SimBiology

For Use with MATLAB®

- Computation
- Visualization
- Programming

Reference

Version 1

The MathWorks

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information
☎ 508-647-7000	Phone
508-647-7001	Fax

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

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

SimBiology Reference

© COPYRIGHT 2005 by The MathWorks, Inc.

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

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

Trademarks

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

Other product or brand names are trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

September 2005 Online only New for Version 1.0 (Release 14SP3+)

Contents

1 Functions – Categorical List

2 Functions — Alphabetical List

3 Methods – Categorical List

4 Methods — Alphabetical List

5 Properties – Categorical List

6 Properties — Alphabetical List

Index

Functions – Categorical List

This chapter is a reference for the functions in SimBiology. Functions are grouped into the following categories.

Tools

Functions for modeling, simulation, and analysis.

`sbiodesktop`	Open SimBiology modeling and simulation GUI
`sbiogetmodel`	Get SimBiology model object that generated simulation data
`sbiogetnamedstate`	Return state and time data from simulation results
`sbiohelp`	Display information for SimBiology functions
`sbioreset`	Delete all SimBiology model and simulation objects
`sbioselect`	Search for SimBiology objects with specified constraints
`sbiosimulate`	Simulate SimBiology model object

Projects

Functions for saving and opening projects in MATLAB.

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

SBML Models

Functions for reading and writing SBML models to files.

`sbmlexport`	Export SimBiology model to SBML file
`sbmlimport`	Import Systems Markup Language (SBML) formatted file

Object Constructors

MATLAB functions that create SimBiology objects.

`sbioabstractkineticlaw (abstractkineticlaw)`	Construct abstract kinetic law object
`sbiomodel (model)`	Construct model object
`sbioparameter (parameter)`	Construct parameter object
`sbioreaction (reaction)`	Construct reaction object
`sbioroot (root)`	Return SimBiology root object
`sbiorule (rule)`	Construct rule object
`sbiospecies (species)`	Construct species object

Units

Functions for unit conversion and creating user defined units.

`sbioconvertunits (unit)`	Convert unit and unit value to new unit
`sbioregisterunit (unit)`	Create user-defined unit
`sbioregisterunitprefix (unit)`	Create user-defined unit prefix
`sbioshowunitprefixes (unit)`	Return information about registered unit prefixes
`sbioshowunits (unit)`	Return information about registered units
`sbiounitcalculator (unit)`	Convert value between units
`sbiounregisterunit (unit)`	Remove user-defined unit from root and library
`sbiounregisterunitprefix (unit)`	Remove user-defined unit prefix from root and library .

2

Functions — Alphabetical List

sbioaddtolibrary (project)

Purpose Add abstract kinetic law to user-defined library

Syntax `sbioaddtolibrary (`*`abstkineticlawObj`*`)`

Description `sbioaddtolibrary (`*`abstkineticlawObj`*`)` adds the abstract kinetic law object (`abstkineticlawObj`) to the user-defined library. `abstkineticlawObj` is added to the SimBiology root object's `UserDefinedKineticLaws` list. `abstkineticlawObj` is available automatically in future MATLAB sessions. You can use the abstract kinetic law objects in the built-in and user-defined library to construct a kinetic law object with the method `addkineticlaw`.

To get the abstract kinetic law objects in the built-in and user-defined libraries, use the commands `get(sbioroot, 'BuiltInKineticLaws')`, `get(sbioroot, 'UserDefinedKineticLaws')`.

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

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

1. Create an abstract kinetic law.

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

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

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

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

   ```
   sbioaddtolibrary(abstkineticlawObj);
   ```

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

```
sbiowhos -kineticlaw -userdefined

Abstract Kinetic Law Object Array

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

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

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A + B <-> B + C');
kineticlawObj = addkineticlaw(reactionObj, 'mylaw1');
```

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

See Also

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

sbioconvertunits (unit)

Purpose

Convert unit and unit value to new unit

Syntax

```
sbioconvertunits(Obj,
'Unit')
```

Description

`sbioconvertunits(Obj, 'Unit')` converts the current `*Units` property on SimBiology object, `Obj` to the unit, `Unit`. This will configure the `*Units` property to `Unit` and update the corresponding value property. For example `sbioconverunits` on a `speciesObj` updates the `InitialAmount` property value and the `InitialAmountUnits` property value.

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

Example

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

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

   At the command prompt type,

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


   Species Object Array

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

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

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

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

   Units are converted from molecule to mole.

   ```
   get (speciesObj2, 'InitialAmountUnits')


   ans =

   mole
   ```

   InitialAmount value is changed.

   ```
   get (speciesObj2, 'InitialAmount')


   ans =

     3.8192e-023
   ```

See Also

`sbioshowunits`

sbiocopylibrary (project)

Purpose

Copy SimBiology library to disk

Syntax

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

Description

`sbiocopylibrary` copies all user-defined abstract kinetic laws to a file.`sbiocopylibrary ('kineticlaw',`*`'LibraryFileName'`*`)` copies all user-defined abstract kinetic laws to the file `LibraryFileName.sbklib`.

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

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

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

Example

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

1 Create an abstract kinetic law.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined
```

```
Abstract Kinetic Law Object Array

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

3 Copy the user-defined kinetic law library.

```
sbiocopylibrary kineticLaw myLibFile
```

4 Verify with `sbiowhos`.

```
sbiowhos -kineticlaw myLibFile
```

See Also

`sbioaddtolibrary`, `sbioabstractkineticlaw`, `sbioregisterunit`, `sbioregisterunitprefix`, `sbioremovefromlibrary`

sbiodesktop

Purpose

Open SimBiology modeling and simulation GUI

Syntax

```
sbiodesktop
sbiodesktop(modelObj)
```

Arguments

`modelObj`	Model object or an array of model objects. Enter a variable name for a top-level SimBiology model object.

Description

`sbiodesktop` opens the SimBiology GUI. The SimBiology GUI enables you to:

- Build a SimBiology model using reaction pathways and enter kinetic data for the reactions.
- Import or export SimBiology models to and from the MATLAB workspace or from a Systems Biology Markup Language (SBML) file.
- Modify an existing SimBiology model.
- Simulate a SimBiology model.
- View results from the simulation.
- Create and/or modify user-defined units and unit prefixes.
- Create and/or modify user-defined abstract kinetic law objects.

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

In contrast, A SimBiology model object that has its property `Parent` set to another SimBiology model is a submodel and is not stored by the SimBiology root.

Example

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

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

See Also

`sbioroot`

sbiogetmodel

Purpose

Get SimBiology model object that generated simulation data

Syntax

`modelObj = sbiogetmodel(timeseriesObj)`

Arguments

`timrseriesObj`	Time series object returned by the function `sbiosimulate`.
`modelObj`	Model object associated with the time series object.

Description

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

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

Example

Retrieve the model object that generated the time series object.

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

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

2. Get the model that generated the simulation results.

   ```
   modelObj2 = sbiogetmodel(timeseriesObj)


   SimBiology Model - Oscillator

   Model Components:
     Models:          0
   ```

```
   Parameters:       0
   Reactions:        42
   Rules:            0
   Species:          23
```

3 Check that the two models are the same.

```
modelObj == modelObj2

ans =
       1
```

See Also `sbiosimulate`

sbiogetnamedstate

Purpose Return state and time data from simulation results

Syntax

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

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

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

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

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

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

1. Import the file for the`oscillator` model.

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

2 Simulate `modelObj`.

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

3 Get the data for the species named`pol`.

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

4 Get the data and associated names for each `x`.

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

5 Plot the results of the simulation.

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

See Also

`sbiosimulate`

sbiohelp

Purpose Display information for SimBiology functions

Syntax

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

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

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

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

Examples

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

See Also MATLAB function `help`

sbioloadproject (project)

Purpose

Load SimBiology project from project file

Syntax

```
sbioloadproject projFilename
sbioloadproject projFilename variableName
sbioloadproject projFilename variableName1 variableName2
variableName3
```

Description

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

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

`sbioloadproject` *`projFilename variableName1 variableName2 variableName3`*loads only the variables *`variableName1`*, *`variableName2`*, and *`variableName3`* from the project. The contents of the project file can be displayed by using the `sbiowhos` command.

See Also

`sbiosaveproject`, `sbiowhos`, `sbioaddtolibrary`, `sbioremovefromlibrary`

Purpose

Create user-defined unit

Syntax

```
sbioregisterunit('Name', 'Composition', Multiplier)
sbioregisterunit('Name', 'Composition', Multiplier, Offset)
```

Description

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

`sbioregisterunit('Name', 'Composition', Multiplier, Offset)` creates a unit with the specified offset. Available units can be listed with the `sbioshowunits` function.

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

Examples

Example 1

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

Example 2

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

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

2 Display the unit using the command `sbiowhos`

```
sbiowhos -userdefined -unit


SimBiology UserDefined Units

Index:  Name:            Composition:         Multiplier:     Offset:

1       firstconstant     1/second            1.000000        0.000000

2       secondconstant    1/molecule*second   1.000000        0.000000
```

See Also

`sbioshowunits`, `sbioregisterunitprefix`, `sbiounregisterunit`

sbioregisterunitprefix (unit)

Purpose

Create user-defined unit prefix

Syntax

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

Description

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

- *`NameValue`* is the name of the prefix. Valid names must begin with a letter and can contain characters, underscores, or numbers. Built-in prefixes are defined based on the International System of Units (SI).
- *`Exponent`* shows the value of `10^Exponent`that defines the relationship of the unit *`Name`* to the base unit. For example for the unit `picomole` `Exponent` is `12`.

Example

1 Register a unit prefix.

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


SimBiology UserDefined Unit Prefixes

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

See Also

`sbioshowunits`, `showunitprefixes`, `sbiounregisterunit`

Purpose

Remove abstract kinetic law or unit from user-defined library

Syntax

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

Description

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

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

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

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

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

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

Example

Shows you how to remove an abstract kinetic law from the user-defined library.

1 Create an abstract kinetic law.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined

Abstract Kinetic Law Object Array

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

3 Remove the abstract kinetic law.

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

See Also

`sbioaddtolibrary`, `sbioabstractkineticlaw`, `sbioregisterunit`, `sbioregisterunitprefix`, `sbiounregisterunit`, `sbiounregisterprefix`

Purpose

Delete all SimBiology model and simulation objects

Syntax

`sbioreset`

Description

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

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

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

Example

Shows you the difference between `sbioreset` and `clear all`.

1. Import a model into the workspace.

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

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

   ```
   rootObj = sbioroot

   SimBiology Root Contains:

    Models:                            1
    Builtin Abstract Kinetic Laws:     3
    User Abstract Kinetic Laws:        0
   ```

```
 Builtin Units:                     54
 User Units:                        0
 Builtin Unit Prefixes:             13
 User Unit Prefixes:                0
```

2 The command `clear all` clears the workspace, but the `modelObj` still exists on the `rootObj`.

```
clear all

rootObj


SimBiology Root Contains:

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

3 The command `sbioreset` deletes the `modelObj` from the root.

```
sbioreset
rootObj


SimBiology Root Contains:

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

```
    User Unit Prefixes:              0
```

See Also

`sbioroot`

sbiosaveproject (project)

Purpose

Save all models in SimBiology root object

Syntax

```
sbiosaveproject projFilename
sbiosaveproject projFilename variableName
sbiosaveproject projFilename variableName1 variableName2
variableName3
```

Description

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

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

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

`sbiosaveproject` *`projFilename variableName1 variableName2 variableName3`* saves *`variableName1`*, *`variableName2`*, and *`variableName3`*.

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

Examples

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

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

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

```
sbiosaveproject myprojectfile modelObj timeseriesObj
```

See Also

`sbioloadproject`, `sbiowhos`, `sbioaddtolibrary`, `sbioremovefromlibrary`

Purpose

Search for SimBiology objects with specified constraints

Syntax

```
Out = sbioselect(Obj, 'PropertyName', PropertyValue)
Out =
sbioselect(Obj,'Type','TypeValue','PropertyName',PropertyValue)
Out = sbioselect(Obj,
'Where','PropertyName', Condition, PropertyValue)
Out = sbioselect(Obj, 'Where', 'PropertyName1','Condition1',
PropertyValue1,
'Where','PropertyName2','Condition2',PropertyValue2,...)
out= sbioselect(Obj,'depth', Number,...)
```

Arguments

`Obj`	SimBiology object to search
`PropertyName`	Any property of *`Obj`*
`PropertyValue`	Valid `PropertyValue`
`TypeValue`	Type of SimBiology object to search, for example `species`, `reaction`, `kineticlaw`
`Condition`	Constraint to impose on the search. See table below for a list of `Conditions`.

Description

`sbioselect` searches for objects with specified constraints.

`Out` `= sbioselect(`*`Obj`*`, `*`'PropertyName'`*`, `*`PropertyValue`*`)` finds the objects with the property name (*`PropertyName`*) and property value (*`PropertyValue`*) contained in any SimBiology object (`Obj`) or an array of SimBiology objects.

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

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

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

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

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

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

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

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

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

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

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

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

- The `regexp` and `regexpi` conditional value supports any of the expressions supported by the functions `regexp` and `regexpi` (`regexp`). When a string property value is searched for without specifying a condition, it must use the same format as `get` returns. For example, if `get` returns the `Name` as '`MyObject`', `sbioselect` will not find an object with a `Name` property value of '`myobject`'.

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

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

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

Examples

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

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

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

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

See Also

`regexp`

Purpose

Return information about registered unit prefixes

Syntax

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

Description

`sbioshowunitprefixes` returns information about registered unit prefixes.

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

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

`[Name, Multiplier, Builtin] = sbioshowunitprefixes`returns whether the unit prefix is built-in or user-defined for each unit prefix in `Name` to `Builtin`. `Builtin` is an array of logical values. If `Builtin` is true for a unit prefix, the unit prefix is built-in. If `Builtin` is false for a unit prefix, the unit prefix is user-defined.

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

- `Name` is the name of the prefix. Built-in prefixes are defined based on the International System of Units (SI).
- `Multiplier` shows the value of `10^Exponent`that defines the relationship of the unit `Name` to the base unit. For example the multiplier in `picomole` is `10e-12`.

Examples

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

See Also

`sbioregisterunit`, `sbioshowunits`, `sbioconvertunits`

sbioshowunits (unit)

Purpose

Return information about registered units

Syntax

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

Description

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

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

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

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

`[Name, Composition, Multiplier, Offset, Builtin] = sbioshowunits` returns whether the unit is built-in or user-defined for each unit in `Name` to `Builtin`. `Builtin` is an array of logical values. If `Builtin` is true for a unit, the unit is built-in. If `Builtin` is false for a unit, the unit is user-defined.

