontent, sbiovariant



**Purpose** Remove species object from reaction object products

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

## Arguments

<i>reactionObj</i>	Reaction object.
<i>SpeciesName</i>	Name for a model object. Enter a species name or a 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 is returned.

## Examples

### Example 1

This example shows how to remove a product that was previously added to a reaction. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP -> creatine + ATP + Pi');  
rmproduct(reactionObj, 'Pi')
```

# rmproduct (reaction)

---

SimBiology Reaction Array

Index: Reaction:

1 Phosphocreatine + ADP -> creatine + ATP

## Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'A -> B + C');  
reactionObj.Reaction
```

```
ans =  
A -> B + C
```

```
rmproduct(reactionObj, modelObj.Species(2));  
reactionObj.Reaction
```

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

## See Also

rmreactant

**Purpose** Remove species object from reaction object reactants

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

## Arguments

<i>reactionObj</i>	Reaction object.
<i>SpeciesName</i>	Name for a species object. Enter a species name or a 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

This example shows how to remove a reactant that was added to a reaction by mistake. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP + Pi -> creatine + ATP');
```

## rmreactant (reaction)

---

```
rmreactant(reactionObj, 'Pi')
```

SimBiology Reaction Array

```
Index:    Reaction:
  1      Phosphocreatine + ADP -> creatine + ATP
```

### Example 2

Remove a species object using a model index to a species object.

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

```
reactionObj.Reaction
ans =
    A + B -> C
```

```
rmreactant(reactionObj, modelObj.Species(1));
reactionObj.Reaction
```

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

### See Also

`delete`, `rmproduct`

## Purpose

Hold models, unit libraries, and abstract kinetic law libraries

## Description

The SimBiology root object contains a list of the SimBiology model objects and SimBiology libraries. The components that the libraries contain are: all available units, unit prefixes, and available abstract kinetic law objects. There are two types of libraries: one contains components that are built in (`BuiltinLibrary`), and the other contains components that are user defined (`UserdefinedLibrary`).

You can retrieve SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object.

See “Property Summary” on page 4-178 for links to root object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

## Constructor Summary

<code>sbioroot</code>	Return SimBiology root object
-----------------------	-------------------------------

## Method Summary

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

# Root object

---

## Property Summary

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

## See Also

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

<b>Purpose</b>	Hold rule for species and parameters	
<b>Description</b>	<p>The SimBiology rule object represents a <i>rule</i>, which is a mathematical expression that modifies a species amount or a parameter value. For a description of the types of SimBiology rules, see <code>RuleType</code>.</p> <p>See “Property Summary” on page 4-179 for links to rule property reference pages.</p> <p>Properties define the characteristics of an object. Use the <code>get</code> and <code>set</code> commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.</p>	
<b>Constructor Summary</b>	<code>addrule (model)</code>	Create rule object and add to model object
<b>Method Summary</b>	<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>get (any object)</code>	Get object properties
	<code>set (any object)</code>	Set object properties
<b>Property Summary</b>	<code>Active</code>	Indicate object in use during simulation
	<code>Name</code>	Specify name of object
	<code>Notes</code>	HTML text describing SimBiology object

# Rule 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 SimBiology object type
UserData	Specify data to associate with object

## See Also

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



## Purpose

Define drug dosing protocol

## Description

A `ScheduleDose` object defines a series of doses to the amount of a species during a simulation. The `TargetName` property defines the species that receives the dose.

Each dose can have a different amount, as defined by an amount array in the `Amount` property. Each dose can be given at specified times, as defined by a time array in the `Time` property. A rate array in the `Rate` property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

To use a dose object in a simulation, you must add the dose object to a model object and set the `Active` property of the dose object to `true`.

When there are multiple active `ScheduleDose` objects on a model, and there are duplicate specifications for a property value, the simulation uses the last occurrence for the property value in the array of doses. You can find out which dose you applied last by looking at the indices of the dose objects stored on the model.

See the “Property Summary” on page 4-213 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` command to list object properties and the `set` command to change their values at the command line. Use can graphically change object properties in the graphical user interface

## Constructor Summary

<code>sbiodose</code>	Construct dose object
-----------------------	-----------------------

## Method Summary

Methods for variant objects

<code>copyobj (any object)</code>	Copy SimBiology object and its children
<code>get (any object)</code>	Get object properties
<code>set (any object)</code>	Set object properties

# ScheduleDose object

---

## Property Summary

Properties for variant objects

Active	Indicate object in use during simulation
Amount	Amount of dose
AmountUnits	Dose amount units
DurationParameterName	parameter length of time
Name	Specify name of object
Notes	HTML text describing SimBiology object
ParameterName	parameter
Parent	Indicate parent object
Rate	of dose
RateUnits	Units for dose rate
Tag	Specify label for SimBiology object
TargetName	Species receiving dose
Time	Simulation time steps or schedule dose times
TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type
UserData	Specify data to associate with object

## See Also

Model object, RepeatDose object, sbiodose, sbiosimulate

**Purpose** Select data from SimData object

**Syntax** `[t,x, names] = select(simDataObj, Query)`  
`[Out] = select(simDataObj, Query, 'Format', 'FormatValue')`

## Arguments

### Output Arguments

- t* An n-by-1 vector of time points.
- x* An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
- names* An m-by-1 cell array of names.
- Out* Data returned in the format specified in 'FormatValue', shown in "Input Arguments" on page 4-183. Depending on the specified 'FormatValue', *Out* contains one of the following:
- Structure array
  - SimData object
  - Time series object
  - Combined time series object from an array of SimData objects

### Input Arguments

*simDataObj* SimData object array. Enter a variable name for a SimData object.

## select (SimData)

---

**Query** A cell array of arguments consisting of some combination of property name/property value pairs and/or 'Where' clauses. For a more complete description of the query syntax, including 'Where' clauses and their supported condition types, see `sbioselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a `SimData` object in a query. These include 'Type', 'Name', 'Units', 'Compartment' (species only), or 'Reaction' (parameter only).

**FormatValue** Choose a format from the following table.

<b>FormatValue</b>	<b>Description</b>
'num'	Specifies the format that lets you return data in numeric arrays. This is the default when <code>select</code> is called with multiple output arguments.
'nummetadata'	Specifies the format that lets you return a cell array of metadata structures in <i>metadata</i> instead of names. The elements of <i>metadata</i> label the columns of <i>x</i> .
'numqualnames'	Specifies the format that lets you return qualified names in <i>names</i> to resolve ambiguities.
'struct'	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
'simdata'	Specifies the format that lets you return data in a new <code>SimData</code> object. This is the default format when <code>select</code> is called with zero or one output argument.
'ts'	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and <code>SimData</code> object in <code>simDataObj</code> .
'tslumped'	Specifies the format that lets you return data in time series objects, combining data from each <code>SimData</code> object into a single time series.

## Description

`[t,x, names] = select(simDataObj, Query)` returns simulation time and state data from the SimData object (`simDataObj`) that matches the query argument `Query`.

In a SimData object `simDataObj`, the columns of the data matrix `simDataObj.Data` are labeled by the cell array of metadata structures given by `simDataObj.DataInfo`. The `select` method enables you to pick out columns of the data matrix based on their metadata labels. For example, to extract data for all parameters logged in a SimData object `simDataObj`, use the syntax `[t, x, names] = select (simDataObj, {'Type', 'parameter'})`.

`[Out] = select(simDataObj, Query, 'Format', 'FormatValue')` returns the data in the specified format. Valid formats are listed in “Input Arguments” on page 4-183.

## Examples

This example shows how to extract data of interest from your simulation data with the `select` method.

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

```
sbioloadproject gprotein_norules m1
```

- 2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj = getconfigset(m1);  
set(csObj, 'SolverType', 'ssa');  
simDataObj = sbioenssemblerun(m1, 10);
```

- 3 Select all species data logged in the SimData array `sdarray`.

```
[t x n] = select(simDataObj, {'Type','species'});
```

- 4 Select data for the parameters with name 'Kd' and return the results in a new SimData object array.

```
newsd = select(simDataObj, {'Type','parameter','name', 'Kd'});
```

## select (SimData)

---

**5** This selects all data from `simDataObj` with a name that matches the pattern 'G' and returns time series objects.

```
ts = select(simDataObj, {'Where','Name','regexp','G'}, ...  
            'Format','ts');
```

### See Also

`getdata`, `sbioselect`, `sbiosimulate`, `selectbyname`, `Simdata` object

**Purpose** Select data by name from SimData object array

**Syntax** `[t,x,n] = selectbyname(simDataObj, 'NameValue')`  
`Out = selectbyname(simDataObj, NameValue, 'Format', Format)`

## Arguments

### Output Arguments

- t* An n-by-1 vector of time points.
- x* An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
- n* An m-by-1 cell array of names.
- Out* Data returned in the format as specified in '*FormatValue*', shown in "Input Arguments" on page 4-187. Depending on the specified '*FormatValue*', *Out* contains one of the following:
- Structure array
  - SimData object
  - Time series object
  - Combined time series object from an array of SimData objects

### Input Arguments

- simDataObj* SimData object array. Enter a variable name for a SimData object.
- NameValue* Names of the states for which you want to select data from *simDataObj*. Must be either a string or a cell array of strings.

## selectbyname (SimData)

---

*Query* A cell array of arguments consisting of some combination of property name/property value pairs and/or 'Where' clauses. For a more complete description of the query syntax, including 'Where' clauses and their supported condition types, see `sbioselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a `SimData` object. These include 'Type', 'Name', 'Units', 'Compartment' (species only), or 'Reaction' (parameter only).

*FormatValue* Choose a format from the following table.

<b>FormatValue</b>	<b>Description</b>
'num'	Specifies the format that lets you return data in numeric arrays. This is the default when <code>selectbyname</code> is called with multiple output arguments.
'nummetadata'	Specifies the format that lets you return a cell array of metadata structures in <i>metadata</i> instead of names. The elements of <i>metadata</i> label the columns of <i>x</i> .
'numqualnames'	Specifies the format that lets you return qualified names in <i>names</i> to resolve ambiguities.
'struct'	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
'simdata'	Specifies the format that lets you return data in a new <code>SimData</code> object. This is the default format when <code>selectbyname</code> is called with zero or one output argument.



FormatValue	Description
'ts'	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in <code>simDataObj</code> .
'tslumped'	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

## Description

The `selectbyname` method allows you to select data from a `SimData` object array by name. `[t,x,n] = selectbyname(simDataObj, 'NameValue')` returns time and state data from the `SimData` object `simDataObj` for states with names `'NameValue'`.

In a `SimData` object `simDataObj`, the names labeling the columns of the data matrix `simDataObj.Data` are given by `simDataObj.DataNames`. A name specified in `'NameValue'` can match more than one data column, for example, when `simDataObj` contains data for a species and parameter both named `'k'`. To resolve ambiguities, use qualified names in `'NameValue'`, such as `'CompartmentName.SpeciesName'` or `'ReactionName.ParameterName'`. `selectbyname` returns qualified names in the output argument `names` when there are ambiguities.

`Out = selectbyname(simDataObj, NameValue, 'Format', Format)` returns the data in the specified format. Valid formats are listed in “Input Arguments” on page 4-187.

## Examples

```
% Get data for the species 'glucose' from the simdata array sdarray.
[t x n] = selectbyname(sdarray,'glucose');
```

```
% Get data for multiple states and return the results in a struct array.
s = selectbyname(sdarray,{'RexGFP';'nuc.GFP';'cytosol.GFP'},...
    'Format','struct');
```

## See Also

`getdata`, `sbioselect`, `sbiosimulate`

# set (any object)

---

**Purpose** Set object properties

**Syntax**  
`set(Obj, 'PropertyName', PropertyValue)`  
`set(Obj, 'PropertyName1', PropertyValue1, 'PropertyName2',  
PropertyValue2...)`

## Arguments

*Obj* Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, PKCompartment, PKData, PKModelDesign, PKModelMap, reaction, rule, SimData, species, or variant object.

*'PropertyName'* Name of the property to set.

*PropertyValue* Specify the value to set. Property values depend on the property being set. See the reference page for an object property for values that can be specified.

## Description

`set(Obj, 'PropertyName', PropertyValue)` sets the property '*PropertyName*' of the object *Obj*, to *PropertyValue*.

`set(Obj, 'PropertyName1', PropertyValue1, 'PropertyName2', PropertyValue2...)` sets the properties '*PropertyName1*' and '*PropertyName2*' to *PropertyValue1* and *PropertyValue2* respectively, and so on in sequence. You can specify multiple *PropertyName*, *PropertyValue* pairs.

When you want to change the name of a compartment, parameter, or species object, use the `rename` method instead of `set`. The `rename` method allows you to change the name and update the expressions in which these components are used.

## Examples

**1** Create a model object.

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

**2** Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

- 3 Set the ConstantValue property of the parameter object to false and verify.

MATLAB returns 1 for true and 0 for false.

```
set (parameterObj, 'ConstantValue', false);  
get(parameterObj, 'ConstantValue')
```

MATLAB returns

```
ans =
```

```
0
```

### See Also

get , rename, setactiveconfigset

# setactiveconfigset (model)

---

**Purpose** Set active configuration set for model object

**Syntax**  
`configsetObj = setactiveconfigset(modelObj, 'NameValue')`  
`configsetObj2 = setactiveconfigset(modelObj, configsetObj1)`

**Description**  
`configsetObj = setactiveconfigset(modelObj, 'NameValue')` sets the configuration set *NameValue* to be the active configuration set for the model *modelObj* and returns to *configsetObj*.  
`configsetObj2 = setactiveconfigset(modelObj, configsetObj1)` sets the configset *configsetObj1* to be the active configset for *modelObj* and returns to *configsetObj2*. Any change in one of these two configset objects *configsetObj1* and *configsetObj2* is reflected in the other. To copy over a configset object from one model object to another, use the `copyobj` method.

The active configuration set contains the settings that are be used during a simulation. A default configuration set is attached to any new model.

**Examples**  
**1** Create a model object by importing the `oscillator.xml` file, and add a Configset object to the model.

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

**2** Configure the simulation stop criteria by setting the `StopTime`, `MaximumNumberOfLogs`, and `MaximumWallClock` properties of the Configset object. Set the stop criteria to a simulation time of 3000 seconds, 50 logs, or a wall clock time of 10 seconds, whichever comes first.

```
set(configsetObj, 'StopTime', 3000, 'MaximumNumberOfLogs', 50,...  
    'MaximumWallClock', 10)  
get(configsetObj)
```

```
Active: 0  
CompileOptions: [1x1 SimBiology.CompileOptions]
```

## setactiveconfigset (model)

---

```
        Name: 'myset'  
        Notes: ''  
        RuntimeOptions: [1x1 SimBiology.RuntimeOptions]  
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]  
        SolverOptions: [1x1 SimBiology.ODESolverOptions]  
        SolverType: 'ode15s'  
        StopTime: 3000  
MaximumNumberOfLogs: 50  
        MaximumWallClock: 10  
        TimeUnits: 'second'  
        Type: 'configset'
```

- 3 Set the new Configset object to be active, simulate the model using the new Configset object, and plot the result.

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

### See Also

addconfigset, getconfigset, removeconfigset

# setparameter (kineticlaw)

---

**Purpose** Specify specific parameters in kinetic law object

**Syntax** `setparameter(kineticlawObj, 'ParameterVariablesValue',  
'ParameterVariableNamesValue')`

## Arguments

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

**Description** Configure ParameterVariableNames in the kinetic law object.

`setparameter(kineticlawObj, 'ParameterVariablesValue', 'ParameterVariableNamesValue')` configures the ParameterVariableNames property of the kinetic law object (kineticlawObj). ParameterVariableValue corresponds to one of the strings in kineticlawObj ParameterVariables property. The corresponding element in the kineticlawObjParameterVariableNames property is configured to ParameterVariableNamesValue. For example, if ParameterVariables is {'Vm', 'Km'} and ParameterVariablesValue is specified as Vm, then the first element of the ParameterVariableNames cell array is configured to ParameterVariableNamesValue.

## Examples

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

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

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

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

- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

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

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) that should be set. To set these variables:

```
setparameter(kineticlawObj, 'Vm', 'Va');  
setparameter(kineticlawObj, 'Km', 'Ka');
```

- 4 Verify that the parameter variables are correct.

```
get(kineticlawObj, 'ParameterVariableNames')
```

MATLAB returns:

```
ans =  
  
    'Va'    'Ka'
```

### See Also

addparameter, getspecies, setspecies

# setspecies (kineticlaw)

---

**Purpose** Specify species in kinetic law object

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

## Arguments

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

## Description

setspecies configures the kinetic law object SpeciesVariableNames property.

setspecies(kineticlawObj, 'SpeciesVariablesValue', 'SpeciesVariableNamesValue') configures the SpeciesVariableNames property of the kinetic law object, kineticlawObj. SpeciesVariablesValue corresponds to one of the strings in the SpeciesVariables property of kineticlawObj. The corresponding element in kineticlawObj SpeciesVariableNames property is configured to SpeciesVariableNamesValue.

For example, if SpeciesVariables are {'S', 'S1'} and SpeciesVariablesValue is specified as S1, the first element of the SpeciesVariableNames cell array is configured to SpeciesVariableNamesValue.

## Examples

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

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

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



- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

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

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has one species variable (S) that should be set. To set this variable:

```
setspecies(kineticlawObj, 'S', 'a');
```

- 4 Verify that the species variable is correct.

```
get(kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =
```

```
'a'
```

### See Also

addparameter, getspecies, setparameter

# SimBiology.export.Model.simulate

---

**Purpose** Simulate exported SimBiology model

**Syntax**

```
[t,x,names] = simulate(model)
[t,x,names] = simulate(model,initialValues)
[t,x,names] = simulate(model,initialValues,doses)
simDataObj = simulate( ___ )
```

**Description** [t,x,names] = simulate(model) simulates a model, using the default initial values specified by model.InitialValues (which are always equal to the InitialValue property on the corresponding ValueInfo object). simulate returns:

- Time samples, t.
- Variation in the quantity of states over time, x.
- Names for the rows and columns of x, names.

You can set additional simulation options using the property SimBiology.export.Model.SimulationOptions.

[t,x,names] = simulate(model,initialValues) simulates a model, using the values specified in initialValues as the initial values of the simulation.

[t,x,names] = simulate(model,initialValues,doses) simulates the model, using the specified initial values and doses.

simDataObj = simulate( \_\_\_ ) returns a SimData object that contains time and state data, as well as metadata, such as the types and names for the reported states. You can access the time, data, and names stores in simDataObj using the properties simDataObj.Time, simDataObj.Data, and simDataObj.DataNames, respectively. You can use any of the previous input arguments.

## Input Arguments

### model

SimBiology.export.Model object.

### initialValues

Vector of values for `simulate` to use as the initial values of the simulation. `initialValues` must have the same number of elements as `model.InitialValues`.

**Default:** Values specified in `model.InitialValues`.

## **doses**

Vector of dose objects specifying the doses used for simulation. The input dose objects must be a subset of the doses in the exported model, as returned by `getdose`.

**Default:** All dose objects in the exported model.

## **Output Arguments**

**t**

$n$ -by-1 vector of time samples from the simulation, where  $n$  is the number of time samples.

**x**

$n$ -by- $m$  matrix of simulation data, where  $n$  is the number of time samples and  $m$  is the number of states logged during the simulation. Each column of `x` describes the variation in the quantity of a state over time.

**names**

$m$ -by-1 cell array of strings with names labeling the rows and columns of `x`, respectively.

**simDataObj**

`SimData` object containing simulation time and state data, as well as metadata, such as the types and names for the reported states.

## **Examples**

### **Simulate an Exported SimBiology Model**

Load a sample SimBiology model object, and select the species `y1` and `y2` for simulation.

# SimBiology.export.Model.simulate

---

```
modelObj = sbmlimport('lotka');  
modelObj.getConfigset.RuntimeOptions.StatesToLog = ...  
    sbioselect(modelObj, 'Name', {'y1', 'y2'});
```

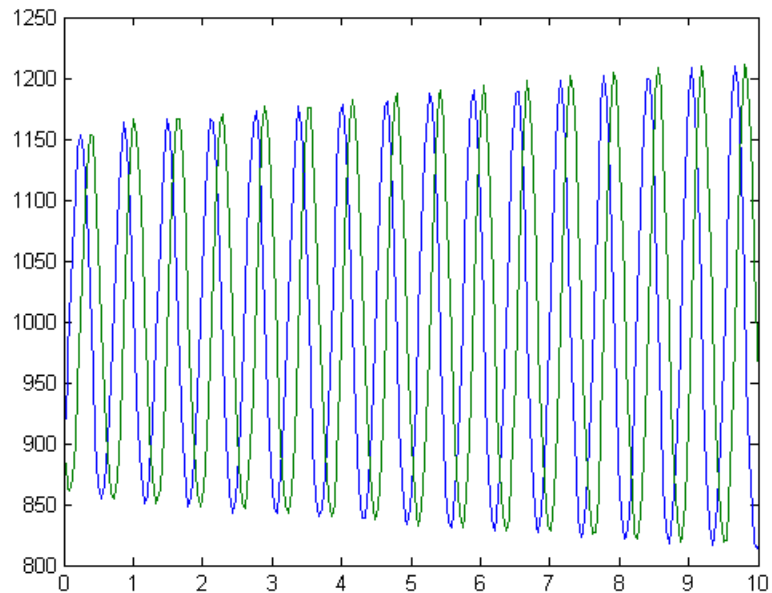
Export the model object.

```
em = export(modelObj);
```

Simulate the exported model.

```
[t,y] = simulate(em);
```

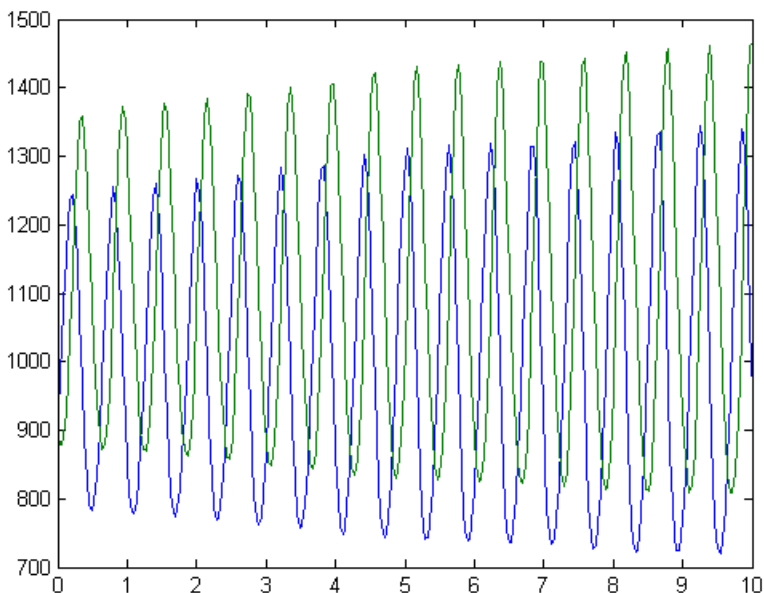
```
figure()  
plot(t,y)
```



Modify the initial conditions, and simulate again.

```
xIndex = em.getIndex('x');  
em.InitialValues(xIndex) = em.InitialValues(xIndex)*1.1;  
[t,y] = simulate(em);
```

```
figure()  
plot(t,y)
```



## See Also

[SimBiology.export.Model](#) | [SimBiology.export.Model.getdose](#)  
| [SimBiology.export.ValueInfo](#) |  
[SimBiology.export.SimulationOptions](#) | [SimData](#) object

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# SimData object

---

**Purpose** Simulation data storage

**Description** The SimBiology `SimData` object contains simulation data. The output from the `sbiosimulate` function, is stored in the `SimData` object which holds time and state data as well as metadata, such as the types and names for the logged states or the configuration set used during simulation.