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

- *`Name`* is the name of the built-in or user-defined unit. *`Name`* must begin with characters and can contain characters, underscores or numbers.
- *`Composition`* shows the combination of base and derived units that defines the unit *`Name`*. For example `molarity` is `mole/liter`.
- *`Multiplier`* is the numerical value that defines the relationship between the unit *`Name`* and the base or derived unit as a product of the *`Multiplier`* and the base unit or derived unit. For example `1 mole` is `6.0221e23*molecule`. The *`Multiplier`* is `6.0221e23`.
- *`Offset`* is the numerical value by which the unit composition is modified from the base unit. For example `°Celsius = (5/9)*(°Fahrenheit-32)`; *`Multiplier`* is `5/9` and *`Offset`* is `32`.

Examples

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

See Also

`sbioregisterunit`, `sbioshowunitprefixes`, `sbioconvertunits`

sbiosimulate

Purpose

Simulate SimBiology model object

Syntax

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

Description

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

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

`[T,X] = sbiosimulate(modelObj, configsetObj)` simulates a model object (*`modelObj`*) using a configuration set (*`configset`*) that overrides the active configuration set attached to the model (*`modelObj`*). After command is executed this override does not exist; the configuration set that is defined as 'active' is reinstated.

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

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

`timeseriesObj = sbiosimulate(modelObj)` simulates a model object (`modelObj`) using the active configuration set attached to the model (*`modelObj`*) and returns the results to time series object (`timesereiesObj`).

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

To get the configuration sets attached to a model, use `getconfigset`. To attach a new or existing configuration set to a model, use `addconfigset`. To set the active configuration set of a model, use `setactiveconfigset`. To use command line help to get more information on these methods, for example, '`help SimBiology.Model.getconfigset`'.

Property Summary

Configuration set property summary

Active	Property to indicate object use during a simulation
CompileOptions	Property holding dimensional analysis and unit conversion information
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
RuntimeOptions	Property holding options for logged species
SolverOptions	Property holding the model solver options
SolverType	Property to select solver type for simulation
StopTime	Property to set the stop time for a simulation

StopTimeType	Property to specify the type of stop time for a simulation
TimeUnits	Property to show the stop time units for a simulation
Type	Property to indicate SimBiology object type

Example 1

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

1. Read the file for the`oscillator` model.

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

2. Get the active `configset`.

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

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

```
set(configsetObj, 'SolverType', 'ode45');
set(configsetObj, 'StopTime', 10);
```

4. Simulate `modelObj`.

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

5. Plot the results of the simulation.

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

Example 2

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

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

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

2. Simulate `modelObj`.

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

3. Plot the results of the simulation.

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

See Also

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

sbiounitcalculator (unit)

Purpose Convert value between units

Syntax

```
result = sbiounitcalculator('fromUnits',
'toUnits', Value)
```

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

Example

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

See Also `sbioshowunits`

Purpose

Remove user-defined unit from root and library

Syntax

```
sbiounregisterunit('Name')
```

Description

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

Example

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

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

2. Display the unit using the command `sbiowhos`

```
sbiowhos -userdefined -unit


SimBiology UserDefined Units

Index:  Name:            Composition:          Multiplier:     Offset:

1       firstconstant     1/second             1.000000        0.000000

2       secondconstant    1/molecule*second    1.000000        0.000000
```

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

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

```
SimBiology UserDefined Units

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

See Also `sbioshowunits`, `sbioregisterunit`, `sbiounregisterunitprefix`, `sbioroot`, `sbiowhos`

Purpose Remove user-defined unit prefix from root and library .

Syntax `sbiounregisterunitprefix('Name')`

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

Example

1. Register a unit prefix.

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


SimBiology UserDefined Unit Prefixes

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

2. Unregister the unit prefix.

```
sbiounregisterunitprefix('peta');
```

See Also `sbioshowunitprefixes`, `sbioshowunits`, `sbioregisterunitprefix`, `sbiounregisterunit`, `sbioroot`, `sbiowhos`

sbiowhos (project)

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

Syntax

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

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

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

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

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

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

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

Examples

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

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

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

% Show all contents of project file.
  sbiowhos myprojectfile

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

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

See Also

MATLAB function `whos`

sbmlimport

Purpose

Import Systems Markup Language (SBML) formatted file

Syntax

`modelObj = sbmlimport('FileName')`

Arguments

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

Description

`modelObj = sbmlimport('FileName')` imports a SBML formatted file with name *`FileName`* into MATLAB and creates a model object `modelObj`. `FileName` extensions can be `.sbml` or `.xml`. The`modelObj` properties can be viewed with the `get` command. `modelObj` properties can be modified with the `set` command. At the command line help for `modelObj` functions can be returned with the `sbiohelp` command.

Functional Characteristics and Limitations

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

Example

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

Reference

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

See Also

`sbmlexport`, `sbiosimulate`

MATLAB functions `get` and `set`

sbmlexport

Purpose

Export SimBiology model to SBML file

Syntax

`sbmlexport(modelObj, 'FileName')`

Arguments

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

Description

`sbmlexport(modelObj, 'FileName')`exports a SimBiology model object (`Mobj`) to a file with a Systems Biology Markup Language (SBML) Level 2 format. The default file extension is `.xml`.

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

Functional Characteristics and Limitations

The `sbmlexport` function,

- Exports SBML Level 2 compatible files
- Exports SBML compliant unit definitions
- Does not support submodels
- Does not export features that are not supported by SBML. These are,
  - Abstract kinetic law name and corresponding expression (note that `sbmlexport` exports the reaction rate equation)
  - Configurations sets
  - `Active` property
  - `UserData`

- Tag

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

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

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

See Also `sbmlimport`, `sbiomodel`, `sbiosaveproject`

Methods – Categorical List

This chapter is a reference for the object methods in SimBiology. Methods are grouped into the following categories.

Abstract Kinetic Laws

Methods for abstract kinetic law objects

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

Configuration Sets

Methods for configuration set objects.

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

Kinetic Laws

Methods for kinetic law objects.

`addparameter (model, kineticlaw)`	Add parameter object 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
`getparameters (kineticlaw)`	Get specific parameters in kinetic law object
`getspecies (kineticlaw)`	Get specific species in kinetic law object
`setparameter (kineticlaw)`	Specify specific parameters in kinetic law object
`setspecies (kineticlaw)`	Specify species in kinetic law object

Models

Methods for model objects.

`addconfigset (model)`	Add configuration set object to model object
`addmodel (model)`	Add submodel object to model object
`addparameter (model, kineticlaw)`	Add parameter object to model or kinetic law object
`addreaction (model)`	Add reaction object to model object
`addrule (model)`	Add rule object to model object
`addspecies (model)`	Add species object to 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
`getadjacencymatrix (model)`	Return adjacency matrix from model object
`getconfigset (model)`	Get configuration set object from model object
`getstoichmatrix (model)`	Return stoichiometry matrix from model object
`removeconfigset (model)`	Remove configuration set from model
`setactiveconfigset (model)`	Set the active configuration set for model object

Parameters

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

Reactions

Methods for reaction objects

`addkineticlaw (reaction)`	Add kinetic law object to reaction object
`addproduct (reaction)`	Add product species object to reaction object
`addreactant (reaction)`	Add species object as a 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
`rmproduct (reaction)`	Remove species object from reaction object products
`rmreactant (reaction)`	Remove species object from reaction object reactants

Root

Methods for the root object.

`copyobj (any object)`	Copy SimBiology object and its children
`delete (any object)`	Delete SimBiology object
`reset (root)`	Delete all model objects from the root object

Rules

Methods for rule objects.

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

Species

Methods for species objects.

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

Using Object Methods

Command line syntax for using methods with SimBiology objects.

Constructing (Creating) Objects

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

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

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

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

To create objects references by a reaction, us the reaction object methods addkineticlaw, addparemeter, addproduct, and addreactant.

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

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

Using Object Methods

Using MATLAB function notation.

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

Using object dot notation.

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

Help for Objects, Methods and Properties

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

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

4

Methods — Alphabetical List

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

addconfigset (model)

Purpose

Add configuration set object to model object

Syntax

`configsetObj = addconfigset(modelObj, 'NameValue')`

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 a pointer (*`configsetObj`*) to the object.

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

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

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

You can view additional configuration set object properties with the function `get`. You can change additional model object properties with the function `set`.

Method Summary

Methods for configuration set objects

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

Property Summary

Properties for configuration set objects

Active	Property to indicate object use during a simulation
CompileOptions	Property holding dimensional analysis and unit conversion information
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
RuntimeOptions	Property holding options for logged species
SolverOptions	Property holding the model solver options
SolverType	Property to select solver type for simulation
StopTime	Property to set the stop time for a simulation
StopTimeType	Property to specify the type of stop time for a simulation
TimeUnits	Property to show the stop time units for a simulation
Type	Property to indicate SimBiology object type

Examples

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

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

2. Configure the `configsetObj` `StopTime` to `3000`.

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


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

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

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

See Also

model object methods `getconfigset`, `removeconfigset`, `setactiveconfigset`

MATLAB functions `get` and `set`.

Purpose Add kinetic law object to reaction object

Syntax

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

Arguments

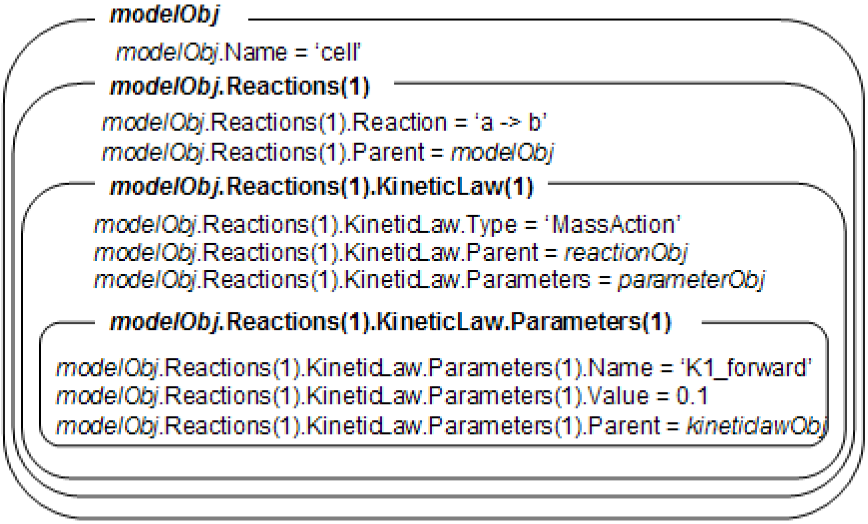
reactionObj	Reaction object. Enter a variable name for a reaction object.
KineticLawNameValue	Property to select the type of kinetic law object to create. Enter either `MassAction` or `Heneri-Michelis-Menten`.

Description

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

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

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



`KineticLawNameValue` is any valid abstract kinetic law. See “Abstract Kinetic Law” on page 6-27 for a definition of abstract kinetic laws and more information about how they are used to get the reaction rate expression.

You can find valid *`KineticLawNameValue`*s by querying the SimBiology root object with the commands: `get(sbioroot, 'BuiltInKineticLaws')`, and `get(sbioroot, 'UserDefinedKineticLaws')`.`sbiowhos -kineticlaw` lists `BuiltInKineticLaws` and `UserDefinedKineticLaws` in the SimBiology root. The root contains all `BuiltInKineticLaws` and all `UserDefinedKineticLaws` that are added using `sbioaddtolibrary` or `addtolibrary`.

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

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

Method Summary

Methods for kinetic law objects

`addparameter (model, kineticlaw)`	Add parameter object 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
`getparameters (kineticlaw)`	Get specific parameters in kinetic law object
`getspecies (kineticlaw)`	Get specific species in kinetic law object
`setparameter (kineticlaw)`	Specify specific parameters in kinetic law object
`setspecies (kineticlaw)`	Specify species in kinetic law object

Property Summary

Properties for kinetic law objects

Annotation	Property with information about a SimBiology object
Expression	Property containing the expression used to determine the reaction rate equation

KineticLawName	Property showing name of abstract kinetic law
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parameters	Property with array of parameter objects
ParameterVariableNames	Property showing cell array of reaction rate parameters
ParameterVariables	Property showing parameters in abstract kinetic law
Parent	Property indicating the parent object
SpeciesVariables	Property showing species in abstract kinetic law
SpeciesVariablesNames	Property showing cell array of species used in reaction rate equation
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Example1

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

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

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

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

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

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

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

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

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

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

SimBiology creates two parameter objects with concrete values that will be associated with the abstract kinetic law parameters.

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

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

SimBiology associates the concrete parameters in the property `ParameterVariablesNames` with the abstract parameters in the property `ParameterVariables` using a one-to-one mapping in the order given.

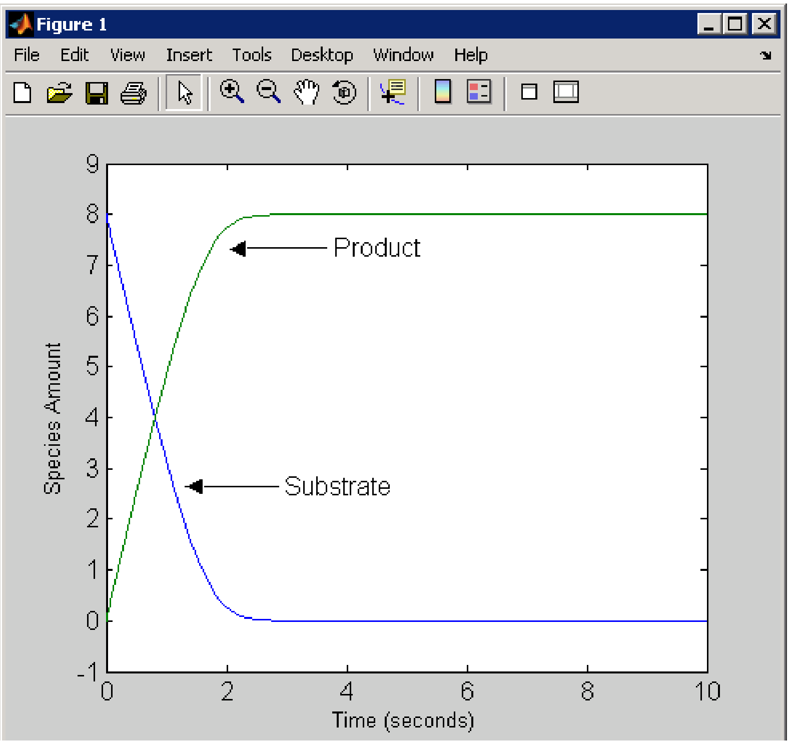
5. Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
get (reactionObj, 'ReactionRate')

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

6. Enter an initial value for the substrate and simulate.

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



Example2

Example using the built-in kinetic law `MassAction`.

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

```
modelObj = sbiomodel ('Cell');
reactionObj = addreaction (modelObj, 'a -> b');
```

2. Define an abstract kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
get(kineticlawObj, 'Expression')

ans =
    MassAction
```

Notice, the property `Expression` for an abstract kinetic law with property Type set to `MassAction` does not show the parameters and species in the reaction rate.