You can also store data from multiple simulation runs as an array of `SimData` objects. Thus, the output of `sbioensemblerun` is an array of `SimData` objects. You can use any `SimData` method on an array of `SimData` objects.

You can access the time, data, and metadata stored in the `SimData` object through the properties in “Property Summary” on page 4-203. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line.

Methods you can use to query the `SimData` object are listed in “Method Summary” on page 4-202.

## Constructor Summary

<code>sbioensemblerun</code>	Multiple stochastic ensemble runs of SimBiology model
<code>sbiosimulate</code>	Simulate model object

## Method Summary

<code>delete</code> (any object)	Delete SimBiology object
<code>display</code> (any object)	Display summary of SimBiology object
<code>get</code> (any object)	Get object properties
<code>getdata</code> ( <code>SimData</code> )	Get data from <code>SimData</code> object array

getsensmatrix (SimData)	Get 3-D sensitivity matrix from SimData array
resample (SimData)	Resample SimData object array onto new time vector
select (SimData)	Select data from SimData object
selectbyname (SimData)	Select data by name from SimData object array
set (any object)	Set object properties

## Property Summary

Data	Store simulation data
DataCount	Numbers of species, parameters, sensitivities
DataInfo	Metadata labels for simulation data
DataNames	Show names in SimData object
ModelName	Name of model simulated
Name	Specify name of object
Notes	HTML text describing SimBiology object
RunInfo	Information about simulation
Time	Simulation time steps or schedule dose times
TimeUnits	Show time units for dosing and simulation
UserData	Specify data to associate with object

# SimData object

---

## Examples

Return simulation results to a SimData object and plot the results.

```
sbioloadproject('radiodecay', 'm1');
simDataObj = sbiosimulate(m1);
sbioplot(simDataObj)
```

Get simulation data at specific time points using the resample method.

```
% Load 'radiodecay' and set the initial amount of species 'x'.
sbioloadproject('radiodecay', 'm1');
x = sbioselect(m1, 'Type', 'species', 'Name', 'x');
x.InitialAmount = 100;
% Change the solver type to a stochastic solver.
cs = m1.getconfigset;
cs.SolverType = 'ssa';
% Simulate the model.
simDataObj = sbiosimulate(m1);
% This result could be misinterpreted as containing fractional molecules.
sbioplot(simDataObj);
title('Simulation Results Before Resampling');
%Resample the data using the zero-order hold method to obtain the correct
%number of molecules at intermediate time steps.
newsimDataObj = resample(simDataObj, linspace(0, 10, 1e4), 'zoh');
sbioplot(newsimDataObj);
title('Simulation Results After Resampling');
```

Initialize a simulation using results from a previous simulation.

```
% Load 'radiodecay'.
sbioloadproject('radiodecay', 'm1');
m1.Species
simDataObj = sbiosimulate(m1);
% Use the Data property to get the states at the final time point.
% Data is an m x n array, where m is the number of time steps in
% the simulation and n is the number of quantities logged.
finaldata = simDataObj.Data(end,:);
% Use the DataInfo property to get names of states.
info = simDataObj.DataInfo;
```



```
% Loop through the states (species) and set their initial amounts.
numSpecies = length(info);
for c = 1:numSpecies
    compObj = sbioselect(m1,'type','compartment','Name',info{c}.CompartmentName);
    speciesObj = sbioselect(compObj,'type','species','Name',info{c}.Name);
    speciesObj.InitialAmount = finaldata(c);
end
% Verify species initial amounts.
m1.Species
```

### See Also

Model object, Parameter object, Reaction object, Root object, Species object

# Species object

---

**Purpose** Options for compartment species

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

To add species that participate in reactions, add the reaction to the model. The process of adding the reaction to the model creates a compartment object (*unnamed*) and the necessary species objects.

Alternatively, create and add a species object to a compartment object, using the `addspecies` method at the command line.

When you use the SimBiology desktop to create a new model, it adds an empty compartment (*unnamed*), to which you can add species.

See “Property Summary” on page 4-207 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

**Constructor Summary**

<code>addspecies (model, compartment)</code>	Create species object and add to compartment object within model 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

get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

## Property Summary

Properties for species objects

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 SimBiology object type
UserData	Specify data to associate with object

## See Also

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

# Unit object

---

## Purpose

Hold information about user-defined unit

## Description

The SimBiology unit object holds information about user-defined units. To create a unit, create the unit object and add the unit to the library using the `sbiaddtolibrary` function.

Use the unit object property `Composition` to specify the composition of your units. See “Property Summary” on page 4-208 for links to unit object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change unit object properties in the SimBiology desktop, using the Library Explorer. For more information, see “What Are SimBiology Libraries?”.

## Constructor Summary

`sbiunit`

Create user-defined unit

## Method Summary

`delete (any object)`

Delete SimBiology object

`display (any object)`

Display summary of SimBiology object

`get (any object)`

Get object properties

`set (any object)`

Set object properties

## Property Summary

`Composition`

Unit composition

`Multiplier`

Relationship between defined unit and base unit

`Name`

Specify name of object

`Notes`

HTML text describing SimBiology object

Offset	Unit composition modifier
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## See Also

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

# UnitPrefix object

---

**Purpose** Hold information about user-defined unit prefix

**Description** The SimBiology unit prefix object holds information about user-defined unit prefixes. To create a unit prefix, create the unit prefix object and add the unit prefix to the library using the `sbiaddtolibrary` function.

Use the unit prefix object property `Exponent`, to specify the exponent of your unit prefix. See “Property Summary” on page 4-210 for links to unit prefix object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change unit prefix object properties in the SimBiology desktop, using the Library Explorer. For more information, see “What Are SimBiology Libraries?”.

**Constructor Summary**

<code>sbiunitprefix</code>	Create user-defined unit prefix
----------------------------	---------------------------------

**Method Summary**

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

**Property Summary**

<code>Exponent</code>	Exponent value of unit prefix
<code>Name</code>	Specify name of object
<code>Notes</code>	HTML text describing SimBiology object
<code>Parent</code>	Indicate parent object

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

## See Also

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

# Variant object

---

**Purpose** Store alternate component values

**Description** The SimBiology variant object stores the names and values of model components and allows you to use the values stored in a variant object as the alternate value to be applied during a simulation. You can store values for species `InitialAmount`, parameter `Value`, and compartment `Capacity` in a variant object. Simulating using a variant does not alter the model component values. The values specified in the variant temporarily apply during simulation.

Using one or more variant objects associated with a model allows you to evaluate model behavior during simulation, with different values for the various model components without having to search and replace these values, or having to create additional models with these values. If you determine that the values in a variant object accurately define your model, you can permanently replace the values in your model with the values stored in the variant object, using the `commit` method.

To use a variant in a simulation you must add the variant object to the model object and set the `Active` property of the variant to true. Set the `Active` property to true if you always want the variant to be applied before simulating the model. You can also enter the variant object as an argument to `sbiosimulate`; this applies the variant only for the current simulation and supersedes any active variant objects on the model.

When there are multiple active variant objects on a model, if there are duplicate specifications for a property's value, the last occurrence for the property value in the array of variants, is used during simulation. You can find out which variant is applied last by looking at the indices of the variant objects stored on the model. Similarly, in the `Content` property, if there are duplicate specifications for a property's value, the last occurrence for the property in the `Content` property, is used during simulation.

Use the `addcontent` method to append contents to a variant object.

See “Property Summary” on page 4-213 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change



their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

<code>sbiovariant</code>	Construct variant object
--------------------------	--------------------------

## Method Summary

Methods for variant objects

<code>addcontent (variant)</code>	Append content to variant object
<code>commit (variant)</code>	Commit variant contents to model
<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>get (any object)</code>	Get object properties
<code>rmcontent (variant)</code>	Remove contents from variant object
<code>set (any object)</code>	Set object properties
<code>verify (model, variant)</code>	Validate and verify SimBiology model

## Property Summary

Properties for variant objects

<code>Active</code>	Indicate object in use during simulation
<code>Content</code>	Contents of variant object
<code>Name</code>	Specify name of object
<code>Notes</code>	HTML text describing SimBiology object

## Variant object

---

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

### See Also

Compartment object, Configset object, Model object, Parameter object, Species object

`sbiosimulate`

**Purpose** Validate and verify SimBiology model

**Syntax**


```
verify(modelObj)
verify(modelObj, configsetObj)
verify(modelObj, variantObj)
verify(modelObj, configsetObj, variantObj)
```

**Description** `verify(modelObj)` performs checks on a model object (`modelObj`) to verify that you can simulate the model. This method generates stacked errors and warnings if any problems are found. To see the entire list of errors and warnings, use `sbiolasterror` and `sbiolastwarning`. The `verify` method uses the active configuration set for verification.

`verify(modelObj, configsetObj)` performs checks on the specified configuration set object (`configsetObj`) in conjunction with the model object (`modelObj`) to verify that you can simulate the model.

`verify(modelObj, variantObj)` performs checks on the variant object (`variantObj`) in conjunction with the model object (`modelObj`) to verify that you can simulate the model. The model object is required for the verification of the variant object.

`verify(modelObj, configsetObj, variantObj)` performs checks on the configuration set object `configsetObj`, and the variant object `variantObj` in conjunction with the model object (`modelObj`) to verify that you can simulate the model.

**Alternatives** If you are building your model in the SimBiology desktop, you can click  (when a model is selected in the Project Explorer) to generate a list of any errors and warnings in the model. The errors and warnings appear in the **Errors and Warnings** pane.

**Examples**

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

**See Also** `sbiolasterror`, `sbiolastwarning`

# verify (covmodel)

---

**Purpose** Check covariate model for errors

**Syntax** `verify(CovModelObj)`

**Description** `verify(CovModelObj)` verifies that the following are true about the Expression property of `CovModelObj`, a `CovariateModel` object:

- The expression strings are valid MATLAB code.
- Each expression string is linear with a transformation.
- There is exactly one expression string for each parameter.
- In each expression string, a covariate is used in at most one term.
- In each expression string, there is at most one random effect (`eta`)
- Fixed effect (`theta`) and random effect (`eta`) names are unique within and across expression strings. That is, each covariate has its own fixed effect.

If the previous requirements are true, then `verify` returns nothing.

**See Also** `construct` | `CovariateModel` | `Expression` | `PKModelDesign` object

- How To**
- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
  - “Specifying a Covariate Model”

# Property Reference

---

Abstract Kinetic Law (p. 5-3)	Properties for abstract kinetic law objects
Compartments (p. 5-4)	Properties for compartment objects
Configuration Sets (p. 5-5)	Properties for configuration set objects
CovariateModel (p. 5-6)	Properties for CovariateModel objects
Events (p. 5-7)	Properties for event objects
Kinetic Laws (p. 5-8)	Properties for kinetic law objects
Models (p. 5-9)	Properties for model objects
NMFileDef (p. 5-10)	Properties for NMFileDef objects
Parameters (p. 5-11)	Properties for parameter objects
PKCompartment (p. 5-12)	Properties for PKCompartment objects
PKData (p. 5-13)	Properties for PKData objects
PKModelDesign (p. 5-14)	Properties for PKModelDesign objects
PKModelMap (p. 5-15)	Properties for PKModelMap objects
Reactions (p. 5-16)	Properties for reaction objects
RepeatDose (p. 5-17)	Properties for RepeatDose objects
Root (p. 5-18)	Properties for the root object
Rules (p. 5-19)	Properties for rule objects

ScheduleDose (p. 5-20)	Properties for ScheduleDose objects
SimData (p. 5-21)	Properties for SimData objects
Species (p. 5-22)	Properties for species objects
Unit (p. 5-23)	Properties for unit objects
Unit Prefix (p. 5-24)	Properties for unit objects
Variant (p. 5-25)	Properties for variant objects
Using Object Properties (p. 5-26)	Command-line syntax for entering and retrieving property values

## Abstract Kinetic Law

Expression (AbstractKineticLaw, KineticLaw)	Expression to determine reaction rate equation
Name	Specify name of object
Notes	HTML text describing SimBiology object
ParameterVariables	Parameters in kinetic law definition
Parent	Indicate parent object
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Compartments

Capacity	Compartment capacity
CapacityUnits	Compartment capacity units
Compartments	Array of compartments in model or compartment
ConstantCapacity	Specify variable or constant compartment capacity
Name	Specify name of object
Notes	HTML text describing SimBiology object
Owner	Owning compartment
Parent	Indicate parent object
Species	Array of species in compartment object
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object



## Configuration Sets

Active	Indicate object in use during simulation
CompileOptions	Dimensional analysis and unit conversion options
MaximumNumberOfLogs	Maximum number of logs criteria to stop simulation
MaximumWallClock	Maximum elapsed wall clock time to stop simulation
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
StartTime	Start time for initial dose time
StopTime	Simulation time criteria to stop simulation
TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type

## CovariateModel

CovariateLabels (CovariateModel)	Labels for covariates in CovariateModel object
Expression (CovariateModel)	Define relationship between parameters and covariates
FixedEffectDescription (CovariateModel)	Descriptions of fixed effects in CovariateModel object
FixedEffectNames (CovariateModel)	Names of fixed effects in CovariateModel object
FixedEffectValues (CovariateModel)	Values for initial estimates of fixed effects in CovariateModel object
ParameterNames (CovariateModel)	Names of parameters in CovariateModel object
RandomEffectNames (CovariateModel)	Names of random effects in CovariateModel object

## Events

### Properties for event objects

Active	Indicate object in use during simulation
EventFcns	Event expression
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Trigger	Event trigger
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Kinetic Laws

Expression (AbstractKineticLaw, KineticLaw)	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 kinetic law definition
Parent	Indicate parent object
SpeciesVariableNames	Cell array of species in reaction rate equation
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Models

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

## **NMFileDef**

## Parameters

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 SimBiology object type
UserData	Specify data to associate with object
Value	Assign value to parameter object
ValueUnits	Parameter value units

## **PKCompartment**

DosingType	Drug dosing type in compartment
EliminationType	Drug elimination type from compartment
HasLag	Lag associated with dose targeting compartment
HasResponseVariable	Compartment drug concentration reported
Name	Specify name of object



## PKData

CovariateLabels	Identify covariate columns in data set
DataSet	Dataset object containing imported data
DependentVarLabel	Identify dependent variable column in data set
DependentVarUnits	Response units in PKData object
DoseLabel	Dose column in data set
DoseUnits	Dose units in PKData object
GroupID	Integer identifying each group in data set
GroupLabel	Identify group column in data set
GroupNames	Unique values from GroupLabel in data set
IndependentVarLabel	Identify independent variable column in data set
IndependentVarUnits	Time units in PKData object
RateLabel	Rate of infusion column in data set
RateUnits	Units for dose rate

## **PKModelDesign**

PKCompartments

Hold compartments in PK model

## PKModelMap

Dosed	Dosed object name
DosingType	Drug dosing type in compartment
Estimated	Names of parameters to estimate
LagParameter	Parameter specifying time lag for doses
Observed	Measured response object name
ZeroOrderDurationParameter	Zero-order dose absorption duration

## Reactions

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

## RepeatDose

Active	Indicate object in use during simulation
Amount	Amount of dose
AmountUnits	Dose amount units
DurationParameterName	parameter length of time
Interval	Time between doses
Name	Specify name of object
Notes	HTML text describing SimBiology object
ParameterName	parameter
Parent	Indicate parent object
Rate	of dose
RateUnits	Units for dose rate
RepeatCount	Dose repetitions
StartTime	Start time for initial dose time
Tag	Specify label for SimBiology object
TargetName	Species receiving dose
TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Root

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

## Rules

Active	Indicate object in use during simulation
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 SimBiology object type
UserData	Specify data to associate with object

## ScheduleDose

Active	Indicate object in use during simulation
Amount	Amount of dose
AmountUnits	Dose amount units
DurationParameterName	parameter length of time
Name	Specify name of object
Notes	HTML text describing SimBiology object
ParameterName	parameter
Parent	Indicate parent object
Rate	of dose
RateUnits	Units for dose rate
Tag	Specify label for SimBiology object
TargetName	Species receiving dose
Time	Simulation time steps or schedule dose times
TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type
UserData	Specify data to associate with object



## SimData

Data	Store simulation data
DataCount	Numbers of species, parameters, sensitivities
DataInfo	Metadata labels for simulation data
DataNames	Show names in SimData object
ModelName	Name of model simulated
Name	Specify name of object
Notes	HTML text describing SimBiology object
RunInfo	Information about simulation
Time	Simulation time steps or schedule dose times
TimeUnits	Show time units for dosing and simulation
UserData	Specify data to associate with object

## Species

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 SimBiology object type
UserData	Specify data to associate with object

## Unit

Composition	Unit composition
Multiplier	Relationship between defined unit and base unit
Name	Specify name of object
Notes	HTML text describing SimBiology object
Offset	Unit composition modifier
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Unit Prefix

Exponent	Exponent value of unit prefix
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

## Variant

Active	Indicate object in use during simulation
Content	Contents of variant object
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display 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. 5-26)	Use either MATLAB functions or object dot notation to enter or change property values.
Retrieving Property Values (p. 5-26)	Use either MATLAB functions or object dot notation to get property values.
Help for Objects, Methods, and Properties (p. 5-27)	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.

<code>help sbio</code>	Display a list of functions and methods.
<code>help FunctionName</code>	Display function information.





# Properties — Alphabetical List

---

# AbsoluteTolerance property

---

**Purpose** Absolute error tolerance applied to state value during simulation

**Description** AbsoluteTolerance is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

The software uses AbsoluteTolerance to determine the largest allowable absolute error at any step in a simulation. How the software uses AbsoluteTolerance to determine this error depends on whether the AbsoluteToleranceScaling property is enabled.

## When AbsoluteToleranceScaling Is Enabled

When the AbsoluteToleranceScaling property is enabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values whose size is of order 1. For all other state values, it scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

## When AbsoluteToleranceScaling Is Disabled

When the AbsoluteToleranceScaling property is disabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values, for example, amounts for all species.

**Algorithm** 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 varies 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 tradeoff 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 seem 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: <code>SolverOptions</code>
Data type	<code>double</code>
Data values	Positive scalar. Default is <code>1e-6</code> .
Access	Read/write

## Examples

This example shows how to change `AbsoluteTolerance`.

- 1 Retrieve the `configset` object from the `modelObj`.

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

- 2 Change the `AbsoluteTolerance` to `1e-8`.

```
set(configsetObj.SolverOptions, 'AbsoluteTolerance', 1.0e-8);  
get(configsetObj.SolverOptions, 'AbsoluteTolerance')
```

```
ans =
```

```
1.0000e-008
```

## See Also

`AbsoluteToleranceScaling`, `AbsoluteToleranceStepSize`,  
`RelativeTolerance`

# AbsoluteToleranceScaling property

---

**Purpose** Control scaling of absolute error tolerance during simulation

**Description** AbsoluteToleranceScaling is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

AbsoluteToleranceScaling controls how the software determines the largest allowable absolute error at any step in a simulation.

## When AbsoluteToleranceScaling Is Enabled

When the AbsoluteToleranceScaling property is enabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values whose size is of order 1. For all other state values, it scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

## When AbsoluteToleranceScaling Is Disabled

When the AbsoluteToleranceScaling property is disabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values, for example, amounts for all species.

## Characteristics

Applies to	Object: SolverOptions
Data type	logical
Data values	1, 0, true, or false. Default is true.
Access	Read/write

**See Also** AbsoluteTolerance, AbsoluteToleranceStepSize, RelativeTolerance

# AbsoluteToleranceStepSize property

---

**Purpose** Initial guess for time step size for scaling of absolute error tolerance

**Description** AbsoluteToleranceStepSize is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

When the AbsoluteToleranceScaling property is enabled, you can set the AbsoluteToleranceStepSize property to specify the initial guess for time step size for scaling. Then, for all state values whose size is of order 1, the software scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

---

**Tip** Use AbsoluteToleranceStepSize when a simulation is unsuccessful and other corrective actions do not work. For example, if state values change rapidly at the beginning of a simulation, then change more slowly. Iteratively set this property and simulate to find the optimal setting. As a starting point, MathWorks® recommends setting this property to  $\text{AbsoluteTolerance} * \text{StopTime} * 0.1$ .

---

## Characteristics

Applies to Object: SolverOptions

Data type double

Data values Scalar in units specified by TimeUnits property. Default is [].

Access Read/write

**See Also** AbsoluteTolerance, AbsoluteToleranceScaling, RelativeTolerance

# Active property

---

**Purpose** Indicate object in use during simulation

**Description** The Active property indicates whether a simulation is using a SimBiology object. A SimBiology model is organized into a hierarchical group of objects. Use the Active property to include or exclude objects during a simulation.

- **Configuration set** — For the configset object, use the method `setactiveconfigset` to set the object Active property to true.
- **Event, Reaction, or Rule** — When an event, a reaction, or rule object Active property is set to false, the simulation does not include the event, reaction, or rule. This is a convenient way to test a model with and without a reaction or rule.
- **Variant** — Set the Active property to true if you always want the variant to be applied before simulating the model. You can also pass the variant object as an argument to `sbiosimulate`; this applies the variant only for the current simulation. For more information on using the Active property for variants, see `Variant` object.

## Characteristics

Applies to	Objects: configset, event, reaction, RepeatDose, rule, ScheduleDose, variant
Data type	boolean
Data values	true or false. The default value for events, reactions, and rules is true. For the configset object, default is true. For added configset object, the default is false. For variants, the default is false.
Access	Read/write

## Examples

1 Create a model object.

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

- 2 Add a reaction object and verify that the Active property setting is 'true' or 1.

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

MATLAB returns:

```
ans =
```

```
1
```

- 3 Set the Active property to 'false' and verify.