3. Assign the rate constant for the reaction.

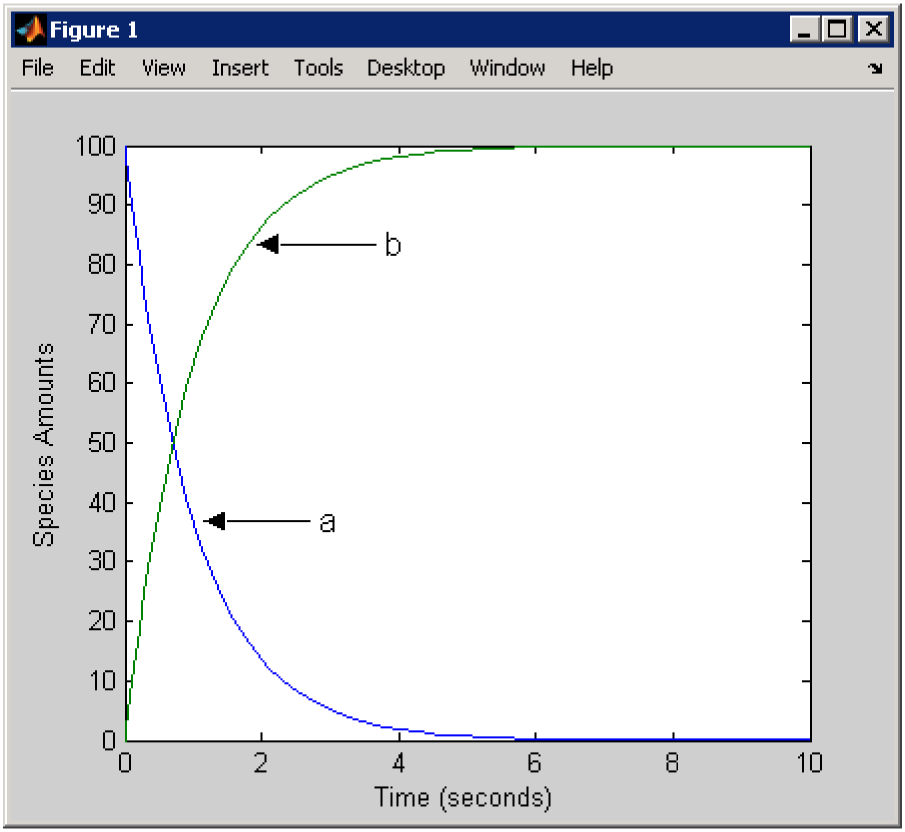
```
parameterObj = addparameter(kineticlawObj, 'k_forward');
set (kineticlawObj, 'ParameterVariablenames', 'k_forward');
get (reactionObj, 'ReactionRate')

ans =
    k_forward*a
```

4. Enter an initial value for the substrate and simulate.

```
modelObj.Species(1).InitialAmount = 100;
[T, X] = sbiosimulate(modelObj);
plot(T,X)
```

The value used for `k_forward` is default value = `1.0`.



See Also

SimBiology method for model object `addreaction`, and method for kinetic law object `setparameter`

addmodel (model)

Purpose

Add submodel object to model object

Syntax

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

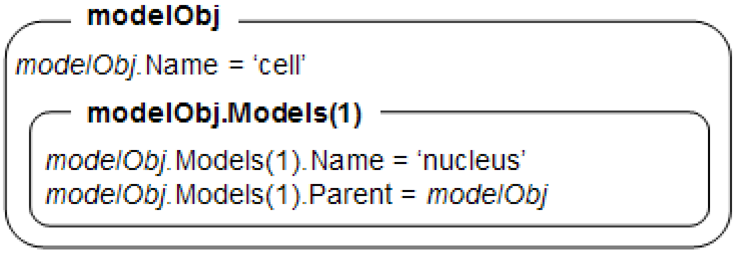
Arguments

modelObj	Model object. Enter a name for a model object.
NameValue	Descriptive name for a model object. Enter a unique character string. A model object can be referenced by other objects using this property.
submodelObj	Model object to be added as submodel.

Description

`submodelObj = addmodel(modelObj, 'NameValue')` creates a submodel object and returns a pointer (`submodelObj`) to the object. In the submodel object, this method assigns a value (`NameValue`) to the property `Name`, and assigns the model object (`modelObj`) to the property `Parent`. In the model object, this method assigns the submodel object to the property `Models`.

```
modelObj = sbiomodel('cell')
submodelObj = addmodel('nucleus')
```



A model object must have a unique name at the level it is created. For example, if you create a model with the name `cell`, you cannot create another model object named `cell`. However, a model object can contain a submodel object named cell which can contain a submodel object named `cell`.

`modelObj` does not have access to *`submodelObj`* parameters. However, *`submodelObj`* does have access and can use *`modelObj`* parameters.

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

You can view additional model object properties with the function `get`. You can change additional model object properties with the function `set`. You can view the submodel objects of `modelObj` with the command, `get(modelObj, 'Models')`.

Examples

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

See Also

`sbiomodel`

Purpose

Add parameter object to model or kinetic law object

Syntax

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

Arguments

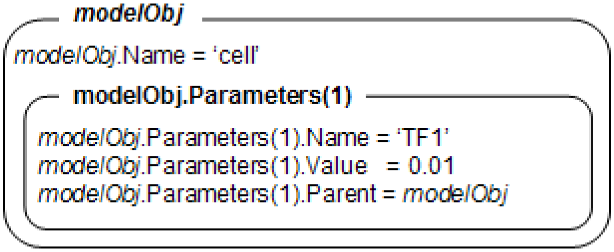
`Obj`	Model or kinetic law object. Enter a variable name for the object.
`NameValue`	Property for a parameter object. Enter a unique character string. `NameValue` can be a cell array of parameter names.Since objects can use this property to reference a parameter, a parameter object must have a unique name at the level it is created. For example, a kinetic law object cannot contain two parameter objects named `kappa`. However, the model object that contains the kinetic law object can contain a parameter object named `kappa` along with the kinetic law object.
`ValueValue`	Property for a parameter object. Enter a number.

Description

`parameterObj = addparameter(`*`Obj`*`, '`*`NameValue`*`')`creates a parameter object and returns a pointer (*`parameterObj`*) to the object. In the parameter object, this method assigns a value (*`NameValue`*) to the property `Name`, assigns a value `1` to the property `Value`, and assigns the model or kinetic law object to the property `Parent`. In the model or kinetic law object, (*`Obj`*), this method assigns the parameter object to the property `Parameters`.

A parameter object defines an assignment that a model, or a kinetic law can use. The scope of the parameter is defined by the parameter parent. If a parameter is defined with a kinetic law object, then only the kinetic law object and objects within the kinetic law object can use the parameter. If a parameter object is defined with a model object as its parent, then all objects within the model (including all rules, submodels and kinetic laws) can use the parameter.

```
modelObj = sbiomodel('cell')
parameterObj = addparameter(modelObj, 'TF1', 0.01)
```



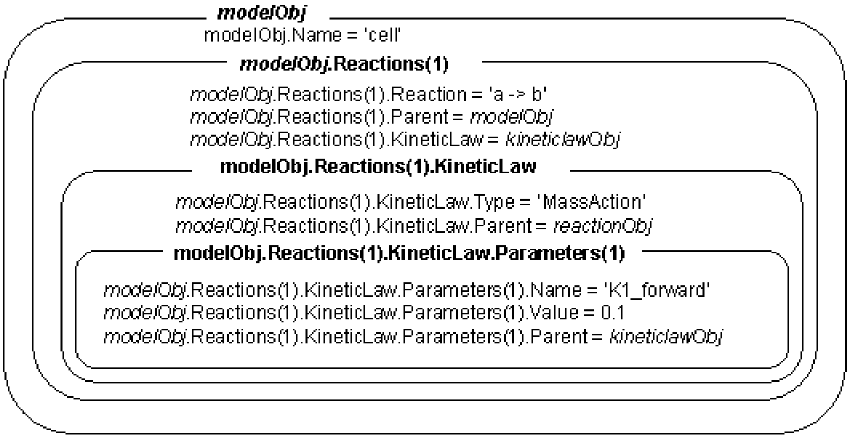
modelObj

modelObj.Name = 'cell'

modelObj.Parameters(1)

modelObj.Parameters(1).Name = 'TF1'
modelObj.Parameters(1).Value = 0.01
modelObj.Parameters(1).Parent = *modelObj*

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



modelObj

modelObj.Name = 'cell'

***modelObj*.Reactions(1)**

modelObj.Reactions(1).Reaction = 'a -> b'
modelObj.Reactions(1).Parent = *modelObj*
modelObj.Reactions(1).KineticLaw = *kineticlawObj*

modelObj.Reactions(1).KineticLaw

modelObj.Reactions(1).KineticLaw.Type = 'MassAction'
modelObj.Reactions(1).KineticLaw.Parent = *reactionObj*

modelObj.Reactions(1).KineticLaw.Parameters(1)

modelObj.Reactions(1).KineticLaw.Parameters(1).Name = 'K1_forward'
modelObj.Reactions(1).KineticLaw.Parameters(1).Value = 0.1
modelObj.Reactions(1).KineticLaw.Parameters(1).Parent = *kineticlawObj*

`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`.

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

Property Summary

Properties for parameter objects

Annotation	Property with information about a SimBiology object
ConstantValue	Property to indicate variable or constant parameter value
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object
Value	Property to assign value to parameter object
ValueUnits	Property with parameter value units

Example

1 Create model object, then add a reaction object.

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

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

3 Add a parameter and assign it to the kinetic law object (`kineticlawObj`); add another parameter and assign to the model object (`modelObj`).

```
% Add parameter to kinetic law object
parameterObj1 = addparameter (kineticlawObj, 'K1');

get (kineticlawObj, 'Parameters')
```

MATLAB returns

```
Parameter Object Array

Index:    Name:    Value:    ValueUnits:
1         K1       1
```

```
% Add parameter with value 0.9 to model object
parameterObj1 = addparameter (modelObj, 'K2', 0.9);

get (modelObj, 'Parameters')
```

MATLAB returns

```
Parameter Object Array

Index:    Name:    Value:    ValueUnits:
1         K2       1
```

See Also

MATLAB functions–`copyobj`,`delete`

addproduct (reaction)

Purpose

Add product species object to reaction object

Syntax

```
speciesObj = addproduct(reactionObj, ’NameValue’)
speciesObj = addproduct(reactionObj, speciesObj)
speciesObj = addproduct(reactionObj,
'NameValue', Stoichcoefficient)
speciesObj = addproduct(reactionObj, speciesObj, Stoichcoefficient)
```

Arguments

`reactionObj`	Reaction object. Enter a name for the reaction object.
`NameValue`	Property of a species object that names the object (not the reaction object). Enter a unique character string. For example, `'fructose 6-phosphate'`.A species object can be referenced by other objects using this property. You can use the function `sbioselect` to find an object with a specific `NameValue`.
`speciesObj`	Species object.
`Stoichcoeffieient`	Stoichiometric coefficients for products, length of array equal to length of `NameValue` or length of *`speciesObj`*.

Description

`speciesObj` `= addproduct(`*`reactionObj`*`, `*`’NameValue’`*`)` creates a species object and returns a pointer (*`speciesObj`*) to the object. In the species object, this method assigns the value (*`NameValue`*) to the property `Name`, and assigns the parent object of the *`reactionObj`* to the property `Parent`. In the reaction object, this method assigns the species object to the property `Products`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `1` to the property `Stoichiometry`.

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

`speciesObj = addproduct(reactionObj, speciesObj)`, in the species object (*speciesObj*), assigns the parent object of the *reactionObj* to the species property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Products`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `1` to the property `Stoichiometry`.

`speciesObj = addproduct(reactionObj, 'NameValue', Stoichcoefficient)`, in addition to the description above, this method adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`. If `NameValue` is a cell array of species names, then `Stoichcoefficient` must be a vector of doubles with the same length as `NameValue`.

`speciesObj = addproduct(reactionObj, speciesObj, Stoichcoefficient)`, in addition to the description above, this method adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species SimBiology updates the reaction to use the new name. You must however configure all other applicable elements such as rules that use the species, and the kinetic law object.

See "Valid Species Names" on page 4-37 for more information on species names.

Example

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

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'A + C -> U');
```

2 Modify the reaction of the `reactionObj` from `A + C -> U` to `A + C -> U + 2 H`.

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

See Also `sbiospecies`, `addspecies`

Purpose

Add species object as a reactant to reaction object

Syntax

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

Arguments

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

Description

speciesObj = `addreactant(`*reactionObj*, *'NameValue'*`)` creates a species object and returns a pointer (*speciesObj*) to the object. In the species object, this method assigns the value (*NameValue*) to the property `Name`, and assigns the parent object of the *reactionObj* to the property `Parent`. In the reaction object, this method assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `-1` to the property `Stoichiometry`.

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

`addreactant(`*`reactionObj`*`, `*`speciesObj`*`, `*`Stoichcoeffieient`*`)`, in the species object (`speciesObj`), this method assigns the parent object to the `speciesObj` property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `-1` to the property `Stoichiometry`. If `speciesObj` is a cell array of species objects, then *`Stoichcoeffieient`* must be a vector of doubles with the same length as `speciesObj`.

`addreactant(`*`reactionObj`*`, '`*`NameValue`*`', `*`Stoichcoeffieient`*`)`, in addition to the description above, this method adds the stoichiometric coefficient (*`Stoichcoeffieient`*) to the property `Stoichiometry`. If *`NameValue`* is a cell array of species names, then *`Coefficient`* must be a vector of doubles with the same length as *`NameValue`*.

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

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species SimBiology updates the reaction to use the new name. You must however configure all other applicable elements such as rules that use the species, and the kinetic law object.

See "Valid Species Names" on page 4-37 for more information on species names.

Example

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

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'A -> U');
```

2 Modify the `reaction` of the `reactionObj` from `A -> U` to be `A + 3 C -> U`.

```
speciesObj = addreactant(reactionObj, 'C', 3);
```

See Also `sbiospecies`, `addspecies`

addreaction (model)

Purpose

Add reaction object 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

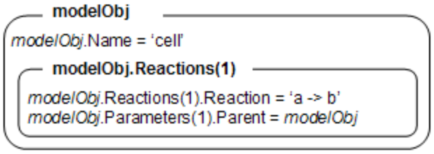
modelObj	SimBiology model object
ReactionValue	Property to specify the reaction equation. Enter a character string. A hyphen preceded by a space and right angle bracket (`->`) indicate reactants going froward to products. A hyphen with left and right angle brackets (`<->`), indicate a reversible reaction.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

`RStoichCoefficients`	Stoichiometric coefficients for reactants, length of array equal to length of *`ReactantsValue`*.
`PStoichCoefficients`	Stoichiometric coefficients for products, length of array equal to length of *`ProductsValue`*.

Description

`reactionObj = addreaction(modelObj, 'ReactionValue')` creates a reaction object, assigns a value (*`ReactionValue`*) to the property `Reaction`, assigns reactant species object(s) to the property `Reactants`, assigns the product species object(s) to the property `Products`, and assigns the model object to the property `Parent`. In the Model object (`modelObj`), this method assigns the reaction object to the property `Reactions`, and returns the reaction object (`reactionObj`).

```
reactionObj = addreaction(modelObj, 'a -> b')
```



If a species specified in a reaction does not exist, a species object is created and assigned to the model object property `Species`. You can manually add a species to a model object with the method `addspecies`.

You can add species to a reaction object using the methods `addreactant` or `addproduct`. You can remove species from a reaction object with the methods `rmreactant` or `rmproduct`. The property `Reaction` is modified by adding or removing species from the reaction equation.

You can copy a SimBiology reaction object can be copied to a SimBiology model object with the function, `copyobj`. You can remove SimBiology reaction object from a SimBiology model object with the function `delete`.

You can view additional reaction object properties with the `get` command, for example, the reaction equation of `reactionObj` can be viewed with the command, `get(reactionObj, 'Reaction')`. You can modify additional reaction object properties with the command, `set` .

`reactionObj = addreaction(modelObj, 'ReactantsValue', 'ProductsValue')` creates a reaction object, assigns a value to the property `Reaction` using the reactant (`ReactantsValue`) and product (`ProductsValue`) names, assigns the species objects to the properties `Reactants` and `Products`, and assigns the model object to the property Parent. In the Model object (`modelObj`), this method assigns the reaction object to the property `Reactions`, and returns the reaction object (`reactionObj`). The stoichiometric values are assumed to be `1`.

`reactionObj = addreaction(modelObj, 'ReactantsValue', RStoichCoefficients, 'ProductsValue', PStoichCoefficients)` adds stoichiometric coefficients (`RStoichCoefficients`) for reactant species, and stoichiometric coefficients (`PStoichCoefficients`) for product species to the property `Stoichiometry`. The length of `Reactants` and `RCoefficients` must be equal, and the length of `Products` and `PCoefficients` must be equal.

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

Method Summary

Methods for reaction objects

`addkineticlaw (reaction)`	Add kinetic law object to reaction object
`addproduct (reaction)`	Add product species object to reaction object
`addreactant (reaction)`	Add species object as a 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
`rmproduct (reaction)`	Remove species object from reaction object products
`rmreactant (reaction)`	Remove species object from reaction object reactants

Property Summary

Properties for reaction objects

Active	Property to indicate object use during a simulation
Annotation	Property with information about a SimBiology object
KineticLaw	Property showing kinetic law for `ReactionRate`
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Products	Property to indicate reaction products
Reactants	Property to indicate reaction reactants.
Reaction	Property to indicate the reaction object reaction
ReactionRate	Property containing the reaction rate equation in reaction object

Reversible	Property to indicate whether a reaction is reversible or irreversible
Stoichiometry	Property that describes species coefficients in a reaction
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Examples

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

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

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

2. Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'` .

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

   `reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3. The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) and one species variable (`S`) that should to be set. To set these variables, first create the parameter variables as parameter objects (`parameterObj1, parameterObj2`) with names `Vm_d,` and `Km_d`, and assign the objects `Parent` property value to the `kineticlawObj`.

   ```
   parameterObj1 = addparameter(kineticlawObj, 'Vm_d');
   ```

```
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

4 Set the variable names for the kinetic law object.

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

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

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns

```
ans =

Vm_d*[a]/(Km_d+[a])
```

See Also

`addkineticlaw`, `addproduct`, `addreactant`, `rmproduct`, `rmreactant`

Reaction object methods , , `KineticLaw` object method– .Reaction object properties– `Active`, `Annotation`, `Name`, `Notes`, `Parent`, `Products`, `Reactants`, `Reaction`, `ReactionRate`, `KineticLaw`, `Reversible`, `Stoichiometry`, `Tag`, `Type`, `UserData`.

Purpose Add rule object to model object

Syntax

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

Arguments

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

Description A rule is a mathematical expression that changes the amount of a species or the value of a parameter. It also defines how species and parameters interact with one another.

`ruleObj` `= addrule(`*`modelObj`*`, '`*`RuleValue`*`')` creates a rule object and returns a pointer (*`ruleOb`*`j`) to the object. In the rule object, this method assigns a value (`'`*`RuleValue`*`'`) to the property `Rule`, assigns the value `'algebraic'` to the property `RuleType`, and assigns the model object (*`modelObj`*) to the property `Parent`. In the model object (`modelObj`), this method assigns the rule object to the property `Rules`.

`ruleObj = addrule(modelObj, 'RuleValue', 'RuleTypeValue')` in addition to the assignments above, assigns a value (`RuleTypeValue`) to the property `RuleType`. For more information on the different types of rules see `RuleType`.

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

View additional rule properties with the function `get`, and modify rule properties with the function `set`. Copy a rule object to a model with the function `copyobj`, or delete a rule object from a model with the function `delete`.

Method Summary

Methods for rule objects

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

Property Summary

Properties for rule objects

Active	Property to indicate object use during a simulation
Annotation	Property with information about a SimBiology object
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object

Rule	Property to define certain species and parameter interactions
RuleType	Property for defining the type of rule for the rule object.
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Examples

Add a rule with default `RuleType`.

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

```
modelObj = sbiomodel('cell');
ruleObj = addrule(modelObj, '0.1*B-A')
```

2 Get a list of properties for a rule object.

```
get(modelObj.Rules(1)) or get(ruleObj)
```

MATLAB displays a list of rule properties.

```
    Active: 1
Annotation: ''
      Name: ''
     Notes: ''
    Parent: [1x1 SimBiology.Model]
      Rule: '0.1*B-A'
  RuleType: 'algebraic'
       Tag: ''
      Type: 'rule'
  UserData: []
```

Add rule with `RuleType` property set to `rate`.

1. Create model object, then add a reaction object

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

2. Add a rule which defines that the quantity of a species `c`. In the rule expression `k` is the rate constant for `a -> b`.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

3. Change the `RuleType` from default (`'algebraic'`) to `'rate'`. and verify using the `get` command.

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
     Active: 1
 Annotation: ''
       Name: ''
      Notes: ''
     Parent: [1x1 SimBiology.Model]
       Rule: 'c = k*(a+b)'
   RuleType: 'rate'
        Tag: ''
       Type: 'rule'
   UserData: []
```

See Also

`copyobj`, `delete`, `sbiomodel`

addspecies (model)

Purpose

Add species object to model object

Syntax

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

Arguments

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

Description

`speciesObj` `= addspecies(`*`modelObj`*`,` *`NameValue`*`)` creates a species object and returns a pointer (`speciesObj`) to the object. In the species object, this method assigns a value (*`NameValue`*) to the property `Name`, assigns the model object (*`modelObj`*) to the property `Parent`. In the model object, this method assigns the species object to the property `Species`.

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

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

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

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

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

`speciesObj = addspecies(...'PropertyName', PropertyValue...)`defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The property summary on this page shows the list of properties.

Valid Species Names

SimBiology species names can have any number or character, for example, `N-acetyl-D-glucosamine`.

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

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

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

Method Summary

Methods for species objects

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

Property Summary

Properties for species objects

Annotation	Property with information about a SimBiology object
BoundaryCondition	Property to set a species object to have a boundary condition
ConstantAmount	Property to specify variable or constant species amount
InitialAmount	Property containing initial amount of a species
InitialAmountUnits	Property containing units for species initial amount
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Examples

Add two species to a model, one is a reactant and the other is the enzyme catalyzing the reaction.

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

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

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

   ```
   speciesObj1 = addspecies (modelObj, 'glucose_6_phosphate');
   speciesObj2 = addspecies (modelObj, ...
                             'glucose_6_phosphate_dehydrogenase');
   ```

3. Set initial amount of `glucose_6_phosphate` to `100` and verify.

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

   MATLAB returns

   ```
   ans =

      100
   ```

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

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

   MATLAB returns,

   ```
   Species Object Array

   Index:  Name:                                 Initial   Initial
                                                 Amount:   AmountUnits:

    1      glucose_6_phosphate                   100
   ```

```
2      glucose_6_phosphate_dehydrogenase     0
```

5 Retrieve information about the first species in the array.

```
get(Mobj.Species(1))

           Annotation: ''
       ConstantAmount: 'false'
        InitialAmount: 100
   InitialAmountUnits: ''
                 Name: 'glucose_6_phosphate'
                Notes: ''
               Parent: [1x1 SimBiology.Model]
                  Tag: ''
                 Type: 'species'
             UserData: []
```

See Also

`addproduct`, `addreactant`, `addreaction`

MATLAB functions– `get` and `set`

Purpose

Copy SimBiology object and its children

Syntax

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

Arguments

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

Description

`copiedObj = copyobj(Obj, parentObj)` makes a copy of a SimBiology object (*`Obj`*) and returns a pointer to the copy (*`copiedObj`*). In the copied object (*`copiedObj`*), this method assigns a value (*`parentObj`*) to the property `Parent`.

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

`copiedObj = copyobj(modelObj)` makes a copy of a model object (*`modelObj`*) and returns the copy (*`copiedObj`*). In the copied model object (*`copiedObj`*), this method assigns the root object to the property `Parent`.

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

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

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

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

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


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

See Also `sbiomodel`, `sbioreaction`, `sbioroot`

Purpose

Delete SimBiology object

Syntax

`delete(Obj)`

Arguments

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

Description

`delete(Obj)` removes an object (*`Obj`*) from its parent.

- If *`Obj`* is a species object that is being used by a reaction object, this method returns an error and the species object is not deleted. You need to delete the reaction or remove the species from the reaction before you can delete the species object.
- If *`Obj`* is a parameter object being used by a kinetic law object, there is no warning when the object is deleted. However, when you try to simulate your model, a error occurs because the parameter cannot be found.
- If *`Obj`* is a reaction object, this method deletes the object, but the species objects that were being used by the reaction object are not deleted.
- If *`Obj`* is an abstract kinetic law object and there is a kinetic law object referencing it, this method returns an error.
- If *`Obj`* is a SimBiology configuration set object, and it is the active configuration set object, this method, after deleting the object, makes the default configuration set object active. Note, you cannot delete the default configuration set.
- You cannot delete the SimBiology `root`.

You can also delete all model objects from the root with one call to the `sbioreset` function.

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`, `sbioroot`, `sbioreset`

Purpose

Display summary of SimBiology object

Syntax

`display(Obj)`

Arguments

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

Description

Display the SimBiology object array. `display(Obj)` is called for the SimBiology object, *`Obj`* when the semicolon is not used to terminate a statement. The display of *`Obj`* gives a brief summary of *`Obj`* configuration. You can view a complete list of *`Obj`* properties with the command `get` . You can modify all *`Obj`* properties that can be changed, with the command `set`.

Examples

```
modelObj = sbiomodel('cell')
reactionObj = addreaction(modelObj, 'A + B -> C')
```

getadjacencymatrix (model)

Purpose

Return adjacency matrix from model object

Syntax

```
M = getadjacenceymatrix(modelObj)
M = getadjacencymatrix(modelObj,
'flat')
[M, Headings]
= getadjacenceymatrix(modelObj)
[M, Headings, Mask]
= getadjacenceymatrix(Obj)
```

Arguments

`M`	Adjacency matrix for *`modelObj`*
`modelObj`	specify model object `modelObj`
`'flat'`	Return adjacency matrix for only specified `modelObj` not for objects contained in the `modelObj`
`Headings`	Return row and column headings to *`Headings`*
`Mask`	Returns `1` for species object `0` for reaction object to *`Mask`*

Description

`getadjacencymatrix` returns adjacency matrix for model object.

`M = getadjacenceymatrix(modelObj)` returns adjacency matrix for model object, (*`modelOBJ`*) to *`M`*.

An adjacency matrix is defined by listing all species contained by *`modelObj`* and all reactions contained by *`modelObj`* column-wise and row-wise in a matrix. The reactants of the reactions are represented in the matrix with a 1 at the location of [row of species, column of reaction]. The products of the reactions are represented in the matrix with a 1 at the location of [row of reaction, column of species]. All other locations in the matrix are 0.

`M = getadjacencymatrix(modelObj, 'flat')` returns the adjacency matrix to *`M`* and defines the adjacency matrix for only *`modelObj`*. If *modelObj* is a SimBiology model then *M* is the adjacency matrix for the reactions and species contained by *modelObj*. *M* does not include any submodel reaction or species information.

`[M, Headings] = getadjacenceymatrix(modelObj)` returns the adjacency matrix to M and the row and column headings to *Headings*. *Headings* is defined by listing all `Name` property values of species contained by *modelObj* and all `Name` property values of reactions contained by *modelObj*. In the above example, *Headings* would be `{'A', 'B', 'C', 'R1'}`.

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

Examples

1. Read in a model using `sbmlimport`.

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

2. Get the adjacency matrix for the `modelObj`.

   ```
      [M, Headings] = getadjacencymatrix(modelObj)
   ```

See Also

`getstoichmatrix`

getconfigset (model)

Purpose

Get configuration set object from model object

Syntax

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

Arguments

`modelObj`	Model object. Enter a variable name for a model object.
`NameValue`	Name of the `configset` object.
`configsetObj`	Object holding the simulation specific information.

Description

`configsetObj = getconfigset(modelObj, 'NameValue')` returns the configuration set attached to `modelObj` that is named *`NameValue`*, to *`configsetObj`*.

`configsetObj = getconfigset(modelObj)` returns a vector of all attached configuration sets, to *`configsetObj`*.

`configsetObj = getconfigset(modelObj, 'active')` retrieves the active configuration set.

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

Use the `setactiveconfigset` function to define the active configset. *`modelObj`* always contains at least one configset object with name configured to 'default'. Additional configset objects can be added to *`modelObj`* with the method, `addconfigset` .