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

MATLAB returns:

```
ans =
```

```
0
```

### See Also

addconfigset, addreaction, addrule, Event object, Reaction object, RepeatDose object, Rule object, ScheduleDose object, Variant object,

# Amount property

---

**Purpose** Amount of dose

**Description** Amount is a property of a RepeatDose or ScheduleDose object. It defines an increase in the amount of a SimBiology species that receives a dose.

A RepeatDose object defines a series of doses. Each dose is the same amount, as defined by the Amount property, and given at equally spaced times, as defined by the Interval property. The number of injections in the series, excluding the initial injection, is defined by the RepeatCount property, and the Rate property defines how fast each dose is given.

A ScheduleDose object defines a series of doses. Each dose can have a different amount, as defined by an amount array in the Amount property, and given at specified times, as defined by a time array in the Time property. A rate array in the Rate property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

<b>Characteristics</b>	Applies to	Object: RepeatDose, ScheduleDose
	Data type	double (RepeatDose) or double array (ScheduleDose)
	Data values	Nonnegative value. Default is 0 (RepeatDose) or [] (ScheduleDose)
	Access	Read/write

**See Also** ScheduleDose object and RepeatDose object



**Purpose** Dose amount units

**Description** AmountUnits is a property of a RepeatDose or ScheduleDose object. This property defines units for the Amount property.

If the TargetName property defines a species, then AmountUnits for a dose must be a chemical amount (for example, milligram, mole, or molecule), not a concentration. To get a list of the defined units in the library, use the sbioshowunits function. To add a user-defined unit to the list, see sbioaddtolibrary.

## Characteristics

Applies to	Objects: RepeatDose, ScheduleDose
Data type	string
Data values	Units from library with dimensions of amount. Default = "" (empty)
Access	Read/write

**See Also** ScheduleDose object and RepeatDose object

# Annotation property

---

**Purpose** Store link to URL or file

---

**Note** The Annotation property will be removed in a future release. Use the Notes property instead.

---

**Description** The Annotation property stores the URL or file name linking to information about a model.

## Characteristics

Applies to	SimBiology objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, or unit
Data type	char string, URL
Data values	Character string with a directory path and filename or a URL
Access	Read/write

## Examples

**1** Create a model object.

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

**2** Set the annotation for a model object.

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

**3** Verify the assignment.

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

MATLAB returns:

```
ans =
```

[www.reactome.org](http://www.reactome.org)

### **See Also**

`addkineticlaw`, `addparameter`, `addreaction`, `addrule`, `addspecies`,  
`sbiomodel`, `sbioroot`, `sbiunit`, `sbiunitprefix`, `RepeatDose`  
`object`, `ScheduleDose object`

# BoundaryCondition property

---

**Purpose** Indicate species boundary condition

**Description** The BoundaryCondition property indicates whether a species object has a boundary condition. If BoundaryCondition is true, the species quantity is determined by InitialAmount and/or a rule object, and not by the reaction rate equation. All SimBiology species are state variables regardless of the BoundaryCondition or ConstantAmount property.

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

## More Information

Consider the following two use cases of boundary conditions:

- Modeling receptor-ligand interactions that affect the rate of change of the receptor but not the ligand. For example, in response to hormone, steroid receptors such as the glucocorticoid receptor (GR) translocate from the cytoplasm (cyt) to the nucleus (nuc). The hsp90/hsp70 chaperone complex directs this nuclear translocation [Pratt 2004]. The natural ligand for GR is cortisol; the synthetic hormone dexamethasone (dex) is used in place of cortisol in experimental systems. In this system dexamethasone participates in the reaction but the quantity of dexamethasone in the cell is regulated using a rule. To simply model translocation of GR you could use the following reactions:

Formation of the chaperone-receptor complex,

```
Hsp90_complex + GR_cyt -> Hsp90_complex:GR_cyt
```

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

```
Hsp90_complex:GR_cyt + dex -> Hsp90_complex + GR_nuc + dex
```

For `dex`,

```
BoundaryCondition = true; ConstantAmount = false
```

In this example `dex` is modeled as a boundary condition with a rule to regulate the rate of change of `dex` in the system. Here, the quantity of `dex` is not determined by the rate of the second reaction but by a rate rule such as

$$ddex/dt = 0.001$$

which is specified in the SimBiology software as

```
dex = 0.001
```

- Modeling the role of nucleotides (for example, GTP, ATP, cAMP) and cofactors (for example,  $Ca^{++}$ ,  $NAD^+$ , coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.

```
Ras-GDP + GTP -> Ras-GTP + GDP
```

For GTP, `BoundaryCondition = true; ConstantAmount = true`

Model GTP and GDP with boundary conditions, thus making them *boundary species*. In addition, you can set the `ConstantAmount` property of these species to `true` to indicate that their quantity does not vary during a simulation.

## Characteristics

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

## Examples

- 1 Create a model object.

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

# BoundaryCondition property

---

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

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

MATLAB returns:

```
ans =  
  
0
```

- 3 Set the boundary condition to 'true' and verify.

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

MATLAB returns:

```
ans =  
  
1
```

## References

Pratt, W.B., Galigniana, M.D., Morishima, Y., Murphy, P.J. (2004), Role of molecular chaperones in steroid receptor action, *Essays Biochem*, 40:41-58.

## See Also

addrule, addspecies, ConstantAmount, InitialAmount

**Purpose** Library of built-in components

**Description** `BuiltInLibrary` is a SimBiology root object property containing all built-in components namely, unit, unit-prefixes, and kinetic laws that are shipped with the SimBiology product. You cannot add, modify, or delete components in the built-in library. The `BuiltInLibrary` property is an object that contains the following properties:

- **Units** — contains all units that are shipped with the SimBiology product. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in units either by using the command `sbiowhos -builtin -unit`, or by accessing the root object.
- **UnitPrefixes** — contains all unit-prefixes that are shipped with the SimBiology product. You can specify unit—prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in unit-prefixes either by using the command `sbiowhos -builtin -unitprefix`, or by accessing the root object.
- **KineticLaws** — contains all kinetic laws that are shipped with the SimBiology product. Use the command `sbiowhos -builtin -kineticlaw` to see the list of built-in kinetic laws. You can use built-in kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, `kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');`

See “Kinetic Law Definition” on page 6-65 for a definition and more information.

**Characteristics** `BuiltInLibrary`

# BuiltInLibrary property

---

Applies to	Object: root
Data type	object
Data values	Unit, unit-prefix, and abstract kinetic law objects
Access	Read-only

Characteristics for BuiltInLibrary properties:

- Units

Applies to	BuiltInLibrary property
Data type	unit objects
Data values	units
Access	Read-only

- UnitPrefixes

Applies to	BuiltInLibrary property
Data type	unit prefix objects
Data values	unit prefixes
Access	Read-only

- KineticLaws

Applies to	BuiltInLibrary property
Data type	Abstract kinetic law object
Data values	kinetic laws
Access	Read-only



## Examples

### Example 1

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

```
sbiowhos -builtin -kineticlaw
sbiowhos -builtin -unit
sbiowhos -builtin -unitprefix
```

### Example 2

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

```
rootObj = sbioroot;
get(rootObj.BuiltinLibrary, 'KineticLaws')
get(rootObj.BuiltinLibrary, 'Units')
get(rootObj.BuiltinLibrary, 'UnitPrefixes')
```

## See Also

Functions — `sbioaddtolibrary`, `sbioremovefromlibrary` `sbioroot`, `sbiounit`, `sbiounitprefix`

Properties — `UserDefinedLibrary`

# Capacity property

---

**Purpose**                   Compartment capacity

**Description**           The Capacity property indicates the size of the SimBiology compartment object. If the size of the compartment does not vary during simulation, set the property `ConstantCapacity` to `true`.

You can vary compartment capacity using rules or events.

---

**Note** Remember to set the `ConstantCapacity` property to `false` for varying capacity.

---

Events cannot result in the capacity having a negative value. Rules could result in the capacity having a negative value.

## Characteristics

Applies to	Object: compartment
Data type	double
Data values	Positive real number. The default value is 1.
Access	Read/write

## Examples

Add a compartment to a model and set the compartment capacity.

**1** Create a model object named `my_model`.

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

**2** Add the compartment object named `nucleus` with a capacity of 0.5.

```
compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);
```

## See Also

`addcompartment`, `addspecies`, `CapacityUnits`, `ConstantCapacity`

**Purpose**                    Compartment capacity units

**Description**            The CapacityUnits property indicates the unit definition for the Capacity property of a compartment object. CapacityUnits can be any unit from the units library. To get a list of the defined units in the library, use the sbioshowunits function. If CapacityUnits changes from one unit definition to another, the Capacity does not automatically convert to the new units. The sbioconvertunits function does this conversion. To add a user-defined unit to the list, see sbioaddtolibrary.

## Characteristics

Applies to	Object: compartment
Data type	char string
Data values	Units from library with dimensions of length, area, or volume. Default = '' (empty).
Access	Read/write

## Examples

**1** Create a model object named my\_model.

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

**2** Add a compartment object named cytoplasm with a capacity of 0.5.

```
compObj = addcompartment (modelObj, 'cytoplasm', 0.5);
```

**3** Set the CapacityUnits to femtoliter, and verify.

```
set (compObj, 'CapacityUnits', 'femtoliter');  
get (compObj, 'CapacityUnits')
```

MATLAB returns:

```
ans =
```

```
femtoliter
```

## CapacityUnits property

---

### **See Also**

InitialAmount, sbioaddtolibrary, sbioconvertunits,  
sbioshowunits

**Purpose** Array of compartments in model or compartment

**Description** `Compartments` shows you a read-only array of SimBiology compartment objects in the model object and the compartment object. In the model object, the `Compartments` property indicates all the compartments in a `Model` object as a flat list. In the compartment object, the `Compartments` property indicates other compartments that are referenced within the compartment. The two instances of `Compartments` are illustrated in “Examples” on page 6-21.

You can add a compartment object using the method `addcompartment`.

## Characteristics

Applies to	Objects: compartment, model
Data type	Array of compartment objects
Data values	Compartment object. Default is [] (empty).
Access	Read-only

## Examples

**1** Create a model object named `modelObj`.

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

**2** Add two compartments to the model object.

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

**3** Add a compartment to one of the compartment objects.

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

**4** Display the `Compartments` property in the model object.

```
get(modelObj, 'Compartments')  
  
SimBiology Compartment Array
```

# Compartments property

---

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

- 5 Display the Compartments property in the compartment object.

```
get(compartmentObj2, 'Compartments')
```

```
SimBiology Compartment - matrix
```

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

## See Also

`addcompartment`, `addreaction`, `addspecies`, `Compartment` object

**Purpose** Dimensional analysis and unit conversion options

**Description** The SimBiology CompileOptions property is an object that defines the compile options available for simulation; you can specify whether dimensional analysis and unit conversion is necessary for simulation. Compile options are checked during compile time. The compile options object can be accessed through the CompileOptions property of the configset object. Retrieve CompileOptions object properties with the get function and configure the properties with the set function.

## Property Summary

DefaultSpeciesDimension	Dimension of species name in expression
DimensionalAnalysis	Perform dimensional analysis on model
Type	Display SimBiology object type
UnitConversion	Perform unit conversion

## Characteristics

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

## Examples

**1** Retrieve the configset object of modelObj.

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

**2** Retrieve the CompileOptions object (optionsObj) from the configsetObj.

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

# CompileOptions property

---

Compile Settings:

UnitConversion: false  
DimensionalAnalysis: true

**See Also**      get, set



**Purpose** Unit composition

**Description** The Composition property holds the composition of a unit object. The Composition property shows the combination of base and derived units that defines the unit. For example, molarity is the name of the unit and the composition is mole/liter. Base units are the set of units used to define all unit quantity equations. Derived units are defined using base units or mixtures of base and derived units.

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

## Characteristics

Applies to	Object: Unit
Data type	char string
Data values	Valid combination of units and prefixes from the library. Default is '' (empty).
Access	Read/write

## Examples

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

**1** Create a unit for the rate constants of a second-order reaction.

```
unitObj = sbiounit('secondconstant', '1/molarity*second', 1);
```

**2** Query the Composition property.

```
get(unitObj, 'Composition')
```

```
ans =
```

```
1/molarity*second
```

**3** Change the Composition property.

# Composition property

---

```
set(unitObj, 'Composition', 'liter/mole*second'))
```

```
ans =
```

```
liter/mole*second
```

**4** Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

## See Also

get, Multiplier, Offset, sbiounit, set

**Purpose** Specify variable or constant species amount

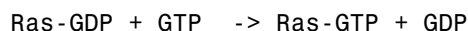
**Description** The ConstantAmount property indicates whether the quantity of the species object can vary during the simulation. ConstantAmount can be either true or false. If ConstantAmount is true, the quantity of the species cannot vary during the simulation. By default, ConstantAmount is false and the quantity of the species can vary during the simulation. If ConstantAmount is false, the quantity of the species can be determined by reactions and rules.

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

### More Information

The following is an example of modeling species as constant amounts:

Modeling the role of nucleotides (GTP, ATP, cAMP) and cofactors (Ca<sup>++</sup>, NAD<sup>+</sup>, 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*.

### Characteristics

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

### Examples

1 Create a model object named my\_model.

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

## ConstantAmount property

---

- 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 the constant amount to 'true' and verify.

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

MATLAB returns:

```
ans =
```

```
1
```

### See Also

addspecies, BoundaryCondition

**Purpose** Specify variable or constant compartment capacity

**Description** The ConstantCapacity property indicates whether the capacity of the compartment object can vary during the simulation. ConstantCapacity can be either true (1) or false (0). If ConstantCapacity is true, the quantity of the compartment cannot vary during the simulation. By default, ConstantCapacity is true and the quantity of the compartment cannot vary during the simulation. If ConstantCapacity is false, the quantity of the compartment can be determined by rules and events.

## Characteristics

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

## Examples

Add a compartment to a model and check the ConstantCapacity property of the compartment.

**1** Create a model object named comp\_model.

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

**2** Add the compartment object named nucleus with a capacity of 0.5.

```
compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);
```

**3** Display the ConstantCapacity property.

```
get(compartmentObj, 'ConstantCapacity')
```

```
ans =
```

```
1
```

## See Also

addcompartment, ConstantAmount, ConstantValue

# ConstantValue property

---

**Purpose** Specify variable or constant parameter value

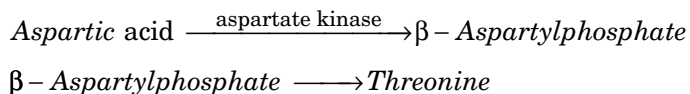
**Description** The ConstantValue property indicates whether the value of a parameter can change during a simulation. Enter either `true` (value is constant) or `false` (value can change).

You can allow the value of the parameter to change during a simulation by specifying a rule that changes the Value property of the parameter object.

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

## More Information

As an example, consider feedback inhibition of an enzyme such as aspartate kinase by threonine. Aspartate kinase has three isozymes that are independently inhibited by the products of downstream reactions (threonine, homoserine, and lysine). Although threonine is made through a series of reactions in the synthesis pathway, for illustration, the reactions are simplified as follows:



To model inhibition of aspartate kinase by threonine, you could use a rule like the algebraic rule below to vary the rate of the above reaction and simulate inhibition. In the rule, the rate constant for the above reaction is denoted by `k_aspartate_kinase` and the quantity of threonine is `threonine`.

```
k_aspartate_kinase -(1/threonine)
```

## Characteristics

Applies to	Object: parameter
Data type	boolean

Data values	true or false. The default value is 'true'.
Access	Read/write

## Examples

- 1 Create a model object.

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

- 2 Add a parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

- 3 Change the ConstantValue property of the parameter object from default (true) to false and verify.

MATLAB returns 1 for true and 0 for false.

```
set (parameterObj, 'ConstantValue', false);  
get(parameterObj, 'ConstantValue')
```

MATLAB returns:

```
ans =
```

```
0
```

## See Also

`addparameter`

# Content property

---

**Purpose** Contents of variant object

**Description** The Content property contains the data for the variant object. Content is a cell array with the structure {'Type', 'Name', 'PropertyName', 'PropertyValue'}. You can store values for species InitialAmount, parameter Value, and compartment Capacity, in a variant object.

For more information about variants, see Variant object.

## Characteristics

Applies to	Object: variant
Data type	cell array
Data values	Default value is [] (empty).
Access	Read/write

## Examples

**1** Create a model containing three species in one compartment.

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

**2** Add a variant object that varies the species' InitialAmount property.

```
variantObj = addvariant(modelObj, 'v1');  
addcontent(variantObj, {'species','A', 'InitialAmount', 5}, ...  
{'species', 'B', 'InitialAmount', 10});  
% Display the variant  
variantObj
```

```
SimBiology Variant - v1 (inactive)
```

ContentIndex:	Type:	Name:	Property:	Value:
1	species	A	InitialAmount	5
2	species	B	InitialAmount	10



### 3 Append data to the Content property.

```
addcontent(variantObj, {'species', 'C', 'InitialAmount', 15});
```

```
SimBiology Variant - v1 (inactive)
```

ContentIndex:	Type:	Name:	Property:	Value:
1	species	A	InitialAmount	5
2	species	B	InitialAmount	10
3	species	C	InitialAmount	15

### 4 Remove a species from the Content property.

```
rmcontent(variantObj, 3);
```

### 5 Replace the data in the Content property.

```
set(variantObj, 'Content', {'species', 'C', 'InitialAmount', 15});
```

## See Also

addcontent, rmcontent, sbiovariant

# CovariateLabels property

---

**Purpose** Identify covariate columns in data set

**Description** CovariateLabels is a property of the PKData object. It specifies the column in DataSet that contains the covariate data.

## Characteristics

Applies to	Object: PKData
Data type	char string or cell array of strings
Data values	Column headers from imported data set
Access	Read/write

**See Also** PKData object

**How To**

- “Specifying and Classifying the Data to Fit”
- “Specifying a Covariate Model”

# CovariateLabels (CovariateModel) property

---

**Purpose** Labels for covariates in CovariateModel object

**Description** The CovariateLabels property is a cell array of strings specifying the labels for the covariates in the Expression property of a CovariateModel object.

## Characteristics

Applies to	Object: CovariateModel
Data type	Cell array of strings
Data values	Labels for the covariates in the Expression property
Access	Read only

**See Also** CovariateModel | Expression

# Data property

---

**Purpose** Store simulation data

**Description** The Data property contains the simulation data stored in the SimData object.

This property contains all data logged during a simulation, including species amounts, parameter values, and sensitivities. The property is an  $m \times n$  array, where  $m$  is the number of time steps in the simulation and  $n$  is the number of quantities logged. The rows of the array are labeled by the time points in the Time property, and the columns are labeled by the metadata in the DataInfo property.

## Characteristics

Applies to	Object: SimData
Data type	double
Data values	Default value is [] (empty).
Access	Read-only

**See Also** DataInfo, ModelName

**Purpose** Numbers of species, parameters, sensitivities

**Description** The DataCount property shows how many species, parameters, and sensitivities are logged in a SimData object. It is a MATLAB structure with the fields `Species`, `Parameter`, and `Sensitivity`. The information in this property is redundant with the `DataInfo` property and is there to give you a convenient means to access the information.

## Characteristics

Applies to	Object: SimData
Data type	struct
Data values	Default value for each field is 0.
Access	Read-only

**See Also** `StopTime`, `StopTimeType`

# DataInfo property

---

**Purpose** Metadata labels for simulation data

**Description** The DataInfo property contains the metadata that label the columns of the SimData object array. It is an  $n \times 1$  cell array of structures. The  $i$ th cell contains metadata labeling the  $i$ th column of the SimData object array.

The possible types of structures are as follows.

Type	Fields
Species	Type: species Name: Compartment: Units:
Parameter	Type: parameter Name: Reaction: <name of reaction that a parameter is scoped to, or '' if parameter is scoped to model> Units:
Sensitivity	Type: sensitivity Name: <for example: $d[x]/d[y]_0$ > OutputType: <The type of the sensitivity output, either 'species' or 'parameter'> OutputName: <The name of the sensitivity output> OutputQualifier: <The compartment or reaction for the sensitivity output, for species or parameters, respectively> InputType: <The type of the sensitivity input, either 'species' or 'parameter'> InputName: <The name of the sensitivity input> InputQualifier: <The compartment or reaction for

Type	Fields
	Units: the sensitivity input, for species or parameters, respectively>

## Characteristics

Applies to	Object: SimData
Data type	n x 1 cell array of structs
Data values	Default value is 0x1 cell array.
Access	Read-only

**See Also** StopTime, StopTimeType

# DataNames property

---

**Purpose** Show names in SimData object

**Description** The DataNames property holds the names that label the columns of the data matrix in the Data property. The property contains an nx1 array of strings. The software provides this information for your convenience.

## Characteristics

Applies to	Object: SimData
Data type	string array
Data values	Default value is 0x1 cell array.
Access	Read-only

**See Also** StopTime, StopTimeType



**Purpose** Dataset object containing imported data

**Description** DataSet is a property of the PKData object. It contains the imported data set. The PKData object constructor (PKData) assigns the specified data set to its DataSet property during construction.

## Characteristics

Applies to	Object: PKData
Data type	dataset object
Data values	Variable containing dataset object
Access	Read-only

**See Also** “Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, PKData object

# DefaultSpeciesDimension property

---

**Purpose** Dimension of species name in expression

**Description** The `DefaultSpeciesDimension` property specifies how SimBiology interprets species names in expressions (namely reaction rate, rule, or event expressions). The valid property values are `substance` or `concentration`. If you specify `InitialAmountUnits`, SimBiology interprets species names appearing in expressions as concentration or substance amount according to the units specified, regardless of the value in `DefaultSpeciesDimension`. Thus, if `DefaultSpeciesDimension` is `concentration` and you specify species units as `molecule`, SimBiology interprets species names in expressions as `substance`. This interpretation applies even when `DimensionalAnalysis` is off.

You can find `DefaultSpeciesDimension` in the `CompileOptions` property.

When you set `DefaultSpeciesDimension` to `substance`, if you do not specify units, SimBiology interprets species names appearing in expressions as substance amounts, and does not scale by compartment capacity. To include a species concentration in an expression, divide by the appropriate compartment capacity in the expression. To specify compartment capacity in an expression enter the compartment name.

When you set `DefaultSpeciesDimension` to `concentration`, SimBiology interprets species names appearing in expressions as concentrations, and scales by compartment capacity in the expressions. To include a species amount in an expression, multiply by the species name by the appropriate compartment name in the expression.

For information on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated”.

## Characteristics

Applies to	Object: <code>CompileOptions</code> (in configset object)
Data type	<code>char</code> string

# DefaultSpeciesDimension property

---

Data values	concentration or substance. Default value is concentration.
Access	Read/write

## See Also

CompileOptions, DimensionalAnalysis, get, getConfigset, sbiosimulate, set

# DependentVarLabel property

---

**Purpose** Identify dependent variable column in data set

**Description** `DependentVarLabel` is a property of a `PKData` object. It specifies the column(s) in `DataSet` that contain the dependent variable(s), for example, measured response(s). The column must contain numeric values, and cannot contain `Inf` or `-Inf`.

## Characteristics

Applies to	Object: <code>PKData</code>
Data type	<code>char</code> string or cell array of strings
Data values	Column header from an imported data set
Access	Read/write

**See Also** “Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, `PKData` object

**Purpose** Response units in PKData object

**Description** `DependentVarUnits` is a property of a `PKData` object. It specifies the units for the column(s) containing the dependent variable(s) (responses) in the imported data set. If unit conversion is on, plot results in the SimBiology desktop show the units specified in `DependentVarUnits`.

To get a list of units, use the `sbioshowunits` function.

## Characteristics

Applies to	Object: <code>PKData</code>
Data type	<code>char</code> string or cell array of strings
Data values	Units from the units library. Default is an empty cell array.

---

**Tip** If there are no units associated with the dependent variable(s) in your data set, you can set this property to a cell array of empty strings, or simply an empty cell array.

---

Access	Read/write
--------	------------

**See Also** `DependentVarLabel`, `PKData` Object

# DimensionalAnalysis property

---

**Purpose** Perform dimensional analysis on model

**Description** The DimensionalAnalysis property specifies whether to perform dimensional analysis on the model before simulation. It is a property of the CompileOptions object. CompileOptions holds the model's compile time options and is the object property of the configset object. When DimensionalAnalysis is set to true, the SimBiology software checks whether the physical quantities of the units involved in reactions and rules, match and are applicable.

For example, consider a reaction  $a + b \rightarrow c$ . 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.

Unit conversion requires dimensional analysis. If DimensionalAnalysis is off, and you turn UnitConversion on, then DimensionalAnalysis is turned on automatically. If UnitConversion is on and you turn off DimensionalAnalysis, then UnitConversion is turned off automatically.

If you have MATLAB function calls in your model, dimensional analysis ignores any expressions containing function calls and generates a warning.

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

## Characteristics

Applies to	Object: CompileOptions (in configset object)
Data type	boolean

Data values	true or false. Default value is true.
Access	Read/write

## Examples

This example shows how to retrieve and set `DimensionalAnalysis` from the default `true` to `false` in the default configuration set in a model object.

### 1 Import a model.

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

```
SimBiology Model - Oscillator
```

```
Model Components:
```

```
Models:          0
Parameters:      0
Reactions:       42
Rules:           0
Species:         23
```

### 2 Retrieve the configset object of the model object.

```
configsetObj = getconfigset(modelObj)
```

```
Configuration Settings - default (active)
```

```
SolverType:      ode15s
StopTime:        10.000000
```

```
SolverOptions:
```

```
AbsoluteTolerance: 1.000000e-006
RelativeTolerance: 1.000000e-003
```

```
RuntimeOptions:
```

```
StatesToLog:     all
```

# DimensionalAnalysis property

---

```
CompileOptions:  
  UnitConversion:      true  
  DimensionalAnalysis: true
```

**3** Retrieve the CompileOptions object.

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

Compile Settings:

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

**4** Assign a value of false to DimensionalAnalysis.

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

## See Also

get, getconfigset, sbiosimulate, set



**Purpose** Dosed object name

**Description** Dosed is a property of the PKModelMap object. It specifies the name(s) of species object(s) that receive an input, such as a drug in a compartment or a ligand.

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the Dosed property and the DosingType property.

## Characteristics

Applies to	Object: PKModelMap
Data type	char string or cell array of strings
Data values	Name of a species object or empty. Default is an empty cell array.
Access	Read/write

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, DosingType, Estimated, Observed, PKModelMap object

# DoseLabel property

---

**Purpose** Dose column in data set

**Description** DoseLabel is a property of the PKData object. DoseLabel specifies the column that contains that contains the dosing information, in DataSet. The column must contain positive values, and cannot contain Inf or Inf.

## Characteristics

Applies to	Object: PKData
Data type	string or array of strings
Data values	Column headers from imported data set
Access	Read/write

**See Also** PKData object, sbionmimport, sbionmfiledef, “Specifying and Classifying the Data to Fit” in the SimBiology documentation

**Purpose** Dose units in PKData object

**Description** The DoseUnits property indicates the units for dose values in the PKData object. Dose units must have dimensions of amount or mass. The length of DoseUnits must be the same as DoseLabel. For example, if the DoseLabel property defines two columns containing dosing information, DoseUnits must also define units for both columns. If unit conversion is on, dose and rate units must be consistent with each other (that is in terms of amount or mass) and must be consistent with the species object that is being dosed.

To get a list of units, use the sbioshowunits function.

## Characteristics

Applies to	Object: PKData
Data type	string or array of strings
Data values	Units from units library. Default is '' (empty).
Access	Read/write

**See Also** DoseLabel, PKData Object

# DosingType property

---

**Purpose** Drug dosing type in compartment

**Description** DosingType is a property of the PKCompartment and PKModelMap objects. It specifies the type of dosing of a drug in a compartment. You can only dose one compartment in the model at any given time. For a description of the types of dosing supported, the model components created for each type of dosing, and the parameters to estimate, see “Dosing Types”.

## Characteristics

Applies to	Objects: PKCompartment, PKModelMap
Data type	char string or cell array of strings
Data values	' ', 'Bolus', 'Infusion', 'ZeroOrder', 'FirstOrder'
Access	Read/write

**See Also** EliminationType, PKCompartment object, PKModelMap object

# DurationParameterName property

---

**Purpose** Parameter specifying length of time

**Description** DurationParameterName is a property of a RepeatDose or ScheduleDose object.

Specify the name of a parameter object that is:

- Scoped to a model
- Constant, that is, its ConstantValue property is true

This property specifies the length of time it takes to administer a dose.

---

**Note** If you set the DurationParameterName property of a dose, you must also specify the Amount property of the dose, and set the Rate property to 0. This is because the rate is calculated from the amount and duration.

---

## Characteristics

Applies to Objects: RepeatDose, ScheduleDose

Data type char string

Data values Name of a parameter object or empty. Default is an empty string.

The parameter object must be:

- Scoped to a model
- Constant, that is, have a ConstantValue property set to true

Access Read/write

**See Also** RepeatDose object, ScheduleDose object

# EliminationType property

---

**Purpose** Drug elimination type from compartment

**Description** EliminationType is a property of the PKCompartment object. It specifies the type of elimination of a drug from a compartment. For a description of the types of elimination supported, the model components created for each type of elimination, and the parameters to estimate, see “Elimination Types”.

## Characteristics

Applies to	Object: PKCompartment
Data type	char string
Data values	'Linear', 'Linear-Clearance', 'Enzymatic', and ''
Access	Read/write

**See Also** addCompartment, DosingType, PKCompartment object

**Purpose** Specify explicit or implicit tau error tolerance

**Description** The ErrorTolerance property specifies the error tolerance for the explicit tau and implicit tau stochastic solvers. It is a property of the SolverOptions object. SolverOptions is a property of the configset object. The explicit and implicit tau solvers automatically chooses a time interval ( $\tau$ ) such that the relative change in the propensity function for each reaction is less than the user-specified error tolerance. A propensity function describes the probability that the reaction will occur in the next smallest time interval, given the conditions and constraints.

If the error tolerance is too large, there may not be a solution to the problem and that could lead to an error. If the error tolerance is small, the solver will take more steps than when the error tolerance is large leading to longer simulation times. The error tolerance should be adjusted depending upon the problem, but a good value for the error tolerance is between 1 % to 5 %.

## Characteristics

Applies to	Object: SolverOptions
Data type	double
Data values	>0, <1. The default is 3e-2.
Access	Read/write

**Examples** This example shows how to change ErrorTolerance settings.

- 1 Retrieve the configset object from the modelObj and change the SolverType to expltau.

```
modelObj = sbiomodel('cell');  
configsetObj = getConfigset(modelObj);  
set(configsetObj, 'SolverType', 'expltau')
```

- 2 Change the ErrorTolerance to 1e-8.

## ErrorTolerance property

---

```
set(configsetObj.SolverOptions, 'ErrorTolerance', 5.0e-2);  
get(configsetObj.SolverOptions, 'ErrorTolerance')
```

```
ans =
```

```
5.000000e-002
```

### **See Also**

LogDecimation, RandomState



**Purpose** Names of parameters to estimate

**Description** Estimated is a property of the PKModelMap object. It specifies the name(s) of the object(s) to estimate. Specify the name(s) of species, compartment, or parameter object(s) that are scoped to a model.

---

**Note** If you specify a species object, you are estimating the InitialAmount property of the species object.

---

## Characteristics

Applies to	Object: PKModelMap
Data type	char string or cell array of strings
Data values	Name of a species, compartment, or parameter object or empty. Default is an empty cell array.
Access	Read/write

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, Dosed, InitialAmount, Observed, PKModelMap object

# EventFcns property

---

**Purpose** Event expression

**Description** Property of the event object that defines what occurs when the event is triggered. Specify a cell array of strings.

EventFcns can be any MATLAB assignment or expression that defines what is executed when the event is triggered. All EventFcn expressions are assignments of the form '*objectname* = *expression*', where *objectname* is the name of a valid SimBiology object.

For more information about how SimBiology handles events, see “How Events Are Evaluated”. For examples of event functions, see “Specifying Event Functions”.

## Characteristics

Applies to	Object: event
Data type	Cell array of strings
Data values	EventFcn strings '' (empty)
Access	Read/write

## Examples

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

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

**2** Set the EventFcns property of the event object.

```
set(eventObj, 'EventFcns', {'pA = 0pA', 'mA = pol'});
```

**3** Get the EventFcns property.

```
get(eventObj, 'EventFcns')
```

## See Also

Event object, Trigger

**Purpose** Contain all event objects

**Description** Property to indicate events in a model object. Read-only array of Event objects.

An event defines an action when a defined condition is met. For example, the quantity of a species may double when the quantity of species B is 100. An event is triggered when the conditions specified in the event are met by the model. For more information, see “Events” and “Event Object”.

Add an event to a Model object with the method `addevent` method and remove an event with the `delete` method. See Event object for more information.

You can view event object properties with the `get` command and modify the properties with the `set` command.

## Characteristics

Applies to	Object: model
Data type	Array of event objects
Data values	Event object. The default is [] (empty).
Access	Read-only

## Examples

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

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

**2** Get a list of properties for an event object.

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

Or

```
get(eventObj)
```

## Events property

---

MATLAB displays a list of event properties.

```
Active: 1
Annotation: ''
EventFcns: {'OpC = 200'}
Name: ''
Notes: ''
Parent: [1x1 SimBiology.Model]
Tag: ''
Trigger: 'time >= 5'
TriggerDelay: 0
TriggerDelayUnits: 'second'
Type: 'event'
UserData: []
```

### See Also

EventFcns, Event object, Model object, Trigger

**Purpose** Exponent value of unit prefix

**Description** *Exponent* shows the value of  $10^{\text{Exponent}}$  that defines the numerical value of the unit prefix *Name*. You can use the unit prefix in conjunction with any built-in or user-defined units. For example, for the unit `mole`, specify as `picomole` to use the *Exponent*, `-12`.

## Characteristics

Applies to	Object: Unit prefix
Data type	<code>double</code>
Data values	Real number. Default is 0.
Access	Read/write

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

**1** Create a unit prefix.

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

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

```
sbioaddtolibrary(unitprefixObj1);
```

**3** Query the *Exponent* property.

```
get(unitprefixObj1, 'Exponent')
```

```
ans =
```

```
15
```

**See Also** `get`, `sbioaddtolibrary`, `sbiounitprefix`, `set`, `UnitPrefix` object

# Expression (CovariateModel) property

## Purpose

Define relationship between parameters and covariates

## Description

The Expression property is a string or cell array of strings, where each string represents the relationship between a parameter and one or more covariates. The Expression property denotes fixed effects with the prefix `theta`, and random effects with the prefix `eta`.

Each expression string must be in the form:

```
parameterName = relationship
```

This example of an expression string defines the relationship between a parameter (`volume`) and a covariate (`weight`), with fixed effects, but no random effects:

```
CovModelObj.Expression = {'volume = theta1 +  
theta2*weight'};
```

This table illustrates expression formats for some common parameter-covariate relationships.

Parameter-Covariate Relationship	Expression Format
Linear with random effect	$C1 = \text{theta1} + \text{theta2} * \text{WEIGHT} + \text{eta1}$
Exponential without random effect	$C1 = \exp(\text{theta}_{C1} + \text{theta}_{C1\_WT} * \text{WEIGHT})$
Exponential, WEIGHT centered by mean, has random effect	$C1 = \exp(\text{theta1} + \text{theta2} * (\text{WEIGHT} - \text{mean}(\text{WEIGHT}))) + \text{eta1}$

## Expression (CovariateModel) property

---

Parameter-Covariate Relationship	Expression Format
Exponential, log(WEIGHT), which is equivalent to power model	$C1 = \exp(\text{theta1} + \text{theta2} \cdot \log(\text{WEIGHT}) + \text{eta1})$
Exponential, dependent on WEIGHT and AGE, has random effect	$C1 = \exp(\text{theta1} + \text{theta2} \cdot \text{WEIGHT} + \text{theta3} \cdot \text{AGE} + \text{eta1})$

---

**Tip** To simultaneously fit data from multiple dose levels, use a `CovariateModel` object as an input argument to `sbionlmeFit`, and omit the random effect (`eta`) from the `Expression` property in the `CovariateModel` object.

---

The `Expression` property must meet the following requirements:

- The expression strings are valid MATLAB code.
- Each expression string is linear with a transformation.
- There is exactly one expression string for each parameter.
- In each expression string, a covariate is used in at most one term.
- In each expression string, there is at most one random effect (`eta`)
- Fixed effect (`theta`) and random effect (`eta`) names are unique within and across expression strings. That is, each covariate has its own fixed effect.

# Expression (CovariateModel) property

---

---

**Tip** Use the `getCovariateData` method to view the covariate data when writing equations for the Expression property of a `CovariateModel` object.

---

---

**Tip** Use the `verify` method to check that the Expression property of a `CovariateModel` object meets the conditions described previously.

---

## Characteristics

Applies to	Object: <code>CovariateModel</code>
Data type	String or cell array of strings
Data values	<code>parameterName = relationship</code>
Access	Read/write

## See Also

`CovariateModel` | `getCovariateData` | `verify`

## How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”



# Expression (AbstractKineticLaw, KineticLaw) property

---

## 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 kinetic law definition used by the reaction. The kinetic law being used is indicated by the property KineticLawName. You can configure Expression for user-defined kinetic laws, but not for built-in kinetic laws. Expression is read only for kinetic law objects.

---

**Note** If you set the Expression property to a reaction rate expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Kinetic Law Definition

The *kinetic law definition* provides a mechanism for applying a specific rate law to multiple reactions. It acts as a mapping template for the reaction rate. The kinetic law is defined by a mathematical expression, (defined in the property Expression), and includes 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 a kinetic law definition, the ReactionRate property of the reaction object shows the result of a mapping from the kinetic law definition. To determine ReactionRate, the species variables and parameter variables that participate in the reaction rate should be mapped in the kinetic law for the reaction. In this case, SimBiology software determines the ReactionRate by using the Expression property of the abstract kinetic law object, and by mapping SpeciesVariableNames to SpeciesVariables and ParameterVariableNames to ParameterVariables.

# Expression (AbstractKineticLaw, KineticLaw) property

---

For example, the kinetic law definition 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.

By applying the Henri-Michaelis-Menten kinetic law to a reaction  $A \rightarrow B$  with  $V_a$  mapping to  $V_m$ ,  $A$  mapping to  $S$ , and  $K_a$  mapping to  $K_m$ , 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 software contains some built-in kinetic laws. You can also define your own kinetic laws. To find the list of available kinetic laws, use the `sbiowhos -kineticlaw` command (`sbiowhos`). You can create a kinetic law definition with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

## Characteristics

Applies to	Objects: abstract kinetic law, kinetic law
Data type	char string
Data values	Defined by kinetic law definition
Access	Read-only in kinetic law object. Read/write in user-defined kinetic law.

# Expression (**AbstractKineticLaw**, **KineticLaw**) property

## Examples

### Example 1

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'});
```

## Expression (AbstractKineticLaw, KineticLaw) property

---

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

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

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

### Example 2

Example with Mass Action kinetics.

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

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

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');  
get(kineticlawObj, 'Expression')
```

MATLAB returns:

```
ans =
```

```
MassAction
```

- 3 Assign the rate constant for the reaction.

```
set (kineticlawObj, 'ParameterVariablenames', 'k');  
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

## Expression (**AbstractKineticLaw**, **KineticLaw**) property

---

$k*a*b$

### **See Also**

KineticLawName, Parameters, ParameterVariableNames,  
ParameterVariables, ReactionRate, sbioaddtolibrary, sbiowhos,  
SpeciesVariables, SpeciesVariableNames

# FixedEffectDescription (CovariateModel) property

---

**Purpose** Descriptions of fixed effects in CovariateModel object

**Description** The FixedEffectDescription property is a cell array of strings describing the fixed effects in the Expression property of a CovariateModel object. Each string describes the role of a fixed effect in the expression equation. For example, in the following expression equation:

$$C1 = \exp(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$$

The description for the fixed effect `theta1` is 'C1', which indicates it is the intercept for the parameter C1. Also, the description for the fixed effect `theta2` is 'C1/WEIGHT', which indicates it is the slope of the line that defines the relationship between the parameter C1 and the covariate WEIGHT.

## Characteristics

Applies to	Object: CovariateModel
Data type	Cell array of strings
Data values	Description of the roles of the fixed effects in the Expression property
Access	Read-only

**See Also** CovariateModel | Expression | FixedEffectNames | FixedEffectValues

# FixedEffectNames (CovariateModel) property

---

**Purpose** Names of fixed effects in CovariateModel object

**Description** The FixedEffectNames property is a cell array of strings specifying the names of the fixed effects in the Expression property of a CovariateModel object. Names of fixed effects are denoted with the prefix theta.

## Characteristics

Applies to	Object: CovariateModel
Data type	Cell array of strings
Data values	Names of the fixed effects in the Expression property. These name are denoted with the prefix theta
Access	Read-only

**See Also** CovariateModel | Expression | FixedEffectDescription | FixedEffectValues

# FixedEffectValues (CovariateModel) property

---

**Purpose** Values for initial estimates of fixed effects in CovariateModel object

**Description** The FixedEffectValues property is a structure containing one field for each fixed effect in the Expression property of a CovariateModel object. Each field contains the value of the initial estimate for a fixed effect.

---

**Tip** You must set this property before using the CovariateModel object as input to sbionlmeFit or sbionlmeFitsa. Use the constructDefaultFixedEffectValues method to create a structure of fixed-effect initial estimate values, set to a default of zero. Then edit the structure and use it to modify this property.

---

## Characteristics

Applies to	Object: CovariateModel
Data type	Structure with one field for each fixed effect
Data values	Each field contains a double specifying the value of the initial estimate for a fixed effect in the CovariateModel object
Access	Read/write

**See Also** CovariateModel | constructDefaultFixedEffectValues | Expression | FixedEffectDescription | FixedEffectNames

**How To**

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”



**Purpose** Integer identifying each group in data set

**Description** GroupID is a property of the PKData object. It is an array of the same length as the DataSet property containing an integer to identify each group. PKData sets this property during construction of the PKData object.

## Characteristics

Applies to	Object: PKData
Data type	double
Data values	Index value for each group
Access	Read-only

**See Also** “Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, PKData object

# GroupLabel property

---

**Purpose** Identify group column in data set

**Description** GroupLabel is a property of the PKData object. It specifies the column in DataSet that contains the group identification labels.

## Characteristics

Applies to	Object: PKData
Data type	char string
Data values	Column header string from imported data set
Access	Read/write

**See Also** “Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, PKData object, GroupNames

**Purpose** Unique values from GroupLabel in data set

**Description** GroupNames is a property of the PKData object. It contains unique values from the data column specified by the GroupLabel property. PKData sets this property during construction of the PKData object.

## Characteristics

Applies to	Object: PKData
Data type	char string or cell array of strings
Data values	Unique values in GroupLabel
Access	Read-only

**See Also** “Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, PKData object, GroupLabel

# HasLag property

---

**Purpose** Lag associated with dose targeting compartment

**Description** HasLag is a property of the PKCompartment object. It is a logical indicating if the dose targeting the compartment has a time lag or not.

## Characteristics

Applies to	Object: PKCompartment
Data type	logical
Data values	1 (true) or 0 (false). Default is 0 (false).
Access	Read/write

**See Also** addCompartment, DosingType, EliminationType, PKCompartment object

# HasResponseVariable property

---

**Purpose** Compartment drug concentration reported

**Description** HasResponseVariable is a property of the PKCompartment object. It is a logical indicating if the drug concentration in this compartment is reported.

---

**Note** The HasResponseVariable property can be true for more than one PKCompartment object in the model. If you perform a parameter fit on a model, at least one PKCompartment object in the model must have a HasResponseVariable property set to true.

---

## Characteristics

Applies to	Object: PKCompartment
Data type	Logical
Data values	1 (true) or 0 (false). Default is 0 (false).
Access	Read/write

**See Also** addCompartment, DosingType, EliminationType, PKCompartment object

# IndependentVarLabel property

---

**Purpose** Identify independent variable column in data set

**Description** IndependentVarLabel is a property of the PKData object. It specifies the column in DataSet that contains the independent variable (for example, time).

The column must contain positive values, and cannot contain, NaN, Inf or -Inf.

## Characteristics

Applies to	Object: PKData
Data type	char string
Data values	Column header from imported data set
Access	Read/write

**See Also** “Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, PKData object

**Purpose** Time units in PKData object

**Description** The IndependentVarUnits property indicates the units for the column containing the independent variable (time) in the PKData object. If unit conversion is on, plot results in the SimBiology desktop show the units specified in IndependentVarUnits.

To get a list of units, use the sbioshowunits function.

## Characteristics

Applies to	Object: PKData
Data type	string
Data values	Time units. Default is '' (empty).
Access	Read/write

**See Also** DependentVarLabel, PKData Object

# InitialAmount property

---

**Purpose** Species initial amount

**Description** The `InitialAmount` property indicates the initial quantity of the SimBiology species object. `InitialAmount` is the quantity of the species before the simulation starts.

## Characteristics

Applies to	Object: species
Data type	double
Data values	Positive real number. Default value is 0.
Access	Read/write

## Examples

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

**1** Create a model object named `my_model`.

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

**2** Add the species object named `glucose`.

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

**3** Set the initial amount to 100 and verify.

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

MATLAB returns:

```
ans =  
  
100
```

## See Also

`addspecies`, `InitialAmountUnits`



**Purpose** Species initial amount units

**Description** The `InitialAmountUnits` property indicates the unit definition for the `InitialAmount` property of a species object. `InitialAmountUnits` can be one of the built-in units. To get a list of the defined units, use the `sbioshowunits` function. If `InitialAmountUnits` changes from one unit definition to another, `InitialAmount` does not automatically convert to the new units. The `sbioconvertunits` function does this conversion. To add a user-defined unit to the list, use `sbiounit` followed by `sbioaddtolibrary`.

See `DefaultSpeciesDimension` for more information on specifying dimensions for species quantities. `InitialAmountUnits` must have corresponding dimensions to `CapacityUnits`. For example, if the `CapacityUnits` are `meter2`, then species must be `amount/meter2` or `amount`.

## Characteristics

Applies to	Object: species
Data type	char string
Data values	Units from library with dimensions of amount, amount/length, amount/area, or amount/volume. Default is '' (empty).
Access	Read/write

## Examples

**1** Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');  
compObj = addcompartment(modelObj, 'cell');
```

**2** Add a species object named `glucose`.

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

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

## InitialAmountUnits property

---

```
set (speciesObj, 'InitialAmountUnits', 'molecule');  
get (speciesObj, 'InitialAmountUnits')
```

MATLAB returns:

```
ans =
```

```
molecule
```

### See Also

DefaultSpeciesDimension, InitialAmount, sbioaddtolibrary,  
sbioconvertunits, sbioshowunits, sbiounit,

**Purpose** Specify species and parameter input factors for sensitivity analysis

**Description** Inputs is a property of the SensitivityAnalysisOptions object. SensitivityAnalysisOptions is a property of the configuration set object.