Example

1. Retrieve the default`configset` object from the `modelObj`.

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

```
configsetObj = getconfigset(modelObj)


 Configuration Settings - default (active)
     SolverType:           ode15s
     StopTime:             10.000000

   SolverOptions:
     AbsoluteTolerance:    1.000000e-006
     RelativeTolerance:    1.000000e-003

   RuntimeOptions:
     StatesToLog:          all

   CompileOptions:
     UnitConversion:       true
     DimensionalAnalysis:  true
```

2 Configure the `SolverType` to `ssa`.

```
set(configsetObj, 'SolverType', 'ssa')
get(configsetObj)


           Active: 1
   CompileOptions: [1x1 SimBiology.CompileOptions]
             Name: 'default'
            Notes: ''
   RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
    SolverOptions: [1x1 SimBiology.SSASolverOptions]
       SolverType: 'ssa'
         StopTime: 10
     StopTimeType: 'simulationTime'
        TimeUnits: 'second'
             Type: 'configset'
```

See Also

`addconfigset`, `removeconfigset`, `setactiveconfigset`

getparameters (kineticlaw)

Purpose

Get specific parameters in kinetic law object

Syntax

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

Arguments

kineticlawObj	Retrieve parameters used by kinetic law object.
ParameterVariablesValue	Retrieve parameters used by kinetic law object corresponding to the specified parameter in `ParameterVariables` property of the kinetic law object.

Description

`parameterObj = getparameters(kineticlawObj)` returns the parameters used by the kinetic law object `kineticlawObj` to `parameterObj`.

`parameterObj = getparameters(kineticlawObj, 'ParameterVariablesValue")` returns the parameter in the `ParameterVariableNames` property that corresponds to the parameter specified in the `ParameterVariables` property of `kineticlawObj`, to `parameterObj`. `ParameterVariablesValue` is the name of the parameter as it appears in the `ParameterVariablesproperty` of `kineticlawObj`. `ParameterVariablesValue` can be a cell array of strings.

If you change the name of a parameter you must configure all applicable elements such as rules that use the parameter, any user specified `ReactionRate`, or the kinetic law object property `ParameterVariableNames`. Use the method `setparameter` to configure `ParameterVariableNames`.

Example

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

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

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

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

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

3 Add two parameter objects.

```
parameterObj1 = addparameter(kineticlawObj,'Va');
parameterObj2 = addparameter(kineticlawObj,'Ka');
```

4 The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) that should to be set. To set these variables,

```
setparameter(kineticlawObj,'Vm', 'Va');
setparameter(kineticlawObj,'Km', 'Ka');
```

5 To retrieve a parameter variable,

```
parameterObj3 = getparameters(kineticlawObj, 'Vm')
```

MATLAB returns

```
Parameter Object Array

   Index:    Name:    Value:    ValueUnits:
   1         Va       1

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

See Also

`addparameter`, `getspecies`, `setparameter`.

getspecies (kineticlaw)

Purpose

Get specific species in kinetic law object

Syntax

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

Arguments

kineticlawObj	Retrieve species used by kinetic law object.
SpeciesVariablesValue	Retrieve species used by kinetic law object corresponding to the specified species in the `SpeciesVariables` property of the kinetic law object.

Description

`speciesObj = getspecies(kineticlawObj)` returns the species used by the kinetic law object `kineticlawObj` to `speciesObj`.

`speciesObj = getspecies(kineticlawObj, 'SpeciesVariablesValue')` returns the species in the `SpeciesVariableNames` property to `speciesObj`.

`SpeciesVariablesValue` is the name of the species as it appears in the `SpeciesVariables` property of `kineticlawObj`. `SpeciesVariablesValue` can be a cell array of strings.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the name of a species SimBiology updates the reaction to use the new name. You must however configure all other applicable elements such as rules that use the species, and the kinetic law object `SpeciesVariableNames`. Use the method `setspecies` to configure `SpeciesVariableNames`.

Example

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

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

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

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'` .

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

`reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has one species variable (`S`) that should to be set. To set this variable,

```
setspecies(kineticlawObj,'S', 'a');
```

4 Retrieve the species variable using `getspecies`.

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

MATLAB returns

```
Species Object Array

   Index:    Name:    InitialAmount:    InitialAmountUnits:
   1         a        0
```

See Also

`addspecies`, `setspecies`, `getparameters`, `setparameter`

getstoichmatrix (model)

Purpose Return stoichiometry matrix from model object

Syntax

```
M = getstoichmatrix(modelObj)
M = getstoichmatrix(modelObj,
'flat')
[M,objSpecies]=
getstoichmatrix(modelObj)
[M,objSpecies,objReactions]=
getstoichmatrix(modelObj)
```

Arguments

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

Description

`getstoichmatrix` returns a stoichiometry matrix for a model object.

`M = getstoichmatrix(modelObj)`returns a stoichiometry matrix for SimBiology a model object, (*`modelObj`*) to *`M`*.

A stoichiometry matrix is defined by listing all reactions contained by *`modelObj`* column-wise and all species contained by *`modelObj`* row-wise in a matrix. The species of the reaction are represented in the matrix with the stoichiometric value at the location of [row of species, column of reaction]. Reactants have negative values. Products have positive values. All other locations in the matrix are 0.

For example, if *`modelObj`* is a model object with two reactions with names `R1` and `R2` and Reaction values of : `2 A + B -> 3 C` and `B + 3 D -> 4 A`, the stoichiometry matrix would be defined as:

```
        A     B     C    D
  R1   -2    -1     3    0
  R2    4    -1     0   -3
```

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

`[M,objSpecies]= getstoichmatrix(modelObj)` returns the stoichiometry matrix to `M` and the species to `objSpecies`. `objSpecies` is defined by listing all `Name` property values of species contained by `Obj`. In the above example, `objSpecies` would be `{'A', 'B', 'C', 'D'}`;

`[M,objSpecies,objReactions]= getstoichmatrix(modelObj)` returns the stoichiometry matrix to `M` and the reactions to `objReactions`. `objReactions` is defined by listing all `Name` property values of reactions contained by `Obj`. In the above example, `ObjReactions` would be `{'R1', 'R2'}`.

Example

1. Read in a model using `sbmlimport`.

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

2. Get the stoichiometry matrix for the `modelObj`.

```
[M,objSpecies,objReactions] = getstoichmatrix(modelObj)
```

See Also

`getadjacencymatrix`

rmproduct (reaction)

Purpose

Remove species object from reaction object products

Syntax

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

Arguments

`reactionObj`	Reaction object.
`SpeciesName`	Name for a model object. Enter a species name or cell array of species names.
`speciesObj`	Species object. Enter a species object or an array of species objects.

Description

`rmproduct(`*`reactionObj`*`, `*`SpeciesName`*`)`, in a reaction object (`reactionObj`), removes a species object with a specified name (`SpeciesName`) from the property `Products`, removes the species name from the property `Reaction`, and updates the property `Stoichiometry` to exclude the species coefficient.

`rmproduct(`*`reactionObj`*`, `*`speciesObj`*`)` removes a species object as described above using a MATLAB variable for a species object.

The species object is not removed from the parent model property `Species`. If the species object is no longer used by any reaction, you can use the function `delete` to remove it from the parent object.

If one of the species specified does not exist as a product, a warning will be returned.

Examples

Example 1

Shows you how to remove a product that was added to a reaction by mistake. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP -> creatine + ATP + Pi');
rmproduct(reactionObj, 'Pi')
```

```
Reaction Object Array

Index:    Reaction:
1         Phosphocreatine + ADP -> creatine + ATP
```

Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A -> B + C');
reactionObj.Reaction

 ans =
    A -> B + C

rmproduct(reactionObj, modelObj.Species(2));
reactionObj.Reaction

  ans =
    A -> C
```

See Also

`rmreactant`

rmreactant (reaction)

Purpose

Remove species object from reaction object reactants

Syntax

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

Arguments

`reactionObj`	Reaction object.
`SpeciesName`	Name for a species object. Enter a species name or cell array of species names.
`speciesObj`	Species object. Enter a species object or an array of species objects.

Description

`rmreactant(`*`reactionObj`*`, `*`SpeciesName`*`)`, in a reaction object (`reactionObj`), removes a species object with a specified name (`SpeciesName`) from the property `Reactants`, removes the species name from the property `Reaction`, and updates the property `Stoichiometry` to exclude the species coefficient.

`rmreactant(`*`reactionObj`*`, `*`speciesObj`*`)`removes a species object as described above using a MATLAB variable for a species object, or a model index for a species object.

The species object is not removed from the parent model property `Species`. If the species object is no longer used by any reaction, you can use the method, `delete` to remove it from the parent object.

If one of the species specified does not exist as a reactant, a warning is returned.

Examples

Example 1

Shows you how to remove a reactant that was added to a reaction by mistake. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP + Pi -> creatine + ATP');
rmreactant(reactionObj, 'Pi')
```

```
Reaction Object Array

Index:    Reaction:
1         Phosphocreatine + ADP -> creatine + ATP
```

Example 2

Remove a species object using a model index to a species object.

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

reactionObj.Reaction

ans =
    A + B -> C

rmreactant(r, m.Species(2));
reactionObj.Reaction


ans =
    A -> C
```

See Also

`rmproduct`, `delete`

removeconfigset (model)

Purpose

Remove configuration set from model

Syntax

```
removeconfigset(modelObj,
'NameValue')
removeconfigset(modelObj, configsetObj)
```

Arguments

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

Description

`removeconfigset(`*`modelObj`*`, '`*`NameValue`*`')` removes the configset object with name, *`NameValue`* from SimBiology model object *`modelObj`*. A configuration set object stores simulation specific information. A SimBiology model can contain multiple configuration sets with one being active at any given time. The active configuration set contains the settings that are used during the simulation. *`modelObj`* always contains at least one configuration set object with name configured to `'default'`. The default configuration set cannot be removed from *`modelObj`*. If the active configuration set is removed from *`modelObj`* then the default configuration set will be made active.

`removeconfigset(`*`modelObj`*`, `*`configsetObj`*`)` removes the configuration set object, *`configsetObj`* from SimBiology model, *`modelObj`*. The configuration set is not deleted; if you want to delete `configsetObj` use the `delete` method.

If however, there is no MATLAB variable holding the configset, `removeconfigset(`*`modelObj`*`, '`*`NameValue`*`')`, removes the configset from the model and deletes it.

Example

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

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

2 Remove the configset from `modelObj` by name or alternatively by indexing.

```
% Remove the configset with name 'myset'.
  removeconfigset(modelObj, 'myset');

 % Get all configset objects and remove the second.
  configsetObj = getconfigset(modelObj);
  removeconfigset(modelObj, configsetObj(2));
```

See Also

`addconfigset`, `getconfigset`, `setactiveconfigset`

reset (root)

Purpose Delete all model objects from the root object

Syntax `reset(sbioroot)`

Description `reset(sbioroot)` deletes all SimBiology model objects contained by the SimBiology root. The SimBiology root object is returned with the method, `sbioroot` . This call is equivalent to `sbioreset`.

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

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

Examples

1 Query `sbioroot` that has two model objects.

```
sbioroot

   SimBiology Root Contains:

    Models:                          2
    Builtin Abstract Kinetic Laws:   3
    User Abstract Kinetic Laws:      1
    Builtin Units:                   54
    User Units:                      0
    Builtin Unit Prefixes:           13
    User Unit Prefixes:              0
```

2 Call `reset`.

```
sbioroot
```

```
SimBiology Root Contains:

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

See Also

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

sbioabstractkineticlaw (abstractkineticlaw)

Purpose

Construct abstract kinetic law object

Syntax

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

Arguments

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

Description

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

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

In order to use *`abstkineticlawObj`* when constructing a kinetic law object with the method `addkineticlaw`, *`abstkineticlawObj`* must be added to the user-defined library with the `sbioaddtolibrary` function. To get the abstract kinetic law objects in the user-defined library, use the command `get(sbioroot, 'UserDefinedKineticLaws')` `abstkineticlawObj = sbioabstractkineticlaw('`*`Name`*`', `*`’Expression’`*`)`, constructs a SimBiology abstract kinetic law object, *`abstkineticlawObj`* with name, *`Name`* and with expression, *`’Expression’`* and returns it to *`abstkineticlawObj`*.

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

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

Method Summary

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

Property Summary

Annotation	Property with information about a SimBiology object
Expression	Property containing the expression used to determine the reaction rate equation
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
ParameterVariables	Property showing parameters in abstract kinetic law

Parent	Property indicating the parent object
SpeciesVariables	Property showing species in abstract kinetic law
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Example

1. Create an abstract kinetic law.

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

2. Assign the parameter and species variables in the expression

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined

Abstract Kinetic Law Object Array

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

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

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A + B <-> B + C');
kineticlawObj = addkineticlaw(reactionObj, 'mylaw1');
```

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

See Also

`sbiomodel`, `addreaction`, `addkineticlaw`, `addparameter`

Purpose

Construct model object

Syntax

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

Arguments

NameValue	Required property to specify a unique name for a `model` object. Enter a character string.
PropertyName	Property name for a `Model` object from the Property Summary table below.
PropertyValue	Property value. Valid value for the specified property.

Description

modelObj = addmodel(*'NameValue'*) creates a model object and returns a pointer (*modelObj*)to the object. 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 the model as a submodel to another model object using the model object method `addmodel` or add as a submodel to a model object with the function `copyobj`. Add objects to a model object using the methods `addkineticlaw`, `addmodel`, `addparameter`, `addreaction`, `addrule`, and `addspecies`.

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

Method Summary

`addconfigset (model)`	Add configuration set object to model object
`addmodel (model)`	Add submodel object to model object
`addparameter (model, kineticlaw)`	Add parameter object to model or kinetic law object
`addreaction (model)`	Add reaction object to model object
`addrule (model)`	Add rule object to model object
`addspecies (model)`	Add species object to 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
`getadjacencymatrix (model)`	Return adjacency matrix from model object
`getconfigset (model)`	Get configuration set object from model object
`getstoichmatrix (model)`	Return stoichiometry matrix from model object
`removeconfigset (model)`	Remove configuration set from model
`setactiveconfigset (model)`	Set the active configuration set for model object

Property Summary

Annotation	Property with information about a SimBiology object
Models	Property showing all model objects
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parameters	Property with array of parameter objects
Parent	Property indicating the parent object
Reactions	Property with an array of reaction objects.
Rules	Property showing rules in model object
Species	Property showing species in model object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Examples

1. Create a SimBiology model object.

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

2. View help for the `modelObj` Tag property.

   ```
   sbiohelp('Tag')
   ```

3 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: []
```

4 Display summary of `modelObj` contents.

```
SimBiology Model - cell

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

See Also

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

MATLAB functions `set`, `get`

sbioparameter (parameter)

Purpose

Construct parameter object

Syntax

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

Arguments

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

Description

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

To use a parameter object (`paramaterObj`) in a simulation, you need to add the object to a SimBiology model, or kinetic law object with the method `copyobj`. You can use the `addparameter` method to simultaneously create and assign a parameter to a model or kinetic law object. SimBiology objects are constructed with the functions `sbiomodel`, `addmodel`, `addkineticlaw`, and `addreaction`

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

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

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

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

Method Summary

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

Property Summary

Annotation	Property with information about a SimBiology object
ConstantValue	Property to indicate variable or constant parameter value
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object

Parent	Property indicating the parent object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object
Value	Property to assign value to parameter object
ValueUnits	Property with parameter value units

Examples

1 Construct a parameter object.

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

2 View parameter object properties.

```
get(parameterObj)
```

MATLAB returns

```
        Annotation: ''
    ConstantValue: 1
             Name: 'kappa'
            Notes: ''
           Parent: [1x1 SimBiology.Reaction]
              Tag: ''
             Type: 'parameter'
         UserData: []
            Value: 4
       ValueUnits: '
```

See Also `addparameter`, `copyobj`, `sbiomodel`.

sbioreaction (reaction)

Purpose

Construct reaction object

Syntax

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

Arguments

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

Description

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

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

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

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

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

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

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

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

Method Summary

`addkineticlaw (reaction)`	Add kinetic law object to reaction object
`addproduct (reaction)`	Add product species object to reaction object
`addreactant (reaction)`	Add species object as a 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
`rmproduct (reaction)`	Remove species object from reaction object products
`rmreactant (reaction)`	Remove species object from reaction object reactants

Property Summary

Active	Property to indicate object use during a simulation
Annotation	Property with information about a SimBiology object
KineticLaw	Property showing kinetic law for `ReactionRate`
Name	Property with name of object

Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Products	Property to indicate reaction products
Reactants	Property to indicate reaction reactants.
Reaction	Property to indicate the reaction object reaction
ReactionRate	Property containing the reaction rate equation in reaction object
Reversible	Property to indicate whether a reaction is reversible or irreversible
Stoichiometry	Property that describes species coefficients in a reaction
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Examples

1. Construct reaction objects.

```
reactionObj1 = sbioreaction('a + 3 b -> 2 c');
reactionObj2 = sbioreaction({'a', 'b'}, [1 3], 'c', 2);

% View the help for the reaction object's Reversible property.
help(robj1, 'Reversible')
```

2 View the property summary for `reactionObj1`.

```
get(reactionObj1)
           Active: 1
       Annotation: ''
       KineticLaw: []
             Name: ''
            Notes: ''
       Parameters: [0x1 double]
           Parent: []
         Products: [1x1 SimBiology.Species]
        Reactants: [2x1 SimBiology.Species]
         Reaction: 'a + 3 b -> 2 c'
     ReactionRate: ''
       Reversible: 0
    Stoichiometry: [-1 -3 2]
              Tag: ''
             Type: 'reaction'
         UserData: []
```

See Also

`addreaction`,`sbiomodel`

Purpose

Return SimBiology root object

Syntax

```
rootObj = sbioroot
modelObj = sbioroot('modelName')
```

Arguments

`rootObj`	Return `sbioroot` to this object.
`modelObj`	Return the model with name *`modelName`* to this object.
`modelName`	Specify the name of the model that is on the root object.

Description

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

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

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

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

Method Summary

`copyobj (any object)`	Copy SimBiology object and its children
`delete (any object)`	Delete SimBiology object
`reset (root)`	Delete all model objects from the root object

Property Summary

BuiltInKineticLaws	Property containing built-in kinetic laws
BuiltInUnitPrefixes	Property containing built-in unit prefixes
BuiltInUnits	Property containing built-in units
Models	Property showing all model objects
Type	Property to indicate SimBiology object type
UserDefinedKineticLaws	Property containing user-defined kinetic laws
UserDefinedUnitPrefixes	Property containing user-defined unit prefixes
UserDefinedUnits	Property containing user-defined units

Examples

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

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

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

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

See Also

`sbiomodel`, `addkineticlaw`, `sbioregisterunit`, `sbiounregisterunit`, `sbioreset`

sbiorule (rule)

Purpose

Construct rule object

Syntax

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

Arguments

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

Description

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

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

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

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

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

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

Method Summary

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

Property Summary

Active	Property to indicate object use during a simulation
Annotation	Property with information about a SimBiology object
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Rule	Property to define certain species and parameter interactions
RuleType	Property for defining the type of rule for the rule object.
Tag	Property to specify a label for a SimBiology object

Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Examples

Example 1

Construct a rule object and copy to a model object.

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

Example 2

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

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

Example 3

List the properties for a rule.

```
get(robj)

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

See Also

`addrule`, `sbiomodel`, `copyobj`

Purpose

Construct species object

Syntax

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

Arguments

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

Description

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

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

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

See "Valid Species Names" on page 4-37 for more information on species names.

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

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

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

Method Summary

Methods for species objects.

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

Property Summary

Properties for species object

Annotation	Property with information about a SimBiology object
BoundaryCondition	Property to set a species object to have a boundary condition
ConstantAmount	Property to specify variable or constant species amount
InitialAmount	Property containing initial amount of a species
InitialAmountUnits	Property containing units for species initial amount
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object

Parent	Property indicating the parent object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Examples

Example 1

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

1. Create a species object with name `H2O` and initial amount 1000.

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

2. View properties for the species object.

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

Example 2

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

1 Create two species objects with the names `glucose_6_phosphate` and `glucose_6_phosphate_dehydrogenase`.

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

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

```
set(speciesObj(1), 'InitialAmount', 100);
get(speciesObj(1), 'InitialAmount')
```

MATLAB returns

```
ans =

   100
```

See Also

`addspecies`,

MATLAB functions`get` and `set`.

Purpose

Set the active configuration set for model object

Syntax

`configsetObj` `= setactiveconfigset(`*`modelObj`*`,`
`'`*`NameValue`*`')`
`configsetObj2` `= setactiveconfigset(`*`modelObj`*`, `*`configsetObj1`*`)`

Description

`configsetObj` `= setactiveconfigset(`*`modelObj`*`, '`*`NameValue`*`')` sets the configuration set *`NameValue`* to be the active configuration set for the model *`modelObj`* and returns to *`configsetObj`*.

`configsetObj2` `= setactiveconfigset(`*`modelObj`*`, `*`configsetObj1`*`)` sets the `configset` *`configsetObj1`* to be the active `configset` for *`modelObj`* and returns to *`configsetObj2`*. Any change in one of these two configset objects *`configsetObj1`* and *`configsetObj2`* is reflected in the other. To copy over a `configset` object from one model object to another use the `copyobj` method.

The active configuration set contains the settings that are be used during a simulation. A default configuration set is attached to any new model.

Examples

1. Create a model object by importing the file `oscillator.xml` and add a configset that simulates for 3000 seconds.

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

2. Configure the `configsetObj StopTime` to `3000`.

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


          Active: 0
  CompileOptions: [1x1 SimBiology.CompileOptions]
            Name: 'myset'
           Notes: ''
```

```
 RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
  SolverOptions: [1x1 SimBiology.ODESolverOptions]
     SolverType: 'ode15s'
       StopTime: 3000
   StopTimeType: 'simulationTime'
      TimeUnits: 'second'
           Type: 'configset'
```

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

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

See Also `addconfigset`, `getconfigset`, `removeconfigset`

setparameter (kineticlaw)

Purpose

Specify specific parameters in kinetic law object

Syntax

`setparameter(`*`kineticlawObj`*`,`
`'ParameterVariablesValue', 'ParameterVariableNamesValue')`

Arguments

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

Description

Configure `ParameterVariableNames` in kinetic law object.

`setparameter(`*`kineticlawObj`*`, 'ParameterVariablesValue', 'ParameterVariableNamesValue')` configures the `ParameterVariableNames` property of the kinetic law object (`kineticlawObj`). `ParameterVariableValue` corresponds to one of the strings in `kineticlawObj` `ParameterVariables` property. The corresponding element in`kineticlawObj``ParameterVariableNames` property is configured to `ParameterVariableNamesValue`.
For example, if `ParameterVariables` is `{'Vm', 'Km'}` and `ParameterVariablesValue` is specified as `Vm`, then the first element of the `ParameterVariableNames` cell array is configured to `ParameterVariableNamesValue`.

Example

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

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

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

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

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

`reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) that should be set. To set these variables,

```
setparameter(kineticlawObj,'Vm', 'Va');
setparameter(kineticlawObj,'Km', 'Ka');
```

4 Verify that the parameter variables are correct.

```
get (kineticlawObj, 'ParameterVariableNames')
```

MATLAB returns

```
ans =

    'Va'    'Ka'
```

See Also `addparameter`, `getspecies`, `setspecies`

Purpose

Specify species in kinetic law object

Syntax

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

Arguments

`SpeciesVariablesValue`	Specify species variable in kinetic law object.
`SpeciesVariableNamesValue`	Specify the species name with which to configure species variable in kinetic law object. Determines species in `ReactionRate` equation

Description

`setspecies` configures kinetic law object `SpeciesVariableNames` property.

`setspecies(kineticlawObj, 'SpeciesVariablesValue', 'SpeciesVariableNamesValue')` configures the `SpeciesVariableNames` property of the kinetic law object, `kineticlawObj`. `SpeciesVariablesValue` corresponds to one of the strings in `SpeciesVariables` property of `kineticlawObj`. The corresponding element in `kineticlawObj` `SpeciesVariableNames` property is configured to `SpeciesVariableNamesValue`.

For example, if `SpeciesVariables are{'S', 'S1'}` and `SpeciesVariablesValue` is specified as `S1`, the first element of the `SpeciesVariableNames` cell array is configured to `SpeciesVariableNamesValue`.

Example

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

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

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

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 SimBiology method `addparameter` `getspecies`, `setparameter`

5

Properties – Categorical List

This chapter is a reference for the object properties in SimBiology. Properties are grouped into the following categories.

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

Abstract Kinetic Law

Properties for abstract kinetic law objects

Annotation	Property with information about a SimBiology object
Expression	Property containing the expression used to determine the reaction rate equation
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
ParameterVariables	Property showing parameters in abstract kinetic law
Parent	Property indicating the parent object
SpeciesVariables	Property showing species in abstract kinetic law
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Configuration Sets

Properties for configuration set objects.

Active	Property to indicate object use during a simulation
CompileOptions	Property holding dimensional analysis and unit conversion information
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
RuntimeOptions	Property holding options for logged species
SolverOptions	Property holding the model solver options
SolverType	Property to select solver type for simulation
StopTime	Property to set the stop time for a simulation
StopTimeType	Property to specify the type of stop time for a simulation
TimeUnits	Property to show the stop time units for a simulation
Type	Property to indicate SimBiology object type

Kinetic Laws

Properties for kinetic law object

Annotation	Property with information about a SimBiology object
Expression	Property containing the expression used to determine the reaction rate equation
KineticLawName	Property showing name of abstract kinetic law
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parameters	Property with array of parameter objects
ParameterVariableNames	Property showing cell array of reaction rate parameters
ParameterVariables	Property showing parameters in abstract kinetic law
Parent	Property indicating the parent object
SpeciesVariables	Property showing species in abstract kinetic law
SpeciesVariablesNames	Property showing cell array of species used in reaction rate equation
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Models

Properties for model objects

Annotation	Property with information about a SimBiology object
Models	Property showing all model objects
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parameters	Property with array of parameter objects
Parent	Property indicating the parent object
Reactions	Property with an array of reaction objects.
Rules	Property showing rules in model object
Species	Property showing species in model object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Parameters

Properties for parameter objects

Annotation	Property with information about a SimBiology object
ConstantValue	Property to indicate variable or constant parameter value
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object
Value	Property to assign value to parameter object
ValueUnits	Property with parameter value units

Reactions

Properties for reaction objects

Active	Property to indicate object use during a simulation
Annotation	Property with information about a SimBiology object
KineticLaw	Property showing kinetic law for `ReactionRate`
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Products	Property to indicate reaction products
Reactants	Property to indicate reaction reactants.
Reaction	Property to indicate the reaction object reaction
ReactionRate	Property containing the reaction rate equation in reaction object
Reversible	Property to indicate whether a reaction is reversible or irreversible
Stoichiometry	Property that describes species coefficients in a reaction
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Root

Properties for the root object

BuiltInKineticLaws	Property containing built-in kinetic laws
BuiltInUnitPrefixes	Property containing built-in unit prefixes
BuiltInUnits	Property containing built-in units
Models	Property showing all model objects
Type	Property to indicate SimBiology object type
UserDefinedKineticLaws	Property containing user-defined kinetic laws
UserDefinedUnitPrefixes	Property containing user-defined unit prefixes
UserDefinedUnits	Property containing user-defined units

Rules

Properties for rule objects

Active	Property to indicate object use during a simulation
Annotation	Property with information about a SimBiology object
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Rule	Property to define certain species and parameter interactions
RuleType	Property for defining the type of rule for the rule object.
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Species

Properties for species objects

Annotation	Property with information about a SimBiology object
BoundaryCondition	Property to set a species object to have a boundary condition
ConstantAmount	Property to specify variable or constant species amount
InitialAmount	Property containing initial amount of a species
InitialAmountUnits	Property containing units for species initial amount
Name	Property with name of object
Notes	Property with HTML text describing SimBiology object
Parent	Property indicating the parent object
Tag	Property to specify a label for a SimBiology object
Type	Property to indicate SimBiology object type
UserData	Property to specify data to associate with object

Using Object Properties

Command line syntax for entering and retrieving property values.

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

Entering property values

Enter or change a single property value using dot notation.

```
ObjectName.PropertyName = PropertyValue
```

Enter or change one or more property values using the MATLAB function set.

```
set(ObjectName, 'Propertyname', PropertyValue, ...)
```

Retrieving property values

Retrieve a single property value using dot notation.

```
PropertyValue = ObjectName.PropertyName
```

Retrieve one or more property values using the MATLAB function get.

```
PropertyValue(s) = get(ObjectName, 'PropertyName', ...)
```

Retrieve one or more property values using the object method get.

```
PropertyValue(s) = ObjectName.get('PropertyName', ...)
```

List or retrieve all property values using one of the following commands.