Use Inputs to specify the species and parameters with respect to which you want to compute the sensitivities of the species or parameter states in your model.

The SimBiology software calculates sensitivities with respect to the values of the parameters and the initial amounts of the species specified in the Inputs property. When you simulate a model with SensitivityAnalysis enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species and parameters specified in the Outputs property. For a description of the output, see the SensitivityAnalysisOptions property description.

## Characteristics

Applies to	Object: SensitivityAnalysisOptions
Data type	Species or parameter object or an array of objects

---

**Note** If this object is determined by a repeated assignment rule, then you cannot use it as an Inputs property.

---

Data values	Species or parameter object, or an array of objects. Default is [ ] (empty array).
Access	Read/write

**Examples** This example shows how to set Inputs for sensitivity analysis.

- 1 Import the radio decay model from the SimBiology demos.

# Inputs property

---

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

- 2 Retrieve the configuration set object from modelObj.

```
configsetObj = getConfigset(modelObj);
```

- 3 Add a parameter to the Inputs property and display it. Use the sbioselect function to retrieve the parameter object from the model.

SimBiology Parameter Array

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

## See Also

Outputs, sbioselect, SensitivityAnalysis, SensitivityAnalysisOptions

**Purpose** Time between doses

**Description** Interval is a property of a RepeatDose object. This property defines the equally spaced times between repeated doses.

---

**Note** When the Interval property is 0, RepeatDose ignores the RepeatCount property, that is, it treats it as though it is set to 0.

---

**Characteristics**

Applies to	Object: RepeatDose
Data type	double
Data values	Nonnegative real number. Default is 0
Access	Read/Write

**See Also** RepeatDose object, ScheduleDose object

# KineticLaw property

---

**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 you update the reaction, the ReactionRate property automatically updates only for mass action kinetics. For all other kinetics, you must set the SpeciesVariableNames property of the kinetic law object.

For information on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated”.

## Characteristics

Applies to	Object: reaction
Data type	Kinetic law object
Data values	Kinetic law object. Default is [] (empty).
Access	Read-only

## Examples

Example with Henri-Michaelis-Menten kinetics

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

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

**2** Define a kinetic law for the reaction object.

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

- 3 Verify that the KineticLaw property for the reaction object is Henri-Michaelis-Menten.

```
get (reactionObj, 'KineticLaw')
```

MATLAB returns:

```
SimBiology Kinetic Law Array
```

```
Index:    KineticLawName:  
    1      Henri-Michaelis-Menten
```

### See Also

KineticLawName, Parameters, ParameterVariableNames, ReactionRate, SpeciesVariableNames

# KineticLawName property

---

**Purpose** Name of kinetic law applied to reaction

**Description** The `KineticLawName` property of the kinetic law object indicates the name of the kinetic law definition applied to the reaction. `KineticLawName` can be any valid name from the built-in or user-defined kinetic law library. See “Kinetic Law Definition” on page 6-65 for more information.

You can find the `KineticLawName` list in the kinetic law library by using the command `sbiowhos -kineticlaw (sbiowhos)`. You can create a kinetic law definition with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

## Characteristics

Applies to	Object: <code>kineticlaw</code>
Data type	<code>char string</code>
Data values	<code>char string</code> specified by kinetic law definition
Access	Read-only

## Examples

- 1 Create a model object, add a reaction object, and define a kinetic law for the reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');  
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

- 2 Verify the `KineticLawName` of `kineticlawObj`.

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

MATLAB returns:

```
ans =
```

```
Henri-Michaelis-Menten
```



## See Also

Expression(AbstractKineticLaw, KineticLaw), Parameters, ParameterVariableNames, ParameterVariables, ReactionRate, sbioaddtolibrary, sbiowhos, SpeciesVariables, SpeciesVariableNames

# LagParameter property

---

**Purpose** Parameter specifying time lag for doses

**Description** LagParameter is a property of the PKModelMap object. It specifies the name(s) of parameter object(s) that represent the time lag(s) of doses associated with the PKModelMap object.

Specify the name(s) of parameter object(s) that are:

- Scoped to a model
- Constant, that is, their ConstantValue property is true

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the LagParameter property and the DosingType property. For a dose that has no lag, use '' (an empty string). For an example, see “Dosing Multiple Compartments in a Model”.

## Characteristics

Applies to	Object: PKModelMap
Data type	char string or cell array of strings
Data values	<u>Name(s) of parameter object(s) or empty.</u> <b>Tip</b> If you are not using any doses with time lags, you can set this property to a cell array of empty strings, or simply an empty cell array. The parameter objects must be:
Access	Read/write

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, DosingType, PKModelMap object

# LagParameterName property

---

**Purpose** Parameter specifying time lag for dose

**Description** LagParameterName is a property of a RepeatDose or ScheduleDose object.

Specify the name of a parameter object that is:

- Scoped to a model
- Constant, that is, its ConstantValue property is true

The parameter specifies the length of time it takes for the dose to reach its target after being introduced.

## Characteristics

Applies to Objects: RepeatDose, ScheduleDose

Data type char string

Data values Name of a parameter object or empty. Default is an empty string.

Access Read/write  
The parameter object must be:

**See Also** RepeatDose object, ScheduleDose object, ScopedModel object

- Constant, that is, have a ConstantValue property set to true

# LogDecimation property

---

**Purpose** Specify frequency to log stochastic simulation output

**Description** LogDecimation is a property of the SolverOptions property, which is a property of a configset object. This property defines how often stochastic simulation data is recorded. LogDecimation is available only for stochastic solvers (ssa, expltau, and impltau).

Use LogDecimation to specify how frequently you want to record the output of the simulation. For example, if you set LogDecimation to 1, for the command `[t,x] = sbiosimulate(modelObj)`, at each simulation step the time will be logged in `t` and the quantity of each logged species will be logged as a row in `x`. If LogDecimation is 10, then every 10th simulation step will be logged in `t` and `x`.

## Characteristics

Applies to	Object: SolverOptions
Data type	int
Data values	>0. Default is 1.
Access	Read/write

## Examples

This example shows how to change LogDecimation settings.

- 1 Retrieve the configset object from the modelObj, and change the SolverType to expltau.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);  
set(configsetObj, 'SolverType', 'expltau')
```

- 2 Change the LogDecimation to 10.

```
set(configsetObj.SolverOptions, 'LogDecimation', 10);  
get(configsetObj.SolverOptions, 'LogDecimation')
```

```
ans =
```

10

**See Also**      ErrorTolerance, RandomState

# MaximumNumberOfLogs property

---

**Purpose** Maximum number of logs criteria to stop simulation

**Description** MaximumNumberOfLogs is a property of a Configset object. This property sets the maximum number of logs criteria to stop a simulation. A simulation stops when it meets any of the criteria specified by StopTime, MaximumNumberOfLogs, or MaximumWallClock. However, if you specify the OutputTimes property of the SolverOptions property of the Configset object, then StopTime and MaximumNumberOfLogs are ignored. Instead, the last value in OutputTimes is used as the StopTime criteria, and the length of OutputTimes is used as the MaximumNumberOfLogs criteria.

<b>Characteristics</b>	Applies to	Object: Configset
	Data type	double
	Data values	Positive value. Default is Inf.
	Access	Read/write

## **Examples**      **Set Maximum Number of Logs Criteria to Stop Simulation**

Set the maximum number of logs that triggers a simulation to stop.

Create a model object named cell and save it in a variable named modelObj.

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

Retrieve the configuration set from modelObj and save it in a variable named configsetObj.

```
configsetObj = getconfigset(modelObj);
```

Configure the simulation stop criteria by setting the MaximumNumberOfLogs property to 50. Leave the StopTime and MaximumWallClock properties at their default values of 10 seconds and Inf, respectively.

# MaximumNumberOfLogs property

---

```
set(configsetObj, 'MaximumNumberOfLogs', 50)
```

View the properties of configsetObj.

```
get(configsetObj)
```

```
Active: 1
CompileOptions: [1x1 SimBiology.CompileOptions]
    Name: 'default'
    Notes: ''
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
SolverOptions: [1x1 SimBiology.ODESolverOptions]
    SolverType: 'ode15s'
    StopTime: 10
MaximumNumberOfLogs: 50
MaximumWallClock: Inf
    TimeUnits: 'second'
    Type: 'configset'
```

When you simulate modelObj, the simulation stops when 50 logs are created or when the simulation time reaches 10 seconds, whichever comes first.

## See Also

Configset object, MaximumWallClock, OutputTimes, StopTime

# MaximumWallClock property

---

**Purpose** Maximum elapsed wall clock time to stop simulation

**Description** MaximumWallClock is a property of a Configset object. This property sets the maximum elapsed wall clock time (seconds) criteria to stop a simulation.

A simulation stops when it meets any of the criteria specified by StopTime, MaximumNumberOfLogs, or MaximumWallClock. However, if you specify the OutputTimes property of the SolverOptions property of the Configset object, then StopTime and MaximumNumberOfLogs are ignored. Instead, the last value in OutputTimes is used as the StopTime criteria, and the length of OutputTimes is used as the MaximumNumberOfLogs criteria.

<b>Characteristics</b>	Applies to	Object: Configset
	Data type	double
	Data values	Positive scalar. Default is Inf.
	Access	Read/write

## **Examples**      **Set Maximum Wall Clock Criteria to Stop Simulation**

Set the maximum wall clock time (in seconds) that triggers a simulation to stop.

Create a model object named cell and save it in a variable named modelObj.

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

Retrieve the configuration set from modelObj and save it in a variable named configsetObj.

```
configsetObj = getconfigset(modelObj);
```



# MaximumWallClock property

---

Configure the simulation stop criteria by setting the `MaximumWallClock` property to 20 seconds. Leave the `StopTime` and `MaximumNumberOfLogs` properties at their default values of 10 seconds and `Inf`, respectively.

```
set(configsetObj, 'MaximumWallClock', 20)
```

View the properties of `configsetObj`.

```
get(configsetObj)
```

```
Active: 1
CompileOptions: [1x1 SimBiology.CompileOptions]
Name: 'default'
Notes: ''
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
SolverOptions: [1x1 SimBiology.ODESolverOptions]
SolverType: 'ode15s'
StopTime: 10
MaximumNumberOfLogs: Inf
MaximumWallClock: 20
TimeUnits: 'second'
Type: 'configset'
```

When you simulate `modelObj`, the simulation stops when the simulation time reaches 10 seconds or the wall clock time reaches 20 seconds, whichever comes first.

## See Also

`Configset` object, `MaximumNumberOfLogs`, `OutputTimes`, `StopTime`

# MaxIterations property

---

**Purpose** Specify nonlinear solver maximum iterations in implicit tau

**Description** The `MaxIterations` property specifies the maximum number of iterations for the nonlinear solver in `impltau`. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object.

The implicit tau solver in SimBiology software internally uses a nonlinear solver to solve a set of algebraic nonlinear equations at every simulation step. Starting with an initial guess at the solution, the nonlinear solver iteratively tries to find the solution to the algebraic equations. The closer the initial guess is to the solution, the fewer the iterations the nonlinear solver will take before it finds a solution. `MaxIterations` specifies the maximum number of iterations the nonlinear solver should take before it issues a “failed to converge” error. If you get this error during simulation, try increasing `MaxIterations`. The default value of `MaxIterations` is 15.

## Characteristics

Applies to	Object: <code>SolverOptions</code>
Data type	<code>int</code>
Data values	>0. Default is 15.
Access	Read/write

## Examples

This example shows how to change `MaxIterations` settings.

- 1 Retrieve the `configset` object from the `modelObj`, and change the `SolverType` to `impltau`.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);  
set(configsetObj, 'SolverType', 'impltau');
```

- 2 Change the `MaxIterations` to 25.

```
set(configsetObj.SolverOptions, 'MaxIterations', 25);
```

```
get(configsetObj.SolverOptions, 'MaxIterations')
```

```
ans =
```

```
25
```

### **See Also**

ErrorTolerance, LogDecimation, RandomState

# MaxStep property

---

**Purpose** Specify upper bound on ODE solver step size

**Description** MaxStep is a property of the SolverOptions property, which is a property of a configset object. This property specifies the bounds on the size of the time steps. MaxStep is available only for ODE solvers (ode15s, ode23t, ode45, and sundials).

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 or empty
Data values	Default value is [] (empty), which is equivalent to setting MaxStep to infinity.
Access	Read/write

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

**Purpose** Name of model simulated

**Description** The ModelName property shows the name of the model for which the SimData object contains the simulation data.

## Characteristics

Applies to	Object: SimData
Data type	string
Data values	Default value is '' (empty).
Access	Read-only

**See Also** Data, DataInfo

# Models property

---

**Purpose** Contain all model objects

**Description** The `Models` property shows the models in the SimBiology root. It is a read-only array of model objects.

SimBiology has a hierarchical organization. A model object has the SimBiology root as its `Parent`. 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')`.

## Characteristics

Applies to	Objects: root
Data type	Array of model objects
Data values	Model object. Default is [] (empty).
Access	Read-only

**See Also** `sbiomodel`

**Purpose** Relationship between defined unit and base unit

**Description** The `Multiplier` is the numerical value that defines the relationship between the unit `Name` and the base unit as a product of the `Multiplier` and the base unit. For example, in `Celsius = (5/9)*( Fahrenheit-32)`; `Multiplier` is 5/9 and `Offset` is -32. For `1 mole = 6.0221e23*molecule`, the `Multiplier` is 6.0221e23.

## Characteristics

Applies to	Object: Unit
Data type	double
Data values	Nonzero real number. Default value is 1.
Access	Read/write

## Examples

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

- 1 Create a user-defined unit called `usermole`, whose composition is `molecule` and `Multiplier` property is 6.0221e23.

```
unitObj = sbiounit('usermole', 'molecule', 6.0221e23);
```

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

```
sbioaddtolibrary(unitObj);
```

- 3 Query the `Multiplier` property.

```
get(unitObj, 'Multiplier')
```

```
ans =
```

```
1/molarity*second
```

## See Also

`Composition`, `get`, `Offset`, `sbiounit`, `set`

# Name property

---

**Purpose** Specify name of object

**Description** The Name property identifies a SimBiology object. Compartments, species, parameters, and model objects can be referenced by other objects using the Name property, therefore Name must be unique for these objects. However, species names need only be unique within each compartment. Parameter names must be unique within a model (if at the model level), or within each kinetic law (if at the kinetic law level). This means that you can have nonunique species names if the species are in different compartments, and nonunique parameter names if the parameters are in different kinetic laws or at different levels. Note that having nonunique parameter names can cause the model to have shadowed parameters and that may not be best modeling practice. For more information on levels of parameters, see “Scope of Parameter Objects”.

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

In addition, note the following constraints and reserved characters for the Name property in objects:

- Model names cannot be empty.
- Parameter names cannot be empty, or have the name `time`.
- If you have a parameter, a species, or compartment name that is not a valid MATLAB variable name, when you write an event function, an event trigger, a reaction, reaction rate equation, or a rule you must enclose that name in brackets. For example, enclose `[DNA polymerase+]` in brackets. In addition, if you have the same species in multiple compartments you must qualify the species with the compartment name, for example, `nucleus.[DNA polymerase+]`, `[nuclear complex].[DNA polymerase+]`.
- Species and compartment names cannot be empty, and note the following reserved words, characters, and constraints:



- The literal words `null` and `time`. Note that you can specify species names with these words contained within the name. For example, `nullaminoacids` or `nullnucleotides`.
- The characters `->`, `<`, `>`, `[`, and `]`.

For more information on valid MATLAB variable names, see `genvarname` and `isvarname`.

## Characteristics

Applies to	Objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, unit, or variant
Data type	char string
Data values	Any char string except reserved words and characters
Access	Read/write

## Examples

- 1 Create a model object named `my_model`.

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

- 2 Add a reaction object to the model object. Note the use of brackets because the names are not valid MATLAB variable names.

```
reactionObj = addreaction(modelObj, '[Aspartic acid] -> [beta-Asparty1-P04]')
```

MATLAB returns:

```
SimBiology Reaction Array
```

```
Index:   Reaction:
      1   [Aspartic acid] -> [beta-Asparty1-P04]
```

- 3 Set the reaction Name and verify.

## Name property

---

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

MATLAB returns:

```
ans =
```

```
Aspartate kinase reaction
```

### See Also

addcompartment, addkineticlaw, addparameter, addreaction,  
addrule, addspecies, RepeatDose object, sbiomodel, sbiunit,  
sbiunitprefix, ScheduleDose object

## Purpose

Specify normalization type for sensitivity analysis

## Description

Normalization is a property of the `SensitivityAnalysisOptions` object. `SensitivityAnalysisOptions` is a property of the configuration set object. Use `Normalization` to specify the normalization for the computed sensitivities.

The following values let you specify the type of normalization. The examples show you how sensitivities of a species  $x$  with respect to a parameter  $k$  are calculated for each normalization type:

- 'None' specifies no normalization.

$$\frac{dx(t)}{dk}$$

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

$$\left( \frac{1}{x(t)} \right) \left( \frac{dx(t)}{dk} \right)$$

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

$$\left( \frac{k}{x(t)} \right) \left( \frac{dx(t)}{dk} \right)$$

## Characteristics

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

## See Also

Inputs, Outputs, `SensitivityAnalysis`, `SensitivityAnalysisOptions`

# Notes property

---

**Purpose** HTML text describing SimBiology object

**Description** Use the Notes property of an object to store comments about the object. You can include HTML tagging in the notes to render formatted text in the SimBiology desktop.

## Characteristics

Applies to	Objects: compartment, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, unit, or unit prefix
Data type	char string
Data values	Any char string
Access	Read/write

## Examples

**1** Create a model object.

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

**2** Write notes for the model object.

```
set (modelObj, 'notes', '09/01/05 experimental data')
```

**3** Verify the assignment.

```
get (modelObj, 'notes')
```

MATLAB returns:

```
ans =
```

```
09/01/05 experimental data
```

## See Also

addkineticlaw, addparameter, addreaction, addrule, addspecies, RepeatDose object, sbiomodel, sbiounit, sbiounitprefix, ScheduleDose object

**Purpose** Measured response object name

**Description** Observed is a property of the PKModelMap object. It specifies the name(s) of one or more objects that represent the measured response (the response variable). Specify the name(s) of species or parameter object(s) that are scoped to a model.

## Characteristics

Applies to	Object: PKModelMap
Data type	char string or cell array of strings
Data values	Name of a species or parameter object or empty. Default is an empty cell array.
Access	Read/write

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, Dosed, Estimated, PKModelMap object

# Offset property

---

**Purpose** Unit composition modifier

## Description

---

**Note** The Offset property is currently not supported.

---

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

## Characteristics

Applies to	Object: Unit
Data type	double
Data values	Real number. Default is 0.
Access	Read/write

## Examples

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

- 1 Create a user-defined unit called `celsius2`, whose composition refers to `fahrenheit`, Multiplier property is 9/5, and Offset property is 32.

```
unitObj = sbiounit('celsius2','fahrenheit',9/5,32);
```

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

```
sbioaddtolibrary(unitObj);
```

- 3 Query the Offset property.

```
get(unitObj, 'Offset')
```

```
ans =
```

32

### **See Also**

Composition, get, Multiplier, sbioaddtolibrary, sbioshowunits, sbiounit, set

# Outputs property

---

**Purpose** Specify species and parameter outputs for sensitivity analysis

**Description** Outputs is a property of the SensitivityAnalysisOptions object. SensitivityAnalysisOptions is a property of the configuration set object.

Use Outputs to specify the species and parameters for which you want to compute sensitivities.

The SimBiology software calculates sensitivities with respect to the values of the parameters and the initial amounts of the species specified in the Inputs property. When you simulate a model with SensitivityAnalysis enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species and parameters specified in Outputs. For a description of the output, see the SensitivityAnalysisOptions property description.

## Characteristics

Applies to	Object: SensitivityAnalysisOptions
Data type	Species or parameter object or array of objects

---

**Note** If a species or parameter object is determined by a repeated assignment rule, then you cannot use it as an Outputs property.

---

Data values	Species or parameter object, or an array of objects. Default is [] (empty array).
Access	Read/write

**Examples** This example shows how to set Outputs for sensitivity analysis.

1 Import the radio decay model from the SimBiology demos.



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

- 2 Retrieve the configuration set object from modelObj.

```
configsetObj = getConfigset(modelObj);
```

- 3 Add a species to the Outputs property and display it. Use the sbioselect function to retrieve the species object from the model.

SimBiology Species Array

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

### See Also

Inputs, sbioselect, SensitivityAnalysis, SensitivityAnalysisOptions

# OutputTimes property

---

**Purpose** Specify times to log deterministic simulation output

**Description** OutputTimes is a property of the SolverOptions property, which is a property of a Configset object. This property specifies the times during a deterministic (ODE) simulation that data is recorded. Time units are specified by the TimeUnits property of the Configset object. OutputTimes is available only for ODE solvers (ode15s, ode23t, ode45, and sundials).

If the criteria set in the MaximumWallClock property causes a simulation to stop before all time values in OutputTimes are reached, then no data is recorded for the latter time values.

The OutputTimes property can also control when a simulation stops:

- The last value in OutputTimes overrides the StopTime property as criteria for stopping a simulation.
- The length of OutputTimes overrides the MaximumNumberOfLogs property as criteria for stopping a simulation.