```
get(ObjectName)
AllPropertyValues = get(ObjectName)
```

```
ObjectName.get
```

Help for Objects, Methods and Properties

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

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

6

Properties — Alphabetical List

Purpose Property to specify largest allowable absolute error

Description `AbsoluteTolerance` specifies the largest allowable absolute error at any step in simulation. It is a property of `SolverOptions` object. `SolverOptions` is a property of the `configset` object. `AbsoluteTolerance` is available for the ode solvers (`'ode45'`, `'ode23'`, `'ode113'`, `'ode15s'`, `'ode23s'`, `'ode23t'`, and `'ode23tb'`).

At each simulation step, the solver estimates the local error e_i in the i^{th} state vector `y`. Simulation converges at that time step if e_i satisfies the following equation:

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

Thus at higher state values, convergence is determined by `RelativeTolerance`. As the state values approach zero, convergence is controlled by `AbsoluteTolerance`. The choice of values for `RelativeTolerance` and `AbsoluteTolerance` will vary depending on the problem. The default values should work for first trials of the simulation; however if you want to optimize the solution, consider that there is a trade-off between speed and accuracy. If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy. If the results appear to be inaccurate you can decrease the tolerance values but this will slow down the solver. If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

This may be important for reactions where species values tend to zero. Even if you are not interested in the value of a state `y(i)` when it is small, you may have to specify `AbsoluteTolerance` small enough to get some correct digits in `y(i)` so that you can accurately compute more interesting state values.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`

Data values	>0, <1; default is `1e-6`
Access	Read/Write

Example

This example shows how to change `AbsoluteTolerance`.

1. Retrieve the `configset` object from the `modelObj`.

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

2. Change the `AbsoluteTolerance` to `1e-8`.

```
set(configsetObj.SolverOptions, 'AbsoluteTolerance', 1.0e-8)
get(configsetObj.SolverOptions, 'AbsoluteTolerance')


ans =

  1.0000e-008
```

See Also

`RelativeTolerance`

Active

Purpose

Property to indicate object use during a simulation

Description

Indicates whether a simulation is using a SimBiology object. A SimBiology model is organized into a hierarchical group of objects. Use the Active property to include or exclude objects during a simulation. When a reaction or rule object Active property is set to be `false`, the simulation does not include the reaction or rule. This is a convenient way to test a model with and without a reaction or rule. For `configset` object, use the method `setactiveconfigset`, to set the object Active property to `true`.

Characteristics

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

Example

1. Create a model object.

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

2. Add reaction object and verify that the `Active` property setting is `'true'` or `1`.

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

MATLAB returns

```
ans =

1
```

3. Set Active property to `'false'` and verify.

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

MATLAB returns

```
ans =

0
```

See Also `addreaction`, `addrule`, `setactiveconfigset`, `addconfigset`

Annotation

Purpose Property with information about a SimBiology object

Description URL or filename linking to information about a model.

Characteristics

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

Example

1. Create a model object

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

2. Set annotation for model object

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

3. Verify the assignment.

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

   MATLAB returns

   ```
   ans =

   www.reactome.org
   ```

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

Purpose

Property to set a species object to have a boundary condition

Description

Indicates whether a species object has a boundary condition. If `BoundaryCondition` is `true`, the species quantity is determined by `InitialAmount` and/or a rule object, and not by the reaction rate equation. In SimBiology, all species are state variables regardless of `BoundaryCondition` or `ConstantAmount` property.

By default `BoundaryCondition` is `false` and SimBiology uses reaction rate equations to determine the rate of change of a species quantity in the model. Boundary condition is used when a species is modeled as a participant of reactions but the species quantity is not determined by a reaction rate equation. Consider the following two use cases of boundary conditions:

- Modeling receptor-ligand interactions that affect the rate of change of the receptor but not the ligand. For example, in response to hormone, steroid receptors such as the glucocorticoid receptor (GR) translocate from the cytoplasm (`cyt`) to the nucleus (`nuc`). The hsp90/hsp70 chaperone complex directs this nuclear translocation [Pratt 2004]. The natural ligand for GR is cortisol; the synthetic hormone dexamethasone (`dex`) is used in place of cortisol in experimental systems. In this system dexamethasone participates in the reaction but the quantity of dexamethasone in the cell is regulated using a rule. To simply model translocation of GR you could use the following reactions:

  Formation of the chaperone–receptor complex,

  ```
  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 SimBiology as

```
dex = 0.001
```

- Modeling the role of nucleotides (for example, GTP, ATP, cAMP) and cofactors (for example, Ca^{++}, NAD^{+}, coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.

  ```
  Ras-GDP + GTP  -> Ras-GTP + GDP

  For GTP, BoundaryCondition = true; ConstantAmount = true
  ```

  Model GTP and GDP with boundary conditions, thus making them *boundary species*. In addition you can set the `ConstantAmount` property of these species to `true` to indicate that their quantity does not vary during a simulation.

Characteristics

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

Example

1. Create a model object

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

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

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

   MATLAB returns

   ```
   ans =

        0
   ```

3. Set boundary condition to `'true'` and verify

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

   MATLAB returns

   ```
   ans =

        1
   ```

References

Pratt, W.B., Galigniana, M.D., Morishima, Y., Murphy, P.J. (2004), Role of molecular chaperones in steroid receptor action, *Essays Biochem*, 40:41-58.

See Also

`addrule`, `addspecies`, `ConstantAmount` `InitialAmount`

BuiltInKineticLaws

Purpose

Property containing built-in kinetic laws

Description

`BuiltInKineticLaws` is a SimBiology root object property showing all abstract kinetic laws that are shipped with SimBiology. Use the command `sbiowhos -builtin -kineticlaw` to see the list of built-in kinetic laws. You can use built-in kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example:

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

You cannot add, modify, or delete `BuiltInKineticLaws`.

See "Abstract Kinetic Law" on page 6-27 for a definition and more information.

Characteristics

Applies to	Object: root
Data type	`char string` of valid abstract kinetic law name.
Data values	Valid kinetic laws
Access	Read-only

Examples

Example 1

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

```
sbiowhos -builtin -kineticlaw


Abstract Kinetic Law Object Array

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

```
3         BuiltIn     Henri-Michaelis-Menten    Vm*S/(Km + S
```

Example 2

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

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


Abstract Kinetic Law Object Array

Index:    Library:    Name:                     Expression:
1         BuiltIn     Unknown                   Unknown
2         BuiltIn     MassAction                MassAction
3         BuiltIn     Henri-Michaelis-Menten    Vm*S/(Km + S
```

See Also

`UserDefinedKineticLaws`, `BuiltInUnits`, `BuiltInUnitPrefixes`

MATLAB functions `get` and `set`

BuiltInUnitPrefixes

Purpose Property containing built-in unit prefixes

Description `BuiltInUnitPrefixes` is a SimBiology root object property showing all unit prefixes that are shipped with SimBiology. You can specify units with prefixes for species amounts and parameter values, because, SimBiology enables you to do dimensional analysis and unit conversion during simulation. The valid units and unit prefixes are either built-in or user-defined. You can display the built-in unit prefixes either by using the command `sbiowhos`, or by accessing the root object. Both methods are illustrated in the examples below.

You cannot add, modify, or delete `BuiltInUnitsPrefixes`.

Characteristics

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

Examples

Example 1

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

```
sbiowhos -builtin -unitprefix
```

Example 2

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

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

See Also

`BuiltInUnitPrefixes`, `UserDefinedUnits`, `BuiltInKineticLaws`

MATLAB functions `get` and `set`.

Purpose

Property containing built-in units

Description

`BuiltInUnits` is a SimBiology root object property showing all units that are shipped with SimBiology. You can specify units for species amounts and parameter values, because, SimBiology enables you to do dimensional analysis and unit conversion during simulation. The valid units are either built-in or user-defined. You can display the built-in units either by using the command `sbioswhos`, or by accessing the root object. Both methods are illustrated in the examples below.

You cannot add, modify, or delete `BuiltInUnits`.

Characteristics

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

Examples

Example 1

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

```
sbiowhos -builtin -units
```

Example 2

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

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

See Also

`BuiltInUnitPrefixes`, `UserDefinedUnits`, `BuiltInKineticLaws`

MATLAB functions `get` and `set`.

CompileOptions

Purpose

Property holding dimensional analysis and unit conversion information

Description

The SimBiology `CompileOptions` object defines the compile options available for simulation; you can specify whether dimensional analysis and unit conversion is necessary for simulation. Compile options are checked during compile time. The compile options object can be accessed through the `CompileOptions` property of the `configset` object. Retrieve `CompileOptions` object properties with the `get` function and configure the properties with the `set` function.

Property Summary

DimensionalAnalysis	Property to indicate whether to perform dimensional analysis
Type	Property to indicate SimBiology object type
UnitConversion	Property to indicate whether to perform unit conversion.

Characteristics

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

Example

1 Retrieve the `configset` object of `modelObj`

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

2 Retrieve the `CompileOptions` object (`optionsObj`) from the `configsetObj`

```
optionsObj = get(cs, 'CompileOptions');
```

```
DimensionalAnalysis: 1
Type: 'compileoptions'
UnitConversion: 1
```

See Also MATLAB functions `get`, `set`

ConstantAmount

Purpose

Property to specify variable or constant species amount

Description

`ConstantAmount` indicates whether the quantity of the species object can vary during the simulation. ConstantAmount can be either `true` or `false`. If ConstantAmount is `true`, the quantity of the species cannot vary during the simulation. By default, ConstantAmount is `false` and the quantity of the species can vary during the simulation. If ConstantAmount is `false`, the quantity of the species can be determined by reactions and rules.

The following is example of modeling species as constant amounts:

Modeling the role of nucleotides (GTP, ATP, cAMP) and cofactors (Ca^{++}, NAD^{+}, coenzyme A. Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.

```
Ras-GDP + GTP  -> Ras-GTP + GDP
```

Model GTP and GDP with constant amount set to `true`. In addition, you can set the `BoundaryCondition` of these species to `true`, thus making them *boundary species*.

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

Characteristics

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

Example

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

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

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

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

MATLAB returns

```
          ans =

     0
```

3 Set constant amount to `'true'` and verify

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

MATLAB returns

```
          ans =

     1
```

See Also

`addspecies`, `BoundaryCondition`

ConstantValue

Purpose

Property to indicate variable or constant parameter value

Description

Indicates whether the value of a parameter can change during a simulation. Enter either `true` (value is constant) or `false` (value can change).

You can allow the value of the parameter to change during a simulation by specifying a rule that changes the `Value` property of the parameter object. For example, consider feedback inhibition of an enzyme such as aspartate kinase by threonine. Aspartate kinase has three isozymes that are independently inhibited by the products of downstream reactions (threonine, homoserine, and lysine). Although threonine is made through a series of reactions in the synthesis pathway, for illustration the reactions are simplified as follows:

$$\textit{Aspartic}\ \text{acid} \xrightarrow{\text{aspartate kinase}} \beta - \textit{Aspartylphosphate}$$

$$\beta - \textit{Aspartylphosphate} \longrightarrow \textit{Threonine}$$

To model inhibition of aspartate kinase by threonine you could use a rule like the `algebraic` rule below to vary the rate of the above reaction and simulate inhibition. In the rule, the rate constant for the above reaction is denoted by `k_aspartate_kinase` and the quantity of threonine is `threonine`.

```
k_aspartate_kinase -(1/threonine)
```

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

Characteristics

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

Example

1. Create a model object.

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

2. Add parameter object.

   ```
   parameterObj = addparameter (modelObj, 'kf');
   ```

3. Change the `ConstantValue` property of the parameter object from default (`true`) to `false` and verify.

   MATLAB returns `1` for `true` and `0` for `false`.

   ```
   set (parameterObj, 'ConstantValue', false)
   get(parameterObj, 'ConstantValue')
   ```

   MATLAB returns

   ```
   ans =

        0
   ```

See Also

`addparameter`

DimensionalAnalysis

Purpose

Property to indicate whether to perform dimensional analysis

Description

`DimensionalAnalysis` specifies whether to perform dimensional analysis on the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object. When `DimensionalAnalysis` is set to `true`, SimBiology checks whether the physical quantities of the units involved in reactions and rules, match and are applicable.

For example, consider a reaction `a + b > c`. Using mass action kinetics, the reaction rate is defined as `a*b*k` where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are 0.01M and 0.005M respectively, then units of `k` are `1/(M*second)`. If you specify `k` with another equivalent unit definition, for example `1/[(moles/liter)*second]`, `DimensionalAnalysis` checks whether the physical quantities match. If the physical quantities do not match, you see an error and the model is not simulated, `UnitConversion` is the next step after `DimensionalAnalysis`.

Valid physical quantities for reaction rates are amount/time, mass/time or concentration/time.

Characteristics

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

Example

Shows how to retrieve and set `DimensionalAnalysis` from the default `true` to `false` in the default configuration set in a model object.

1. Import a model.

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

```
SimBiology Model - Oscillator

   Model Components:
     Models:            0
     Parameters:        0
     Reactions:         42
     Rules:             0
     Species:           23
```

2 Retrieve the `configset` object of the model object.

```
configsetObj = getconfigset(modelObj)

 Configuration Settings - default (active)
     SolverType:            ode15s
     StopTime:              10.000000

   SolverOptions:
     AbsoluteTolerance:     1.000000e-006
     RelativeTolerance:     1.000000e-003

   RuntimeOptions:
     StatesToLog:           all

   CompileOptions:
     UnitConversion:        true
     DimensionalAnalysis:   true
```

3 Retrieve the `CompileOptions` object.

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

Compile Settings:

     UnitConversion:        true
     DimensionalAnalysis:   true
```

4 Assign a value of false to `DimensionalAnalysis`.

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

See Also `getconfigset`, `sbiosimulate`

MATLAB functions `get` and `set`.

Purpose

Property specifies explicit or implicit tau error tolerance

Description

`ErrorTolerance` 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 an error. If the error tolerance is small, the solver will take more steps than when the error tolerance is large leading to longer simulation times. The error tolerance should be adjusted depending upon the problem, but a good value for the error tolerance is between `1 %` to `5 %`.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`
Data values	>0, <1; default is `3e-2`
Access	Read/Write

Example

Shows how to change `ErrorTolerance` settings.

1. Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `expltau`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
set(configsetObj, 'SolverType', 'expltau')
```

2. Change the `ErrorTolerance` to `1e-8`.

```
set(configsetObj.SolverOptions, 'ErrorTolerance', 5.0e-2)
get(configsetObj.SolverOptions, 'ErrorTolerance')


ans =

  5.000000e-002
```

See Also `LogDecimation`, `RandomState`

Purpose

Property containing the expression used to determine the reaction rate equation

Description

Indicates the expression that is used to determine the `ReactionRate` property of the reaction object. `Expression` is a reaction rate expression assigned by the abstract kinetic law used by the kinetic law object. The abstract kinetic law being used is indicated by the property `KineticLawName`. You can configure `Expression` for user-defined abstract kinetic laws but not for builtin abstract kinetic laws. `Expression` in read-only for kinetic law objects.