<b>Characteristics</b>	Applies to	Object: SolverOptions
	Data type	double
	Data values	Vector of nonnegative, monotonically increasing values, or [], an empty vector. Default is [], which results in data being logged every time the simulation solver takes a step.
	Access	Read/write

## **Examples** Specify Times to Log Deterministic Simulation Output

Specify the times during a deterministic (ODE) simulation that data is recorded.

Create a model object named cell and save it in a variable named modelObj.

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

Retrieve the configuration set from `modelObj` and save it in a variable named `configsetObj`.

```
configsetObj = getconfigset(modelObj);
```

Specify to log output every second for the first 10 seconds of the simulation. Do this by setting the `OutputTimes` property of the `SolverOptions` property of `ConfigsetObj`.

```
set(configsetObj.SolverOptions, 'OutputTimes', [1:10])  
get(configsetObj.SolverOptions, 'OutputTimes')
```

```
ans =
```

```
1 2 3 4 5 6 7 8 9 10
```

When you simulate `modelObj`, output is logged every second for the first 10 seconds of the simulation. Also, the simulation stops after the 10th log.

### See Also

`MaximumNumberOfLogs`, `MaximumWallClock`, `SolverOptions`, `StopTime`, `TimeUnits`

# Owner property

---

**Purpose**           Owning compartment

**Description**       Owner shows you the SimBiology compartment object that owns the compartment object. In the compartment object, the `Owner` property shows you whether the compartment resides within another compartment. The `Compartments` property indicates whether other compartments reside within the compartment. You can add a compartment object using the method `addcompartment`.

## Characteristics

Applies to	Object: compartment
Data type	char string
Data values	Name of compartment object. Default is [].
Access	Read-only

## Examples

**1** Create a model object named `modelObj`.

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

**2** Add two compartments to the model object.

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

**3** Add a compartment to one of the compartment objects.

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

**4** Display the `Owner` property in the compartment objects.

```
get(compartmentObj3, 'Owner')
```

The result shows you the owning compartment and its components:

```
SimBiology Compartment - mitochondrion
```

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

**5** Change the owning compartment.

```
set(compartmentObj3, 'Owner', compartmentObj1)
```

### **See Also**

Compartments, Parent

# ParameterNames (CovariateModel) property

---

**Purpose** Names of parameters in CovariateModel object

**Description** The ParameterNames property is a cell array of strings specifying the names of the parameters in the Expression property of a CovariateModel object.

## Characteristics

Applies to	Object: CovariateModel
Data type	Cell array of strings
Data values	Names of the parameters in the Expression property
Access	Read only

**See Also** CovariateModel | Expression

**Purpose** Array of parameter objects

**Description** The `Parameters` property indicates the parameters in a `Model` or `KineticLaw` object. Read-only array of `Parameter` objects.

The scope of a parameter object is hierarchical and is defined by the parameter's parent. If a parameter is defined with a kinetic law object as its parent, then only the kinetic law object can use the parameter. If a parameter object is defined with a model object as its parent, then components such as rules, events, and kinetic laws (reaction rate equations) can use the parameter.

You can add a parameter to a model object, or kinetic law object with the method `addparameter` and delete it with the method `delete`.

You can view parameter object properties with the `get` command and configure properties with the `set` command.

## Characteristics

Applies to	Objects: <code>model</code> , <code>kineticlaw</code>
Data type	Array of parameter objects
Data values	Parameter objects. Default value is <code>[]</code> (empty).
Access	Read-only

## Examples

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

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

**2** Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

**3** Add a parameter and assign it to the kinetic law object (`kineticlawObj`);

```
parameterObj1 = addparameter (kineticlawObj, 'K1');
```

## Parameters property

---

```
get (kineticlawObj, 'Parameters')
```

```
SimBiology Parameter Array
```

Index:	Name:	Value:	ValueUnits:
1	K1	1	

- 4** Add a parameter and assign it to the model object (modelObj);.

```
parameterObj1 = addparameter(modelObj, 'K2');
```

```
get(modelObj, 'Parameters')
```

```
SimBiology Parameter Array
```

Index:	Name:	Value:	ValueUnits:
1	K2	1	

### See Also

addparameter, delete, get, set



# ParameterVariableNames property

---

**Purpose** Cell array of reaction rate parameters

**Description** The `ParameterVariableNames` property shows the parameters used by the kinetic law object to determine the `ReactionRate` equation in the reaction object. Use `setParameter` to assign `ParameterVariableNames`. When you assign species to `ParameterVariableNames`, SimBiology software maps these parameter names to `ParameterVariables` in the kinetic law object.

If the reaction is using a kinetic law, the `ReactionRate` property of a reaction object shows the result of a mapping from a “Kinetic Law Definition” on page 6-65. 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

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

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

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

- 2 Create a kinetic law object for the reaction object of type 'Henri-Michaelis-Menten'.

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

```
reactionObj.KineticLaw property is configured to kineticlawObj.
```

## ParameterVariableNames property

---

- 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

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariables, ReactionRate, setparameter,  
SpeciesVariables, SpeciesVariableNames

**Purpose** Parameters in kinetic law definition

**Description** The `ParameterVariables` property shows the parameter variables that are used in the `Expression` property of the abstract kinetic law object. Use this property to specify the parameters in the `ReactionRate` equation. Use the method `set` to assign `ParameterVariables` to a kinetic law definition. For more information, see “Kinetic Law Definition” on page 6-65.

## Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Specified by kinetic law definition
Access	Read/write in kinetic law definition. Read-only in kinetic law.

## Examples

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

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

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

- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

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

reactionObj `KineticLaw` property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables. To set these variables:

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

# ParameterVariables property

---

MATLAB returns:

```
ans =
```

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

## See Also

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariableNames, ReactionRate, set, setparameter,  
SpeciesVariables, SpeciesVariableNames

**Purpose** Indicate parent object

**Description** The Parent property indicates the parent object for a SimBiology object (read-only). The Parent property indicates accessibility of the object. The object is accessible to the Parent object and other objects within the Parent object. The value of Parent depends on the type of object and how it was created. All models always have the SimBiology root as the Parent.

### More Information

The following table shows you the different objects and the possible Parent value.

Object	Parent
abstract kinetic law	<ul style="list-style-type: none"> <li>• [] (empty) until added to library</li> <li>• root object upon addition to library</li> </ul>
compartment	model object
event	model object or [] (empty)
kinetic law	reaction object
model	root object
parameter	model object, kinetic law object, or [] (empty)
reaction	model object or [] (empty)
RepeatDose	model object or [] (empty)
rule	model object or [] (empty)
ScheduleDose	model object or [] (empty)
species	compartment

# Parent property

---

Object	Parent
variant	model object or [] (empty)
unit and unit prefixes	<ul style="list-style-type: none"><li>• [] (empty) until added to library</li><li>• root object upon addition to library</li></ul>

## Characteristics

Applies to	Objects: abstract kinetic law, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, variant, unit, or unit prefix
Data type	Object
Data values	SimBiology component object or [] (empty)
Access	Read-only

## See Also

addkineticlaw, addparameter, addreaction, RepeatDose object, sbiomodel, ScheduleDose object

**Purpose** Hold compartments in PK model

**Description** PKCompartments is a property of the PKModelDesign object. It is used to specify the compartments in the PKModelDesign object. Each compartment is a PKCompartment object added using the addCompartment method.

## Characteristics

Applies to	Objects: PKModelDesign
Data type	object
Data values	PKCompartment object
Access	Read-only

**See Also** “Creating Pharmacokinetic Models” in the SimBiology User’s Guide, addCompartment, PKCompartment object, PKModelDesign object

# Products property

---

**Purpose** Array of reaction products

**Description** The Products property contains an array of `SimBiology.Species` objects.

Products is a 1-by-n species object array that indicates the species that are changed by the reaction. If the Reaction property is modified to use a different species, the Products property is updated accordingly.

You can add product species to the reaction with `addproduct` function. You can remove product species from the reaction with `rmproduct`. You can also update reaction products by setting the Reaction property with the function `set`.

## Characteristics

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

## Examples

**1** Create a model object.

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

**2** Add reaction objects.

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

**3** Verify the assignment.

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

MATLAB returns:

```
SimBiology Species Array
```

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



1	unnamed	c	0
2	unnamed	d	0

**See Also** `addkineticlaw, addproduct, addspecies, rmproduct`

# RandomEffectNames (CovariateModel) property

---

**Purpose** Names of random effects in CovariateModel object

**Description** The RandomEffectNames property is a cell array of strings specifying the names of the random effects in the Expression property of a CovariateModel object. Names of random effects are denoted with the prefix eta.

## Characteristics

Applies to	Object: CovariateModel
Data type	Cell array of strings
Data values	Names of the random effects in the Expression property. These names are denoted with the prefix eta.
Access	Read only

**See Also** CovariateModel | Expression

**Purpose** Set random number generator

**Description** The `RandomState` property sets the random number generator for the stochastic solvers. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object.

`SimBiology` software uses a pseudorandom number generator. The sequence of numbers generated is determined by the state of the generator, which can be specified by the integer `RandomState`. If `RandomState` is set to integer  $J$ , the random number generator is initialized to its  $J^{\text{th}}$  state. The random number generator can generate all the floating-point numbers in the closed interval  $[2^{(-53)}, 1 - 2^{(-53)}]$ . Theoretically, it can generate over  $2^{1492}$  values before repeating itself. But for a given state, the sequence of numbers generated will be the same. To change the sequence, change `RandomState`. `SimBiology` software resets the state at startup. The default value of `RandomState` is `[]`.

## Characteristics

Applies to	Objects: <code>SolverOptions</code> for SSA, <code>expltau</code> , <code>impltau</code>
Data type	<code>int</code>
Data values	Default is <code>[]</code> (empty).
Access	Read/write

**Examples** This example shows how to change `RandomState` settings.

- 1 Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `expltau`.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);  
set(configsetObj, 'SolverType', 'expltau')
```

- 2 Change the `Randomstate` to 5.

## RandomState property

---

```
set(configsetObj.SolverOptions, 'RandomState', 5);  
get(configsetObj.SolverOptions, 'RandomState')
```

```
ans =
```

```
5
```

### **See Also**

ErrorTolerance, LogDecimation, MaxIterations

**Purpose** Rate of dose

**Description** Rate is a property of a RepeatDose or ScheduleDose object.  
This property defines how fast a dose is given.

---

**Note** If you set the Rate property of a dose, you must also specify the Amount property of the dose, and set the DurationParameterName property to ''. This is because the duration is calculated from the amount and rate.

---

## Characteristics

Applies to	Objects: RepeatDose, ScheduleDose
Data type	double (RepeatDose) or double array (ScheduleDose)
Data values	Nonnegative real number. Default is 0 (RepeatDose) or [ ] (ScheduleDose).
Access	Read/write

**See Also** RepeatDose object, ScheduleDose object

# RateUnits property

---

**Purpose** Units for dose rate

**Description** RateUnits is a property of a PKData, RepeatDose or ScheduleDose object.

- In RepeatDose or ScheduleDose objects, this property defines units for the Rate property.
- In PKData object, this property defines units for the RateLabel property.

## Characteristics

Applies to	Object: RepeatDose, ScheduleDose, PKData
Data type	string
Data values	Units from library with dimensions of amount divided by time. You cannot use units of concentration divided by time. Default = "
Access	Read/write

**See Also** PKData object, ScheduleDose object, RepeatDose object, Rate, RateLabel

**Purpose** Rate of infusion column in data set

**Description** RateLabel is a property of the PKData object. It specifies the column in DataSet that contains the rate of infusion. This applies only when dosing type is infusion. The data set must contain the rate and not an infusion time. The values must be positive and the column cannot contain Inf or -Inf. 0 specifies an infinite rate (equivalent to a bolus dose), and NaN specifies no rate.

## Characteristics

Applies to	Objects: PKData
Data type	char string
Data values	Column header string
Access	Read/write

**See Also** “Specifying and Classifying the Data to Fit” in the SimBiology User’s Guide, PKData object, DosingType

# Reactants property

---

**Purpose** Array of reaction reactants

**Description** The `Reactants` property is a 1-by-n species object array of reactants in the reaction. If the `Reaction` property is modified to use a different reactant, the `Reactants` property will be updated accordingly.

You can add reactant species to the reaction with the `addreactant` method.

You can remove reactant species from the reaction with the `rmreactant` method. You can also update reactants by setting the `Reaction` property with the function `set`.

## Characteristics

Applies to	Object: reaction
Data type	Species object or array of species objects
Data values	Species objects. Default is [] (empty).
Access	Read-only

## Examples

**1** Create a model object.

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

**2** Add reaction objects.

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

**3** View the reactants for `reactionObj`.

```
get(reactionObj, 'Reactants')
```

MATLAB returns:

SimBiology Species Array

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



2          unnamed          b          0

### See Also

addreactant, addreaction, addspecies, rmreactant

# Reaction property

---

**Purpose** Reaction object reaction

**Description** Property to indicate the reaction represented in the reaction object. Indicates the chemical reaction that can change the amount of one or more species, for example, 'A + B > C'. This property is different from the model object property called Reactions.

See `addreaction` for more information on how the Reaction property is set.

## Characteristics

Applies to	Object: reaction
Data type	char string
Data values	Valid reaction string. Default is '' (empty).
Access	Read/write

## Examples

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

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

**2** Verify that the reaction property records the input.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =
```

```
a + b -> c + d
```

**See Also** `addreaction`

**Purpose** Reaction rate equation in reaction object

**Description** The ReactionRate property defines the reaction rate equation. You can define a ReactionRate with or without the KineticLaw property. KineticLaw defines the type of reaction rate. The addkineticlaw function configures the ReactionRate based on the KineticLaw and the species and parameters specified in the kinetic law object properties SpeciesVariableNames and ParameterVariableNames.

The reaction takes place in the reverse direction if the Reversible property is true. This is reflected in ReactionRate. The ReactionRate includes the forward and reverse rate if reversible.

You can specify ReactionRate without KineticLaw. Use the set function to specify the reaction rate equation. SimBiology software adds species variables while creating reactionObj using the addreaction method. You must add the parameter variables (to the modelObj in this case). See the example below.

After you specify the ReactionRate without KineticLaw and you later configure the reactionObj to use KineticLaw, the ReactionRate is unset until you specify SpeciesVariableNames and ParameterVariableNames.

For information on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated” .

---

**Note** If you set the ReactionRate property to an expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Characteristics

Applies to	Object: reaction
Data type	char string

# ReactionRate property

---

Data values	Reaction rate string. Default is '' (empty).
Access	Read/write

## Examples

### Example 1

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

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

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

- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

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

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that you should set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm\_d and Km\_d and assign them to kineticlawObj.

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

- 4 Set the variable names for the kinetic law object.

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

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

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

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

## Example 2

Create a model, add a reaction, and specify ReactionRate without a kinetic law.

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

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

- 2 Specify ReactionRate and verify the assignment.

```
set (reactionObj, 'ReactionRate', 'k*a');  
get(reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
k*a
```

- 3 You cannot simulate the model until you add the parameter k to the modelObj.

```
parameterObj = addparameter(modelObj, 'k');
```

SimBiology adds the parameter to the modelObj with default Value = 1.0 for the parameter.

## See Also

addparameter, addreaction, Reversible

# Reactions property

---

**Purpose** Array of reaction objects

**Description** Property to indicate the reactions in a Model object. Read-only array of reaction objects.

A reaction object defines a chemical reaction that occurs between species. The species for the reaction are defined in the Model object property Species.

You can add a reaction to a model object with the method `addreaction`, and you can remove a reaction from the model object with the method `delete`.

## Characteristics

Applies to	Object: model
Data type	Array of reaction objects
Data values	Reaction object
Access	Read-only

## Examples

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

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

**2** Verify that the reactions property records the input.

```
get (modelObj, 'Reactions')
```

MATLAB returns:

SimBiology Reaction Array

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

**See Also** `addreaction`, `delete`

**Purpose** Allowable error tolerance relative to state value during a simulation

**Description** `RelativeTolerance` is a property of the `SolverOptions` object, which is a property of a `Configset` object. It is available for the ode solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`).

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, amounts for all the species.

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.

**Algorithm** At each simulation step, the solver estimates the local error  $e_i$  in the  $i$ th state vector  $y$ . Simulation converges at that time step if  $e_i$  satisfies the following equation:

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

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: <code>SolverOptions</code>
Data type	<code>double</code>

# RelativeTolerance property

---

Data values	Positive scalar that is <1. Default is 1e-3.
Access	Read/write

## Examples

This example shows how to change AbsoluteTolerance.

**1** Retrieve the configset object from the modelObj.

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

**2** Change the AbsoluteTolerance to 1e-8.

```
set(configsetObj.SolverOptions, 'RelativeTolerance', 1.0e-6);  
get(configsetObj.SolverOptions, 'RelativeTolerance')
```

```
ans =
```

```
1.0000e-006
```

## See Also

AbsoluteTolerance



**Purpose** Dose repetitions

**Description** RepeatCount is a property of a RepeatDose object. This property defines the number of doses after the initial dose in a repeat dose series.

---

**Note** When the Interval property is 0, RepeatDose ignores the RepeatCount property, that is, it treats it as though it is set to 0.

---

## Characteristics

Applies to	Object: RepeatDose
Data type	double
Data values	Nonnegative integer. Default is 0
Access	Read/Write

**See Also** ScheduleDose object and RepeatDose object

# Reversible property

---

**Purpose** Specify whether reaction is reversible or irreversible

**Description** The `Reversible` property defines whether a reaction is reversible or irreversible. The rate of the reaction is defined by the `ReactionRate` property. For a reversible reaction, the reaction rate equation is the sum of the rate of the forward and reverse reactions. The type of reaction rate is defined by the `KineticLaw` property. If a reaction is changed from reversible to irreversible or vice versa after `KineticLaw` is assigned, the new `ReactionRate` is determined only if `Type` is `MassAction`. All other `Types` result in unchanged `ReactionRate`. For `MassAction`, the first parameter specified is assumed to be the rate of the forward reaction.

## Characteristics

Applies to	Object: reaction
Data type	boolean
Data values	true, false. Default value is false.
Access	Read/write

## Examples

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

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

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

- 2 Set the `Reversible` property for the `reactionObj` to true and verify this setting.

```
set (reactionObj, 'Reversible', true)  
get (reactionObj, 'Reversible')
```

MATLAB returns:

```
ans =
```

1

MATLAB returns 1 for true and 0 for false.

In the next steps the example illustrates how the reaction rate equation is assigned for reversible reactions.

- 3 Create a kinetic law object for the reaction object of the type 'MassAction'.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

reactionObj KineticLaw property is configured to kineticlawObj.

- 4 The 'MassAction' kinetic law for reversible reactions has two parameter variables ('Forward Rate Parameter' and 'Reverse Rate Parameter') that you should set. The species variables for MassAction are automatically determined. To set the parameter variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) named Kf and Kr and assign the object to kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Kf');  
parameterObj2 = addparameter(kineticlawObj, 'Kr');
```

- 5 Set the variable names for the kinetic law object.

```
set(kineticlawObj, 'ParameterVariableNames', {'Kf' 'Kr'});
```

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

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
Kf*a*b - Kr*c*d
```

## Reversible property

---

### **See Also**

addparameter, addreactant, addreaction, 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`.

---

**Note** If you set the Rule property for an algebraic rule, rate rule, or repeated assignment rule, and the rule expression is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Characteristics

Applies to	Object: rule
Data type	char string
Data values	char string defined as species or parameter objects. Default is empty.
Access	Read/write

## Examples

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

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

**2** Add a rule.

```
ruleObj = addrule(modelObj, '10-a+b')
```

## Rule property

---

MATLAB returns:

SimBiology Rule Array

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

### See Also

addrule, delete

## Purpose

Specify type of rule for rule object

## Description

The `RuleType` property indicates the type of rule defined by the rule object. A Rule object defines how certain species, parameters, and compartments should interact with one another. For example, a rule could state that the total number of species A and species B must be some value. `Rule` is a MATLAB expression that defines the change in the species object quantity or a parameter object `Value` when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule`.

The types of rules in SimBiology are as follows:

- `initialAssignment` — Lets you specify the initial value of a parameter, species, or compartment capacity, as a function of other model component values in the model.
- `repeatedAssignment` — Lets you specify a value that holds at all times during simulation, and is a function of other model component values in the model.
- `algebraic` — Lets you specify mathematical constraints on one or more parameters, species, or compartments that must hold during a simulation.
- `rate` — Lets you specify the time derivative of a parameter value, species amount, or compartment capacity.

### Constraints on Varying Species Using a Rate Rule

If the model has a species defined in concentration, being varied by a rate rule, and it is in a compartment with varying volume, you can only use `rate` or `initialAssignment` rules to vary the compartment volume.

Conversely, if you are varying a compartment's volume using a `repeatedAssignment` or `algebraic` rules, then you cannot vary a species (defined in concentration) within that compartment, with a rate rule.

# RuleType property

---

The reason for these constraints is that, if a species is defined in concentration and it is in a compartment with varying volume, the time derivative of that species is a function of the compartment's rate of change. For compartments varied by rate rules, the solver has that information.

Note that if you specify the species in amounts there are no constraints.

## Characteristics

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

## Examples

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

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

- 2 Add a rule that specifies the quantity of a species c. In the rule expression, k is the rate constant for a -> b.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

- 3 Change the RuleType from the default ('algebraic') to 'rate' and verify it using the get command.

```
set(ruleObj, 'RuleType', 'rate');  
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
Active: 1  
Annotation: ''  
Name: ''
```



```
Notes: ''
Parent: [1x1 SimBiology.Model]
Rule: 'c = k*(a+b) '
RuleType: 'rate'
Tag: ''
Type: 'rule'
UserData: []
```

**See Also** “Rule Object” in the *SimBiology User’s Guide*, `addrule`, `delete`

# Rules property

---

**Purpose** Array of rules in model object

**Description** The Rules property shows the rules in a Model object. Read-only array of SimBiology.Rule objects.

A *rule* is a mathematical expression that modifies a species amount or a parameter value. A rule defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species A and species B must be some value.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule` and `RuleType`.

## Characteristics

Applies to	Object: model
Data type	Array of rule objects
Data values	Rule object
Access	Read-only

## Examples

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

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

**2** Add a rule.

```
ruleObj = addrule(modelObj, '10-a+b')
```

MATLAB returns:

SimBiology Rule Array

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

## See Also

`addrule`, `delete`

**Purpose** Information about simulation

**Description** The RunInfo property contains information describing the simulation run that yielded the data in the SimData object.

The following information is stored:

- **Configset** — A struct form of the configuration set used during simulation. This would typically be the model's active configset.
- **Variant** — A struct form of the variant(s) used during simulation.
- **SimulationDate** — The date/time of simulation.
- **SimulationType** — Either 'single run' or 'ensemble run', depending on whether the data object was created using the function sbiosimulate or the function sbioensemblerrun.

## Characteristics

Applies to

Object: SimData

Data type

struct

Data values

Default values are as follows:

```
ConfigSet: []  
SimulationDate: ''  
SimulationType: ''  
Variant: []
```

In practice, the ConfigSet, SimulationDate, and SimulationType fields are rarely empty, since they are populated after simulation.

Access

Read-only

**See Also** StopTime, StopTimeType

# RuntimeOptions property

---

**Purpose** Options for logged species

**Description** The RuntimeOptions property holds options for species that will be logged during the simulation run. The run-time options object can be accessed through this property.

The LogDecimation property of the configuration set object defines how often data is logged.

**Property Summary**

StatesToLog	Specify species, compartment, or parameter data recorded
Type	Display SimBiology object type

## Characteristics

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

## Examples

**1** Create a model object, and retrieve its configuration set.

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

**2** Retrieve the RuntimeOptions object from the configset object.

```
runtimeObj = get(configsetObj, 'RunTimeOptions')  
Runtime Settings:
```

```
StatesToLog: all
```

**See Also** get, set

## Purpose

Enable or disable sensitivity analysis

## Description

`SensitivityAnalysis` is a property of the `SolverOptions` property, which is a property of a `configset` object. This property lets you compute the time-dependent sensitivities of all the species states defined by the `StatesToLog` property with respect to the `Inputs` that you specify in the `SensitivityAnalysisOptions` property of the configuration set object.

`SensitivityAnalysis` is available only for the ODE solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`)

---

**Note** Models containing the following active components do not support sensitivity analysis:

- Nonconstant compartments
- Algebraic rules
- Events

---

For more information on setting up sensitivity analysis, see `SensitivityAnalysisOptions`. For a description of sensitivity analysis calculations, see “Sensitivity Calculation”.

## Characteristics

Applies to	Object: <code>SolverOptions</code>
Data type	logical
Data values	1, 0, true, false. Default is false.
Access	Read/write

# SensitivityAnalysis property

---

## 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 property

---

**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 supported only for deterministic (ODE) simulations.

---

**Note** The SensitivityAnalysisOptions property controls the settings related to sensitivity analysis. To enable or disable sensitivity analysis, use the SensitivityAnalysis property.

---

Properties of SensitivityAnalysisOptions are summarized in “Property Summary” on page 6-160.

When sensitivity analysis is enabled, the following command

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

returns [t,x,names], where

- **t** is an  $n$ -by-1 vector, where  $n$  is the number of steps taken by the ode solver and **t** defines the time steps of the solver.
- **x** is an  $n$ -by- $m$  matrix, where  $n$  is the number of steps taken by the ode solver and  $m$  is:

Number of species and parameters specified in StatesToLog +  
(Number of sensitivity outputs \* Number of sensitivity input factors)

A SimBiology state includes species and nonconstant parameters.

- **names** is the list of states logged and the list of sensitivities of the species specified in StatesToLog with respect to the input factors.

For an example of the output, see “Examples” on page 6-160.

You can add a number of configuration set objects with different SensitivityAnalysisOptions to the model object with the

# SensitivityAnalysisOptions property

---

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

Inputs	Specify species and parameter input factors for sensitivity analysis
Normalization	Specify normalization type for sensitivity analysis
Outputs	Specify species and parameter outputs for sensitivity analysis

## Characteristics

Applies to	Object: configuration set
Data type	Object
Data values	SensitivityAnalysisOptions properties as summarized in “Property Summary” on page 6-160.
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 configuration settings and the sensitivity analysis options from modelObj.

```
configsetObj = getconfigset(modelObj);  
sensitivityObj = get(configsetObj, 'SensitivityAnalysisOptions');
```



# SensitivityAnalysisOptions property

---

**3** Add a species and a parameter to the Inputs property. Use the `sbioselect` function to retrieve the species and parameter objects from the model.

**4** Add a species to the Outputs property and display.

```
SimBiology Species Array
```

Index:	Compartment:	Name:	InitialAmount:	InitialAmount:
1	unnamed	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” on page 6-159 for more information.

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

**7** Display the names.

```
names
```

```
names =
```

```
'x'  
'z'  
'd[z]/d[z]_0'  
'd[z]/d[Reaction1.c]'
```

Display state values `x`.

```
x
```

## SensitivityAnalysisOptions property

---

The display follows the column order shown in `names` for the values in `x`. The rows correspond to `t`.

**See Also** `addconfigset`, `getconfigset`, `SensitivityAnalysis`

**Purpose** Specify model solver options

**Description** The SolverOptions property is an object that holds the model solver options in the configset object. Changing the property SolverType changes the options specified in the SolverOptions object.

Properties of SolverOptions are summarized in “Property Summary” on page 6-163.

## Property Summary

AbsoluteTolerance	Absolute error tolerance applied to state value during simulation
AbsoluteToleranceScaling	Control scaling of absolute error tolerance during simulation
AbsoluteToleranceStepSize	Initial guess for time step size for scaling of absolute error tolerance
ErrorTolerance	Specify explicit or implicit tau error tolerance
LogDecimation	Specify frequency to log stochastic simulation output
MaxIterations	Specify nonlinear solver maximum iterations in implicit tau
MaxStep	Specify upper bound on ODE solver step size
OutputTimes	Specify times to log deterministic simulation output
RandomState	Set random number generator
RelativeTolerance	Allowable error tolerance relative to state value during a simulation

# SolverOptions property

---

SensitivityAnalysis	Enable or disable sensitivity analysis
Type	Display 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

## Examples

This example shows the changes in SolverOptions for various SolverType settings.

- 1 Retrieve the configset object from the modelObj.

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

- 2 Configure the SolverType to ode45.

```
set(configsetObj, 'SolverType', 'ode45');  
get(configsetObj, 'SolverOptions')
```

```
Solver Settings: (ode)
```

```
AbsoluteTolerance: 1.000000e-006  
RelativeTolerance: 1.000000e-003
```

- 3 Configure the SolverType to ssa.

```
set(configsetObj, 'SolverType', 'ssa');  
get(configsetObj, 'SolverOptions')
```

Solver Settings: (ssa)

```
LogDecimation:      1
RandomState:        []
```

#### 4 Configure the SolverType to impltau.

```
set(configsetObj, 'SolverType', 'impltau');
get(configsetObj, 'SolverOptions')
```

Solver Settings: (impltau)

```
ErrorTolerance:     3.000000e-002
LogDecimation:      1
AbsoluteTolerance:  1.000000e-002
RelativeTolerance:  1.000000e-002
MaxIterations:      15
RandomState:        []
```

#### 5 Configure the SolverType to expltau.

```
set(configsetObj, 'SolverType', 'expltau');
get(configsetObj, 'SolverOptions')
```

Solver Settings: (expltau)

```
ErrorTolerance:     3.000000e-002
LogDecimation:      1
RandomState:        []
```

### See Also

`addconfigset`, `getconfigset`

# SolverType property

---

**Purpose** Select solver type for simulation

**Description** The SolverType property lets you specify the solver to use for a simulation. For a discussion about solver types, see “Choosing a Simulation 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 value you assigned.

## Characteristics

Applies to	Object: Configset
Data type	enum
Data values	'ode15s', 'ode23t', 'ode45', 'sundials', 'ssa', 'expltau', 'impltau'. Default is 'ode15s'.

---

**Note** If your model contains events, you cannot specify 'expltau' or 'impltau' for the SolverType property.

---

---

**Note** If your model contains doses, you cannot specify 'ssa', 'expltau', or 'impltau' for the SolverType property.

---

Access	Read/write
--------	------------

## Examples

1 Retrieve the configset object from the modelObj.

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

```
configsetObj = getconfigset(modelObj)
```

```
Configuration Settings - default (active)
```

```
SolverType:           ode15s  
StopTime:             10.000000
```

```
SolverOptions:  
AbsoluteTolerance:    1.000000e-006  
RelativeTolerance:    1.000000e-003  
SensitivityAnalysis:  false
```

```
RuntimeOptions:  
StatesToLog:          all
```

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

```
SensitivityAnalysisOptions:  
Inputs:               0  
Outputs:              0
```

## 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  
SensitivityAnalysis:  false
```

## SolverType property

---

```
RuntimeOptions:  
  StatesToLog:          all  
  
CompileOptions:  
  UnitConversion:      false  
  DimensionalAnalysis: true  
  
SensitivityAnalysisOptions:  
  Inputs:              0  
  Outputs:             0
```

**See Also**      `getConfigset`, `set`



**Purpose** Array of species in compartment object

**Description** The `Species` property is a property of the compartment object and indicates all the species in a compartment object. `Species` is a read-only array of SimBiology species objects.

In the model object, `Species` contains a flat list of all the species that exist within all the compartments in the model. You should always access a species through its compartment rather than the model object. Use the format `compartmentName.speciesName`, for example, `nucleus.DNA`. Another example of the syntax is `modelObj.Compartments(2).Species(1)`. The `Species` property in the model object might not be available in a future version of the software.

Species are entities that take part in reactions. A species object is added to the `Species` property when a reaction is added to the model object with the method `addreaction`. A species object can also be added to the `Species` property with the method `addspecies`.

If you remove a reaction with the method `delete`, and a species is no longer being used by any of the remaining reactions, the species object is *not* removed from the `Species` property. You have to use the `delete` method to remove species.

There are reserved characters that cannot be used in species object names. See `Name` for more information.

## Characteristics

Applies to	Object: compartment
Data type	Array of species objects
Data values	Species object. Default is [ ] (empty).
Access	Read-only

**See Also** `addcompartment`, `addreaction`, `addspecies`, `delete`

# SpeciesVariableNames property

---

**Purpose** Cell array of species in reaction rate equation

**Description** The SpeciesVariableNames property shows the species used by the kinetic law object to determine the ReactionRate equation in the reaction object. Use setspecies to assign SpeciesVariableNames. When you assign species to SpeciesVariableNames, SimBiology software maps these species names to SpeciesVariables in the kinetic law object.

The ReactionRate property of a reaction object shows the result of a mapping from kinetic law definition. The ReactionRate is determined by the kinetic law object Expression property by mapping ParameterVariableNames to ParameterVariables and SpeciesVariableNames to SpeciesVariables.

## Characteristics

Applies to	Object: kinetic law
Data type	Cell array of strings
Data values	Cell array of species names
Access	Read/write

## Examples

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

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

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

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'

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

The reactionObj KineticLaw property is configured to kineticlawObj.

## SpeciesVariableNames property

---

- 3** The 'Henri-Michaelis-Menten' kinetic law has one species variable (S) that you should set. To set this variable:

```
setspecies(kineticlawObj, 'S', 'a');
```

- 4** Verify that the species variable is correct.

```
get (kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =
```

```
'a'
```

### See Also

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariables, ParameterVariableNames, ReactionRate,  
setParameter, SpeciesVariables

# SpeciesVariables property

---

**Purpose** Species in abstract kinetic law

**Description** This property shows species variables that are used in the Expression property of the kinetic law object to determine the ReactionRate equation in the reaction object. Use the MATLAB function `set` to assign `SpeciesVariables` to an abstract kinetic law. For more information, see abstract kinetic law.

## Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Defined by abstract kinetic law
Access	Read/write in abstract kinetic law. Read-only in kinetic law.

## Examples

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

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

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

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');  
  
reactionObj.KineticLaw property is configured to kineticlawObj.
```

- 3 View the species variable for 'Henri-Michaelis-Menten' kinetic law.

```
get(kineticlawObj, 'SpeciesVariables')
```

MATLAB returns:

```
ans =  
  
'S'
```

### See Also

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariables, ParameterVariableNames, ReactionRate, set,  
setparameter, SpeciesVariableNames

# StartTime property

---

**Purpose** Start time for initial dose time

**Description** StartTime is a property of a RepeatDose object. For a series of repeated doses, the StartTime property defines the amount of time that elapses before the first (initial) dose is given.

**Characteristics**

Applies to	Objects: RepeatDose
Data type	double
Data values	Nonnegative real number. Default value is 0
Access	Read-write

**See Also** RepeatDose object

**Purpose** Specify species, compartment, or parameter data recorded

**Description** The StatesToLog property specifies the species, compartment, or parameter data to log during a simulation. This is the data returned in `x` during execution of `[t,x] = sbiosimulate(modelObj)`. By default, all species, nonconstant compartments, and nonconstant parameters are logged.

## Characteristics

Applies to	Object: RuntimeOptions
Data type	Object or vector of objects
Data values	Species objects, compartment objects, or parameter objects. Default is <code>all</code> , which means all species objects, all compartment objects whose <code>ConstantCapacity = false</code> , and all parameter objects whose <code>ConstantValue = false</code> .
Access	Read/write

**Examples** This example shows how to assign species to StatesToLog.

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

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

- 2 Retrieve the first and second species in `modelObj`.

```
speciesObj1 = modelObj.Species(1);  
speciesObj2 = modelObj.Species(2);
```

- 3 Retrieve the `configsetObj` of `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

- 4 Set the StatesToLog to record three species: two using the retrieved species objects and one using indexing and view the species in StatesToLog.

## StatesToLog property

---

```
set (configsetObj.RuntimeOptions, 'StatesToLog', ...  
    [speciesObj1, speciesObj2, modelObj.Species(3)]);  
get(configsetObj.RuntimeOptions, 'StatesToLog')
```

- 5 Set the StatesToLog property back to the default setting of all.

```
set (configsetObj.RuntimeOptions, 'StatesToLog', 'all');
```



## Purpose

Species coefficients in reaction

## Description

The **Stoichiometry** property specifies the species coefficients in a reaction. Enter an array of **doubles** indicating the stoichiometry of reactants (negative value) and products (positive value). Example: [-1 -1 2].

The **double** specified cannot be 0. The reactants of the reaction are defined with a negative number. The products of the reaction are defined with a positive number. For example, the reaction  $3\text{H} + \text{A} \rightarrow 2\text{C} + \text{F}$  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** function. Add a product or reactant and set stoichiometry with methods **addproduct** and **addreactant**.

ODE solvers support **double** stoichiometry values such as 0.5. Stochastic solvers and dimensional analysis currently support only integers in **Stoichiometry**, therefore you must balance the reaction equation and specify integer values for these two cases.

$\text{A} \rightarrow \text{null}$  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 is [ ] (empty).
Access	Read/write

# Stoichiometry property

---

## Examples

- 1 Create a reaction object.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, '2 a + 3 b -> d + 2 c');
```

- 2 Verify the Reaction and Stoichiometry properties for reactionObj.

```
get(reactionObj, 'Stoichiometry')
```

MATLAB returns:

```
ans =
```

```
-2    -3     1     2
```

- 3 Set stoichiometry to [-1 -2 2 2].

```
set (reactionObj, 'Stoichiometry', [-1 -2 2 2]);  
get (reactionObj, 'Stoichiometry')
```

MATLAB returns:

```
ans =
```

```
-1    -2     2     2
```

- 4 Note with get that the Reaction property updates automatically.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =
```

```
a + 2 b -> 2 d + 2 c
```

## See Also

addproduct, addreactant, addreaction, Reaction, rmproduct, rmreactant

**Purpose** Simulation time criteria to stop simulation

**Description** StopTime is a property of a Configset object. This property sets the maximum simulation time criteria to stop a simulation. Time units are specified by the TimeUnits property of the Configset object.

A simulation stops when it meets any of the criteria specified by StopTime, MaximumNumberOfLogs, or MaximumWallClock. However, if you specify the OutputTimes property of the SolverOptions property of the Configset object, then StopTime and MaximumNumberOfLogs are ignored. Instead, the last value in OutputTimes is used as the StopTime criteria, and the length of OutputTimes is used as the MaximumNumberOfLogs criteria.

<b>Characteristics</b>	Applies to	Object: Configset
	Data type	double
	Data values	Nonnegative scalar. Default is 10.
	Access	Read/write

## Examples **Set Simulation Time Criteria to Stop Simulation**

- 1 Create a model object named cell and save it in a variable named modelObj. Retrieve the configuration set from modelObj and save it in a variable named configsetObj.

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

- 2 Configure the simulation stop criteria by setting the StopTime property to 20 seconds. Leave the MaximumNumberOfLogs and MaximumWallClock properties at their default values of Inf.

```
set(configsetObj, 'StopTime', 20)  
get(configsetObj)
```

## StopTime property

---

```
Active: 1
CompileOptions: [1x1 SimBiology.CompileOptions]
    Name: 'default'
    Notes: ''
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
SolverOptions: [1x1 SimBiology.ODESolverOptions]
    SolverType: 'ode15s'
    StopTime: 20
MaximumNumberOfLogs: Inf
MaximumWallClock: Inf
TimeUnits: 'second'
Type: 'configset'
```

When you simulate `modelObj`, the simulation stops when the simulation time reaches 20 seconds.

### See Also

`Configset` object, `MaximumNumberOfLogs`, `MaximumWallClock`, `OutputTimes`, `TimeUnits`

**Purpose** Specify type of stop time for simulation

---

**Note** StopTimeType will be removed in a future release. Use StopTime,MaximumNumberOfLogs, MaximumWallClock and OutputTimes instead.

---

**Description** StopTimeType is a property of a Configset object. 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

## StopTimeType property

---

### **See Also**

set, StatesToLog, StopTime, TimeUnits

**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 `sbiobject` 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 `sbiobject`. 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.

## Characteristics

Applies to	Objects: abstract kinetic law, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species
Data type	char string
Data values	Any char string
Access	Read/write

## Examples

**1** Create a model object.

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

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

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

**3** Verify the Tag assignment.

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

MATLAB returns:

```
ans =
```

## Tag property

---

'Synthesis Reaction'

### **See Also**

addkineticlaw, addparameter, addreaction, addrule, addspecies,  
RepeatDose object, sbioabstractkineticlaw, sbiomodel, sbioroot,  
ScheduleDose object



**Purpose** Species receiving dose

**Description** TargetName is a property of a RepeatDose or ScheduleDose object. This property defines the SimBiology species receiving the dose. The dose amount increases the species amount at each time interval defined by a repeat dose or at each time point defined by a schedule dose.

The value of TargetName is the name of a species. If the model has more than one species with the same name, TargetName is defined as *compartmentName.speciesName*, where *compartmentName* is the name of the compartment containing the species.

## Characteristics

Applies to	Objects: RepeatDose, ScheduleDose
Data type	string
Data values	Species name. Default value is "" (empty).
Access	Read/Write

**See Also** ScheduleDose object and RepeatDose object

# Trigger property

---

**Purpose** Event trigger

**Description** Trigger is a property of an Event object

A Trigger is a condition that must become true for an event to execute. You can use a combination of relational and logical operators to build a trigger expression. Trigger can be a string, an expression, or a function handle that when evaluated returns a value of `true` or `false`. A Trigger can access species, parameters, and compartments.

A trigger can contain the keyword `time` and relational operators to trigger an event that occurs at a specific time during the simulation. For example, `time >= x`. In this example trigger, note that:

- The units associated with the keyword `time` are the units for the `TimeUnits` property for the `Configset` object associated with the simulation.
- If `x` is an expression containing compartments, species, or parameters, then any units associated with the expression must have the same dimensions as the keyword `time`.
- If `x` is a raw number, then its dimensions (and units, if unit conversion is on) are assumed to be the same as the keyword `time`.

For more information about how the SimBiology software handles events, see “How Events Are Evaluated”. For examples of event functions, see “Specifying Event Triggers”.

<b>Characteristics</b>	Applies to	Object: event
	SimBiology type	String, function handle
	SimBiology values	Specify a MATLAB expression as a string. Default is '' (empty string).
	Access	Read/write

## Examples

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

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

- 2 Set the Trigger property of the event object.

```
set(eventObj, 'Trigger', '(time >=5) && (speciesA<1000)');
```

- 3 Get the Trigger property.

```
get(eventObj, 'Trigger')
```

## See Also

Event object, EventFcns

# Time property

---

**Purpose** Simulation time steps or schedule dose times

**Description** Time is a property of a SimData or ScheduleDose object.

## **SimData Object**

For a simulation, the Time property records the time steps.

## **ScheduleDose Object**

For a series of scheduled doses, the Time property defines the times to give a dose.

A ScheduleDose object defines a series of doses. Each dose can have a different amount, as defined by an amount array in the Amount property, and given at specified times, as defined by a time array in the Time property. A rate array in the Rate property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

## **Characteristics**

Applies to	Objects: SimData, ScheduleDose
Data type	double (SimData), double array (ScheduleDose)
Data values	Vector of doubles (SimData)Array of nonnegative real numbers. Default value is [] (ScheduleDose)
Access	Read-only

**See Also** ScheduleDose object, SimData object, StopTime, StopTimeType

**Purpose** Show time units for dosing and simulation

**Description** The TimeUnits property specifies time units for these properties:

- StopTime property of a Configset object
- OutputTimes and AbsoluteToleranceStepSize properties of the SolverOptions property of a Configset object
- StartTime and Interval properties of a RepeatDose object
- Time property of a ScheduleDose object
- Time property of a SimData object

---

**Note** If you change the value of the TimeUnits property, make sure:

- You update the values of the Time, StartTime, Interval, StopTime, and OutputTimes properties accordingly.
  - You update raw numbers used in any event triggers that use the keyword `time` accordingly. For more information, see [Trigger](#).
  - The units, if any, associated with expressions used in any event triggers that use the keyword `time`, are consistent with the updated TimeUnits property. For more information, see [Trigger](#).
- 

**Characteristics**

Applies to	Objects: Configset, RepeatDose, ScheduleDose, SimData
Data type	string

## TimeUnits property

---

Data values	<p>Empty string or a string specifying any unit defined in the Units Library.</p> <p>Default value is:</p> <ul style="list-style-type: none"><li>• <code>second</code> — properties of a <code>Configset</code> object or <code>SimData</code> object for a <code>model</code> object created using <code>sbiomodel</code></li><li>• <code>hour</code> — properties of a <code>Configset</code> object or <code>SimData</code> object for a <code>model</code> object created from a <code>PKModelDesign</code> object</li><li>• <code>''</code> (empty string) — properties of <code>RepeatDose</code> and <code>ScheduleDose</code> objects</li></ul>
Access	<p>Read/write for properties of <code>Configset</code>, <code>RepeatDose</code>, and <code>ScheduleDose</code> objects</p> <p>Read only for properties of <code>SimData</code> objects</p>

### See Also

`Configset` object, `RepeatDose` object, `ScheduleDose` object, `SimData` object, `Interval`, `OutputTimes`, `StartTime`, `StopTime`, `Time`

**Purpose** Display SimBiology object type

**Description** The Type property indicates a SimBiology object type. When you create a SimBiology object, the value of Type is automatically defined.

For example, when a Species object is created, the value of the Type property is automatically defined as 'species'.

**Characteristics**

Applies to	Objects: abstract kinetic law, compartment, configuration set, CompileOptions, event, kinetic law, model, parameter, reaction, RepeatDose, root, rule, ScheduleDose, species, RuntimeOptions, SolverOptions, unit, unitprefix, and variant
Data type	char string
Data values	abstract_kinetic_law, compartment, configset, compileoptions, event, kineticlaw, parameter, reaction, repeatdose, root, rule, runtimeoptions, sbiomodel, scheduledose, species, solveroptions, unit, unitprefix, and variant
Access	Read-only

**See Also** RepeatDose object, sbiomodel, sbioroot, ScheduleDose object, setactiveconfigset

# UnitConversion property

---

**Purpose** Perform unit conversion

**Description** The `UnitConversion` property specifies whether to perform unit conversion for the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object.

When `UnitConversion` is set to `true`, the SimBiology software converts the matching physical quantities to one consistent unit system in order to resolve them. This conversion is in preparation for correct simulation, but species amounts are returned in the user-specified units.

For example, consider a reaction  $a + b \rightarrow c$ . 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,  $1 / ((\text{molecules/liter}) \cdot \text{second})$ , `UnitConversion` occurs after `DimensionalAnalysis`.

Unit conversion requires dimensional analysis. If `DimensionalAnalysis` is off, and you turn `UnitConversion` on, then `DimensionalAnalysis` is turned on automatically. If `UnitConversion` is on and you turn off `DimensionalAnalysis`, then `UnitConversion` is turned off automatically.

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

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

Unit conversion involving temperature supports `Celsius` as the temperature unit. Avoid using mixed temperature units as you might get an error.



## Characteristics

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

## Examples

This example shows how to retrieve and set `unitconversion` from the default `true` to `false` in the default configuration set in a model object.

- 1 Import a model.

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

```
SimBiology Model - Oscillator
```

```
Model Components:
```

```
Models:          0
Parameters:      0
Reactions:       42
Rules:           0
Species:         23
```

- 2 Retrieve the `configset` object of the model object.

```
configsetObj = getconfigset(modelObj)
```

```
Configuration Settings - default (active)
```

```
SolverType:      ode15s
StopTime:        10.000000
```

```
SolverOptions:
```

```
AbsoluteTolerance: 1.000000e-006
RelativeTolerance: 1.000000e-003
```

# UnitConversion property

---

```
RuntimeOptions:
  StatesToLog:      all

CompileOptions:
  UnitConversion:   false
  DimensionalAnalysis: true
```

**3** Retrieve the CompileOptions object.

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

Compile Settings:

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

**4** Assign a value of false to UnitConversion.

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

## See Also

get, getconfigset, sbiosimulate, set

**Purpose** Specify data to associate with object

**Description** Property to specify data that you want to associate with a SimBiology object. The object does not use this data directly, but you can access it using the function `get` or dot notation.

## Characteristics

Applies to	Objects: abstract kinetic law, configuration set, compartment, data, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, or unit
Data type	Any
Data values	Any. Default is empty.
Access	Read/write

**See Also** RepeatDose object, sbioabstractkineticlaw, sbiomodel, sbioroot, sbiounit, sbiounitprefix, ScheduleDose object

# UserDefinedLibrary property

---

**Purpose** Library of user-defined components

**Description** `UserDefinedLibrary` is a SimBiology root object property containing all user-defined components of unit, unit prefixes, and kinetic laws that you define. You can add, modify, or delete components in the user-defined library. The `UserDefinedLibrary` property is an object that contains the following properties:

- **Units** — Contains any user-defined units. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined units either by using the command `sbiowhos -userdefined -unit`, or by accessing the root object.
- **UnitPrefixes** — Contains any user-defined unit prefixes. You can specify unit prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined unit prefixes either by using the command `sbiowhos -userdefined -unitprefix`, or by accessing the root object.
- **KineticLaws** — Contains any user-defined kinetic laws. Use the command `sbiowhos -userdefined -kineticlaw` to see the list of user-defined kinetic laws. You can use user-defined kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, `kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');`

See “Kinetic Law Definition” on page 6-65 for a definition and more information.

## Characteristics

Applies to	Object: root
Data type	object

# UserDefinedLibrary property

---

Data values	Unit, unit prefix, and abstract kinetic law objects
Access	Read-only

Characteristics for UserDefinedLibrary properties:

- **Units**

Applies to	UserDefinedLibrary property
Data type	Unit objects
Data values	Units
Access	Read/write

- **UnitPrefixes**

Applies to	UserDefinedLibrary property
Data type	Unit prefix objects
Data values	Unit prefixes
Access	Read/write

- **KineticLaws**

Applies to	UserDefinedLibrary property
Data type	Abstract kinetic law object
Data values	Kinetic laws
Access	Read/write

# UserDefinedLibrary property

---

## Examples

### Example 1

This example uses the command `sbiowhos` to show the current list of user-defined components.

```
sbiowhos -userdefined -kineticlaw
sbiowhos -userdefined -unit
sbiowhos -userdefined -unitprefix
```

### Example 2

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

```
rootObj = sbioroot;
get(rootObj.UserDefinedLibrary, 'KineticLaws')
get(rootObj.UserDefinedLibrary, 'Units')
get(rootObj.UserDefinedLibrary, 'UnitPrefixes')
```

## See Also

`BuiltInLibrary`, `sbioaddtolibrary`, `sbioremovefromlibrary`, `sbioroot`, `sbiounit`, `sbiounitprefix`

**Purpose** Assign value to parameter object

**Description** The Value property is the value of the parameter object. The parameter object defines an assignment that can be used by the model object and/or the kinetic law object. Create parameters and assign Value using the method `addparameter`.

## Characteristics

Applies to	Object: parameter
Data type	double
Data values	Any double. Default value is 1.0.
Access	Read/write

## Examples

Assign a parameter with a value to the model object.

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

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

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

```
parameterObj1 = addparameter (modelObj, 'K1', 0.5)
```

MATLAB returns:

SimBiology Parameter Array

Index:	Name:	Value:	ValueUnits:
1	K1	0.5	

## See Also

`addparameter`

# ValueUnits property

---

**Purpose** Parameter value units

**Description** The ValueUnits property indicates the unit definition of the parameter object Value property. ValueUnits can be one of the built-in units. To get a list of the built-in units, use the `sbioshowunits` function. If ValueUnits changes from one unit definition to another, the Value does not automatically convert to the new units. The `sbioconvertunits` function does this conversion.

You can add a parameter object to a model object or a kinetic law object.

## Characteristics

Applies to	Object: parameter
Data type	char string
Data values	Unit from units library. Default is '' (empty string). Note that the default value of an empty string means unspecified. Unspecified units are permitted during dimensional analysis, but not during unit conversion. (Use the string 'dimensionless' to specify dimensionless units.)
Access	Read/write

## Examples

Assign a parameter with a value to the model object.

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

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

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

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

MATLAB returns:



SimBiology Parameter Array

Index:	Name:	Value:	ValueUnits:
1	K1	0.5	1/second

### See Also

`addparameter`, `sbioconvertunits`, `sbioshowunits`

# ZeroOrderDurationParameter property

---

**Purpose** Zero-order dose absorption duration

**Description** ZeroOrderDurationParameter is a property of the PKModelMap object. It specifies the name(s) of parameter object(s) that represent the duration of absorption when the DosingType property is ZeroOrder.

Specify the name(s) of parameter object(s) that are:

- Scoped to a model
- Constant, that is, their ConstantValue property is true

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the ZeroOrderDurationParameter property and the DosingType property. For a dose that is not dosed with zero-order kinetics, use '' (an empty string). For an example, see “Dosing Multiple Compartments in a Model”.

## Characteristics

Applies to Object: PKModelMap

Data type char string or cell array of strings

---

**Tip** If you are not using any zero-order doses, you can set this property to a cell array of empty strings, or simply an empty cell array.

---

# ZeroOrderDurationParameter property

---

Data values	Name of a parameter object or empty. Default is an empty cell array. The parameter object(s) must be: <ul style="list-style-type: none"><li>• Scoped to a model</li><li>• Constant, that is, have a <code>ConstantValue</code> property set to <code>true</code></li></ul>
Access	Read/write

## See Also

“Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, `DosingType`, `PKModelMap` object

# ZeroOrderDurationParameter

---

## A

AbsoluteTolerance property  
reference 6-2

AbsoluteToleranceScaling property  
reference 6-4

AbsoluteToleranceStepSize property  
reference 6-5

abstract kinetic law object  
reference 4-4

Active property  
reference 6-6

addcompartment method  
reference 4-6

addCompartment method  
reference 4-11

addconfigset method  
reference 4-15

addcontent method  
reference 4-19

adddose method  
reference 4-21

addevent method  
reference 4-23

addkineticlaw method  
reference 4-26

addparameter method  
reference 4-34

addproduct method  
reference 4-39

addreactant method  
reference 4-42

addreaction method  
reference 4-45

addrule method  
reference 4-52

addspecies method  
reference 4-57

addvariant method  
reference 4-63

Amount property  
reference 6-8

AmountUnits property  
reference 6-9

## B

BoundaryCondition property  
reference 6-12

BuiltInLibrary property  
reference 6-15

## C

Capacity property  
reference 6-18

CapacityUnits property  
reference 6-19

commit method  
reference 4-65

compartment object  
reference 4-67

Compartments property  
reference 6-21

CompileOptions property  
reference 6-23

Composition property 6-25

configset object  
reference 4-71

Conserved Moieties  
function for 2-12

ConstantAmount property  
reference 6-27

ConstantCapacity property  
reference 6-29

ConstantValue property  
reference 6-30

construct method  
reference 4-74

constructDefaultFixedEffectValues method  
reference 4-76

Content property  
reference 6-32

copyobj method  
reference 4-77

CovariateLabels property  
reference 6-34

CovariateModel object  
reference 4-79

## D

Data property  
reference 6-36

DataCount property  
reference 6-37

DataInfo property  
reference 6-38

DataNames property  
reference 6-40

DataSet property  
reference 6-41

DefaultSpeciesDimension property  
reference 6-42

delete method  
reference 4-83

DependentVarLabel property  
reference 6-44

DependentVarUnits property  
reference 6-45

DimensionalAnalysis property  
reference 6-46

display method  
reference 4-85

Dosed property  
reference 6-49

DoseLabel property  
reference 6-50

DoseUnits property  
reference 6-51

DosingType property

reference 6-52

DurationParameterName property  
reference 6-53

## E

EliminationType property  
reference 6-54

Ensemble Runs  
function for 2-25 2-27 2-31

ErrorTolerance property  
reference 6-55

Estimated property  
reference 6-57

event object  
reference 4-86

Exponent property  
reference 6-61

Expression property  
reference 6-65

## F

functions

sbioabstractkineticlaw 2-2

sbioaccelerate 2-6

sbioaddtolibrary 2-9

sbioconsmoiety 2-12

sbioconvertunits 2-17

sbiocopylibrary 2-19

sbiodesktop 2-21

sbiodose 2-23

sbioensembleplot 2-25

sbioensemblerrun 2-27

sbioensemblestats 2-31

sbiogetmodel 2-38

sbio alasterror 2-41

sbio alastwarning 2-45

sbio loadproject 2-46

sbio model 2-47

sbionlinfit 2-51  
sbionlmefit 2-58  
sbionlmefitsa 2-66  
sbioparamestim 2-81  
sbioplot 2-92  
sbioremovefromlibrary 2-94  
sbioreset 2-96  
sbioroot 2-99  
sbiosaveproject 2-101  
sbioselect 2-103  
sbioshowunitprefixes 2-115  
sbioshowunits 2-117  
sbiosimulate 2-119  
sbiosubplot 2-124  
sbiotrellis 2-126  
sbiunit 2-128  
sbiunitcalculator 2-132  
sbiunitprefix 2-133  
sbiovariant 2-136  
sbiowhos 2-139  
sbmlexport 2-141  
setactiveconfigset 4-192  
setparameter 4-194  
setspecies 4-196  
simbiology 2-145

## G

get method  
reference 4-91  
getadjacencymatrix method  
reference 4-93  
getconfigset method  
reference 4-95  
getdata method  
reference 4-99  
getdose method  
reference 4-106  
getparameters method  
reference 4-115

getsensmatrix method  
reference 4-117  
getspecies method  
reference 4-120  
getstoichmatrix method  
reference 4-122  
getvariant method  
reference 4-124  
GroupID property  
reference 6-73  
GroupLabel property  
reference 6-74  
GroupNames property  
reference 6-75

## H

HasLag property  
reference 6-76  
HasResponseVariable property  
reference 6-77

## I

IndependentVarLabel property  
reference 6-78  
IndependentVarUnits property  
reference 6-79  
InitialAmount property  
reference 6-80  
InitialAmountUnits property  
reference 6-81  
Inputs property  
reference 6-83  
Interval property  
reference 6-85

## K

kinetic law definition 4-4  
kinetic law object

reference 4-128  
KineticLaw property  
reference 6-86  
KineticLawName property  
reference 6-88

## L

LagParameter property  
reference 6-90  
LagParameterName property  
reference 6-91  
LogDecimation property  
reference 6-92

## M

MaximumNumberOfLogs property  
reference 6-94  
MaximumWallClock property  
reference 6-96  
MaxIterations property  
reference 6-98  
MaxStep property  
reference 6-100  
methods  
addcompartment 4-6  
addCompartment 4-11  
addconfigset 4-15  
addcontent 4-19  
adddose 4-21  
addevent 4-23  
addkineticlaw 4-26  
addparameter 4-34  
addproduct 4-39  
addreactant 4-42  
addreaction 4-45  
addrule 4-52  
addspecies 4-57  
addvariant 4-63

commit 4-65  
construct 4-74  
constructDefaultFixedEffectValues 4-76  
copyobj 4-77  
delete 4-83  
get 4-91  
getadjacencymatrix 4-93  
getconfigset 4-95  
getdata 4-99  
getdose 4-106  
getparameters 4-115  
getsensmatrix 4-117  
getspecies 4-120  
getstoichmatrix 4-122  
getvariant 4-124  
removeconfigset 4-152  
removedose 4-154  
removevariant 4-156  
rename 4-158  
reorder 4-160  
resample 4-165  
reset 4-168  
rmcontent 4-170  
rmproduct 4-173  
rmreactant 4-175  
select 4-183  
selectbyname 4-187  
set 4-190  
verify 4-215

## Methods

display 4-85  
model object  
reference 4-136 4-162 4-181 4-206 4-212  
ModelName property  
reference 6-101  
Models property  
reference 6-102  
Moiety Conservation  
function for 2-12  
Multiplier property



reference 6-103

## N

Name property  
reference 6-104

Normalization property  
reference 6-107

Notes property  
reference 6-108

## O

object

- abstract kinetic law 4-4
- compartment 4-67
- configset 4-71
- CovariateModel 4-79
- event 4-86
- kinetic law 4-128
- model 4-136 4-162 4-181 4-206 4-212
- parameter 4-140
- PKCompartment 4-142
- PKData 4-144
- PKModelDesign 4-146
- PKModelMap 4-148
- reaction 4-149
- root 4-177
- rule 4-179
- SimData 4-202
- unit 4-208 4-210

Observed property  
reference 6-109

Offset property  
reference 6-110

Outputs property  
reference 6-112

OutputTimes property  
reference 6-114

Owner property

reference 6-116

## P

Parameter Estimation  
function for 2-81

parameter object  
reference 4-140

Parameters property  
reference 6-119

ParameterVariableNames property  
reference 6-121

ParameterVariables property  
reference 6-123

Parent property  
reference 6-125

PKCompartment object  
reference 4-142

PKCompartments property  
reference 6-127

PKData object  
reference 4-144

PKModelDesign object  
reference 4-146

PKModelMap object  
reference 4-148

Products property  
reference 6-128

properties

- AbsoluteTolerance 6-2
- AbsoluteToleranceScaling 6-4
- AbsoluteToleranceStepSize 6-5
- Active 6-6
- Amount 6-8 to 6-9
- BoundaryCondition 6-12
- BuiltInLibrary 6-15
- Capacity 6-18
- CapacityUnits 6-19
- Compartments 6-21
- CompileOptions 6-23

Composition 6-25  
ConstantAmount 6-27  
ConstantCapacity 6-29  
ConstantValue 6-30  
Content 6-32  
CovariateLabels 6-34  
Data 6-36  
DataCount 6-37  
DataInfo 6-38  
DataNames 6-40  
DataSet 6-41  
DefaultSpeciesDimension 6-42  
DependentVarLabel 6-44  
DependentVarUnits 6-45  
DimensionalAnalysis 6-46  
Dosed 6-49  
DoseLabel 6-50  
DoseUnits 6-51  
DosingType 6-52  
DurationParameterName 6-53  
EliminationType 6-54  
ErrorTolerance 6-55  
Estimated 6-57  
Exponent 6-61  
Expression 6-65  
GroupID 6-73  
GroupLabel 6-74  
GroupNames 6-75  
HasLag 6-76  
HasResponseVariable 6-77  
IndependentVarLabel 6-78  
IndependentVarUnits 6-79  
InitialAmount 6-80  
InitialAmountUnits 6-81  
Inputs 6-83  
Interval 6-85  
KineticLaw 6-86  
KineticLawName 6-88  
LagParameter 6-90  
LagParameterName 6-91  
LogDecimation 6-92  
MaximumNumberOfLogs 6-94  
MaximumWallClock 6-96  
MaxIterations 6-98  
MaxStep 6-100  
ModelName 6-101  
Models 6-102  
Multiplier 6-103  
Name 6-104  
Normalization 6-107  
Notes 6-108  
Observed 6-109  
Offset 6-110  
Outputs 6-112  
OutputTimes 6-114  
Owner 6-116  
Parameters 6-119  
ParameterVariableNames 6-121  
ParameterVariables 6-123  
Parent 6-125  
PKCompartments 6-127  
Products 6-128  
RandomState 6-131  
Rate 6-133  
RateLabel 6-135  
RateUnits 6-134  
Reaction 6-138  
ReactionRate 6-139  
Reactions 6-142  
RelativeTolerance 6-143  
RepeatCount 6-145  
Reversible 6-146  
Rule 6-149  
Rules 6-154  
RuleType 6-151  
RunInfo 6-155  
RuntimeOptions 6-156  
SensitivityAnalysis 6-157  
SensitivityAnalysisOptions 6-159  
SolverOptions 6-163

- SolverType 6-166
  - Species 6-169
  - SpeciesVariableNames 6-170
  - SpeciesVariables 6-172
  - StartTime 6-174
  - StatesToLog 6-175
  - Stoichiometry 6-177
  - StopTime 6-179
  - Tag 6-183
  - TargetName 6-185
  - Time 6-188
  - TimeUnits 6-189
  - Type 6-191
  - UnitConversion 6-192
  - UserData 6-195
  - UserDefinedLibrary 6-196
  - Value 6-199
  - ValueUnits 6-200
  - ZeroOrderDurationParameter 6-202
- Properties
- Reactants 6-136
- R**
- RandomState property
    - reference 6-131
  - Rate property
    - reference 6-133
  - RateLabel property
    - reference 6-135
  - RateUnits property
    - reference 6-134
  - Reactants property
    - reference 6-136
  - reaction object
    - reference 4-149
  - Reaction property
    - reference 6-138
  - ReactionRate property
    - reference 6-139
  - Reactions property
    - reference 6-142
  - RelativeTolerance property
    - reference 6-143
  - removeconfigset method
    - reference 4-152
  - removedose method
    - reference 4-154
  - removevariant method
    - reference 4-156
  - rename method
    - reference 4-158
  - reorder method
    - reference 4-160
  - RepeatCount property
    - reference 6-145
  - resample method
    - reference 4-165
  - reset method
    - reference 4-168
  - Reversible property
    - reference 6-146
  - rmcontent method
    - reference 4-170
  - rmproduct method
    - reference 4-173
  - rmreactant method
    - reference 4-175
  - root object
    - reference 4-177
  - rule object
    - reference 4-179
  - Rule property
    - reference 6-149
  - Rules property
    - reference 6-154
  - RuleType property
    - reference 6-151
  - RunInfo property
    - reference 6-155

RuntimeOptions property  
reference 6-156

## S

sbioabstractkineticlaw function  
reference 2-2

sbioaccelerate function  
reference 2-6

sbioaddtolibrary function  
reference 2-9

sbioconsmoiety function  
reference 2-12

sbioconvertunits function  
reference 2-17

sbiocopylibrary function  
reference 2-19

sbiodesktop function  
reference 2-21

sbiodose function  
reference 2-23

sbioensembleplot function  
reference 2-25

sbioenssemblerun function  
reference 2-27

sbioensemblestats function  
reference 2-31

sbiogetmodel function  
reference 2-38

sbio alasterror function  
reference 2-41

sbio alastwarning function  
reference 2-45

sbio loadproject function  
reference 2-46

sbio model function  
reference 2-47

sbio nlmefit function  
reference 2-58

sbio nlmefitsa function

reference 2-66

sbio paramestim function  
reference 2-81

sbio plot function  
reference 2-92

sbio removefromlibrary function  
reference 2-94

sbio reset function  
reference 2-96

sbio root function  
reference 2-99

sbio saveproject function  
reference 2-101

sbio select function  
reference 2-103

sbio showunitprefixes function  
reference 2-115

sbio showunits function  
reference 2-117

sbio simulate function  
reference 2-119

sbio subplot function  
reference 2-124

sbio trellis function  
reference 2-126

sbio unit function  
reference 2-128

sbio unitcalculator function  
reference 2-132

sbio unitprefix function  
reference 2-133

sbio variant function  
reference 2-136

sbio whos function  
reference 2-139

sbm lexport function  
reference 2-141

select method  
reference 4-183

selectbyname method

reference 4-187

Sensitivity Analysis  
properties for 6-83 6-107 6-112 6-157 6-159

SensitivityAnalysis property  
reference 6-157

SensitivityAnalysisOptions property  
reference 6-159

set method  
reference 4-190

setactiveconfigset function  
reference 4-192

setParameter function  
reference 4-194

setspecies function  
reference 4-196

simbiology function  
reference 2-145

SimData object  
reference 4-202

SolverOptions property  
reference 6-163

SolverType property  
reference 6-166

Species property  
reference 6-169

SpeciesVariableNames property  
reference 6-170

SpeciesVariables property  
reference 6-172

StartTime property  
reference 6-174

StatesToLog property  
reference 6-175

Stoichiometry property  
reference 6-177

StopTime property  
reference 6-179

**T**

Tag property  
reference 6-183

TargetName property  
reference 6-185

Time property  
reference 6-188

TimeUnits property  
reference 6-189

Type property  
reference 6-191

**U**

unit object  
reference 4-208 4-210

UnitConversion property  
reference 6-192

UserData property  
reference 6-195

UserDefinedLibrary property  
reference 6-196

**V**

Value property  
reference 6-199

ValueUnits property  
reference 6-200

verify method  
reference 4-215

**Z**

ZeroOrderDurationParameter property  
reference 6-202