Abstract Kinetic Law

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

If a reaction is using an abstract kinetic law, the `ReactionRate` property of the reaction object shows the result of a mapping from an abstract kinetic law. To determine `ReactionRate` the species variables and parameter variables that participate in the reaction rate should be clearly mapped in the kinetic law for the reaction. In this case SimBiology determines the `ReactionRate` by using the`Expression` property of the abstract kinetic law object, and by mapping `SpeciesVariableNames` to `SpeciesVariables` and `ParameterVariableNames` to `ParameterVariables`.

For example, the abstract kinetic law `Henri-Michaelis-Menten` has the `Expression` Vm*[S]/(Km + [S]) , where `Vm` and `Km` are defined as parameters in the `ParameterVariables` property of the abstract kinetic law object, and `S` is defined as a species in the `SpeciesVariable` property of the abstract kinetic law object.

By applying the abstract kinetic law `Henri-Michaelis-Menten` to a reaction `A -> B` with `Va` mapping to `Vm` and `A` mapping to `S` the rate equation for the reaction becomes Va*[A]/(Ka + [A]).

The exact expression of a reaction using `MassAction` kinetic law varies depending upon the number of reactants. Thus, for mass action kinetics the `Expression` property is set to `MassAction` because In general for mass action kinetics the reaction rate is defined as

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

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

SimBiology comes with some built-in kinetic laws. Users can also define their own abstract kinetic laws. To find the list of available kinetic laws, use the `sbiowhos -kineticlaw` command (`sbiowhos`). You can create an abstract kinetic law with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

Characteristics

Applies to	Objects: kineticlaw, abstract kineticlaw
Data type	`char string`
Data values	Defined by abstract kinetic law
Access	Read-only in kinetic law object. Read/Write in user-defined abstract kinetic law.

Examples

Example with `Henri-Michaelis-Menten` kinetics

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

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

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

2 Define a kinetic law for the reaction object

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

3 Verify that the `Expression` property for the kinetic law object is `Henri-Michaelis-Menten`

```
get (kineticlawObj, 'Expression')
```

MATLAB returns

```
ans =

Vm*[S]/(Km + [S])
```

4 The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) and one species variable (`S`) that you should set. To set these variables, first create the parameter variables as parameter objects (`parameterObj1, parameterObj2`) with names `Vm_d, Km_d`, and assign the objects `Parent` property value to the `kineticlawObj`. The species object with `Name`,a is created when `reactionObj`is created and need not be redefined.

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

5 Set the variable names for the kinetic law object

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

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

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns

```
ans =

Vm_d*[a]/(Km_d+[a])
```

Example with `Mass Action` kinetics.

1. Create a model object, then add a rection object

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

2. Define a kinetic law for the reaction object

   ```
   kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
   get(kineticlawObj, 'Expression')
   ```

   MATLAB returns

   ```
   ans =

   MassAction
   ```

3. Assign the rate constant for the reaction.

   ```
   set (kineticlawObj, 'ParameterVariablenames', 'k');

   get (reactionObj, 'ReactionRate')
   ```

   MATLAB returns

   ```
   ans =

   [k]*[a]*[b]
   ```

See Also

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

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

Reaction object property: `ReactionRate`

Functions: `sbioaddtolibrary`,`sbiowhos`

InitialAmount

Purpose

Property containing initial amount of a species

Description

Indicates the initial quantity of the SimBiology species object. `InitialAmount` is the quantity of the species before the simulation starts.

Characteristics

Applies to	Object: species
Data type	`double`
Data values	Positive real number. Default value is `0`.
Access	Read/Write

Example

Add a species with name and value to a model object.

1. Create a model object with named `my_model`.

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

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

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

3. Set the initial amount to `100` and verify.

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

   MATLAB returns

   ```
   ans =

      100
   ```

See Also

`addspecies`, `InitialAmountUnits`

Purpose Property containing units for species initial amount

Description Indicates the unit definition for the `InitialAmount` property of a species object. `InitialAmountUnits` can be one of the builtin units. To get a list of the defined units use the `sbioshowunits` function. If `InitialAmountUnits` changes from one unit definition to another, the `InitialAmount` does not automatically convert to the new units. The `sbioconvertunits` function does this conversion. To add a user-defined unit to the list see `sbioregisterunit`.

Characteristics

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

Example

1. Create a model object named `my_model`.

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

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

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

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

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

MATLAB returns

```
ans =
```

`molecule`

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

Purpose

Property showing kinetic law for `ReactionRate`

Description

`KineticLaw` defines the kinetics used to determine the reaction rate that is specified in the `ReactionRate` property of the reaction object. This property shows the kinetic law used to define `ReactionRate`.

KineticLaw can be configured with the `addkineticlaw` method. The `addkineticlaw` function configures the `ReactionRate` based on the `KineticLaw` and the species and parameters specified in the kinetic law object properties `SpeciesVariableNames` and `ParameterVariableNames`. `SpeciesVariableNames` are determined automatically for mass action kinetics.

If the reaction is updated, the `ReactionRate` is automatically updated only for mass action kinetics. For all other kinetics the `SpeciesVariableNames` property of the kinetic law object should be reconfigured.

Characteristics

Applies to	Object: reaction
Data type	Kinetic law object
Data values	Kinetic law object. Default is empty (`[]`).
Access	Read-only

Example

Example with `Henri-Michaelis-Menten` kinetics

1 Create a model object, then add a reaction object

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

2 Define a kinetic law for the reaction object

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

3. Verify that the `KineticLaw` property for the reaction object is `Henri-Michaelis-Menten`

   ```
   get (reactionObj, 'KineticLaw')
   ```

   MATLAB returns

   ```
   Kinetic Law Object Array

       Index:    KineticLawName:
       1         Henri-Michaelis-Menten
   ```

See Also

`KineticLawName`, `Parameters`, `ParameterVariableNames`, ReactionRate, `SpeciesVariableNames`

KineticLawName

Purpose

Property showing name of abstract kinetic law

Description

Indicates the name of the abstract kinetic law in the kinetic law object. `KineticLawName` can be any valid name from the builtin or user-defined abstract kinetic law library. See "Abstract Kinetic Law" on page 6-27 for a definition and more information.

You can find the `KineticLawName` list in the abstract kinetic law library by using the command `sbiowhos -kineticlaw` (`sbiowhos`). You can create an abstract kinetic law with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

Characteristics

Applies to	Object: kineticlaw
Data type	`char string`
Data values	`char string` defined by abstract kinetic law
Access	Read-only

Examples

1. Create a model object, add a reaction object, and define a kinetic law for the reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

2. Verify `KineticLawName` of `kineticlawObj`

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

MATLAB returns

```
ans =

Henri-Michaelis-Menten
```

See Also

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

Kinetic law object properties: `SpeciesVariableNames`, `ParameterVariableNames`, , `sbioaddtolibrary`, `sbiowhos`

Purpose

Property to specify recorded simulation output frequency

Description

`LogDecimation` defines how often the simulation data is recorded as output. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object. `LogDecimation` is available for `ssa`, `expltau`, and `inmpltau` solvers.

Use `LogDecimation` to specify how frequently you want to record the output of the simulation. For example, if the `LogDecimation` is set to 1, for the command `(t,x) = sbiosimulate(modelObj)`, at each simulation step the time will be logged in `t` and the quantity of each logged species will be logged as a row in `x`. If `LogDecimation` is 10, then every 10th simulation step will be logged in `t` and `x`.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`int`
Data values	>0 default is `1`.
Access	Read/Write

Example

Shows how to change `LogDecimation` settings.

1. Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `expltau`

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
set(configsetObj, 'SolverType', 'expltau')
```

2. Change the `LogDecimation` to `10`.

```
set(configsetObj.SolverOptions, 'LogDecimation', 10)
get(configsetObj.SolverOptions, 'LogDecimation')
```

```
ans =

  10
```

See Also `ErrorTolerance`, `RandomState`

Purpose

Property to specify nonlinear solver maximum iterations in implicit tau

Description

`MaxIterations` specifies the maximum number of iterations for the nonlinear solver in `impltau`. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object.

The implicit tau solver in SimBiology internally uses a nonlinear solver to solve a set of algebraic nonlinear equations at every simulation step. Starting with an initial guess at the solution, the nonlinear solver iteratively tries to find the solution to the algebraic equations. The closer the initial guess is to the solution, the fewer the iterations the nonlinear solver will take before it finds a solution. `MaxIterations` specifies the maximum number of iterations the nonlinear solver should take before it issues a "failed to converge" error. If you get this error, during simulation try increasing `MaxIterations`. The default value of `MaxIterations` is `15`.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`int`
Data values	>0 default is `15`.
Access	Read/Write

Example

Shows how to change `MaxIterations` settings.

1 Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `impltau`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
set(configsetObj, 'SolverType', 'impltau')
```

2 Change the `MaxIterations` to `25`.

```
set(configsetObj.SolverOptions, 'MaxIterations', 25)
get(configsetObj.SolverOptions, 'MaxIterations')
```

```
ans =

  25
```

See Also `ErrorTolerance`, `LogDecimation`, `RandomState`

Purpose

Property showing all model objects

Description

Indicates the models in a `Model` object or in the SimBiology root. Read-only array of `Model` objects. SimBiology has a hierarchical organization. A top-level model object has the SimBiology root as its `Parent`. Model objects with another model object as `Parent` are submodels. For a model object to access configset, kinetic law, reaction, rule and species objects, you must assign the model object as `Parent` in these objects. Parameter objects can have a model object or kinetic law object as `Parent`. You can display all the component objects with *`modelObj`*`.Models` or `get (`*`modelObj`*`, 'Models')`.

The components of a submodel are contained within the submodel. In addition, a submodel object can reference parameter variables that have been assigned to the model object. For example, a parameter defined within a submodel cannot be used by the parent model or another model object. A submodel object however, can use the parameters assigned to the model object.

You can add a submodel to a model object with the method `addmodel` and removed from its parent with the method `delete`.

Characteristics

Applies to	Objects: model, root
Data type	Array of model objects
Data values	Model object, Default is empty (`[]`).
Access	Read-only

Example

1 Create a model object

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

2 Add submodels to model object and verify

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

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

MATLAB returns

```
SimBiology Model Object Array

   Index:    Name:
   1         nucleus
   2         mitochondrion
```

See Also `sbiomodel`, `addmodel`

Purpose Property with name of object

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

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

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

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

      For example, enclose `[DNA polymerase+]` within brackets in reaction rates and rules; enter `DNA polymerase+` when specifying the name of the species or while writing the reaction.
  - The literal words `null` and `time`. Note that you could specify species names with these words contained within the name. For example `nullaminoacids`, or `nullnucleotides` .
  - The characters `i`, `j`, `->` `<>`,`[`, and`]`.
- Parameters cannot have an empty in `Name` or have the name `time`.

Name

Characteristics

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

Example

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

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

2. Add a reaction object to the model object

```
reactionObj = addreaction(modelObj, 'Aspartic acid -> beta-Aspartyl-PO4')
```

MATLAB returns

```
Reaction Object Array

   Index:    Reaction:
   1         Aspartic acid -> beta-Aspartyl-PO4
```

3. Set reaction `Name` and verify

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

MATLAB returns

```
ans =

  Aspartate kinase reaction
```

See Also

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

Notes

Purpose

Property with HTML text describing SimBiology object

Description

Contains user-specified comments about a SimBiology object.

Characteristics

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

Example

1 Create a model object.

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

2 Write notes for the model object.

```
set (modelObj, 'notes', '09/01/05 experimental data')
```

3 Verify the assignment

```
get (modelObj, 'notes')
```

MATLAB returns

```
ans =

09/01/05 experimental data
```

See Also

`sbiomodel`, `sbioparameter`, `sbioreaction`, `sbiorule`, `sbiospecies`

Parameters

Purpose

Property with array of parameter objects

Description

Indicates the parameters in a `Model`, or `KineticLaw` object. Read-only array of `Parameter` objects. Display with `modelObj.Parameters` or `get(modelObj, 'Parameters')`.

The scope of a parameter object is hierarchical and is defined by the parameter's parent. If a parameter is defined with a kinetic law object as its parent, then only the kinetic law object can use the parameter. If a parameter object is defined with a model object as its parent, then all components within the model (including all rules, submodels and kinetic laws (reaction rate equations) can use the parameter.

You can add a parameter to a model object, or kinetic law object with the method `addparameter` and delete it with the method `delete`.

You can view parameter object properties with the `get` command and configure properties with the `set` command.

Characteristics

Applies to	Objects: `model`, `kineticlaw`
Data type	array of parameter objects
Data values	Parameter objects; Default value is empty (`[]`).
Access	Read-only

Example

1 Create a model object, then add a reaction object

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

2 Define a kinetic law for the reaction object

```
kineticlawObj = addkineticlaw(REACTIONobj, 'MassAction');
```

3 Add a parameter and assign it to the kinetic law object (`kineticlawObj`);

```
parameterObj1 = addparameter (kineticlawObj, 'K1');
get (kineticlawObj, 'Parameters')

   Parameter Object Array

   Index:    Name:    Value:    ValueUnits:
   1         K1       1
```

4 Add a parameter and assign it to the model object (`modelObj`);

```
parameterObj1 = addparameter (modelObj, 'K2');

get (modelObj, 'Parameters')

   Parameter Object Array

   Index:    Name:    Value:    ValueUnits:
   1         K2       1
```

See Also

`addparameter`, `delete`, `sbioparameter`

MATLAB functions `get` and `set`

ParameterVariableNames

Purpose Property showing cell array of reaction rate parameters

Description `ParameterVariableNames` shows the parameters used by the kinetic law object to determine the `ReactionRate` equation in the reaction object. Use `setparameter` to assign `ParameterVariableNames`. When you assign species to `ParameterVariableNames`, SimBiology maps these parameter names to `ParameterVariables` in the kinetic law object.

If the reaction is using a kinetic law the `ReactionRate` property of a reaction object shows the result of a mapping from an abstract kinetic law. The `ReactionRate` is determined by the kinetic law object `Expression` property by mapping `ParameterVariableNames` to `ParameterVariables` and `SpeciesVariableNames` to `SpeciesVariables`.

Characteristics

Applies to	Object: `kineticlaw`
Data type	Cell array of strings
Data values	Cell array of parameters
Access	Read/Write

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

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

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

2 Create a kinetic law object for the reaction object, of type `'Henri-Michaelis-Menten'`

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

`reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3. The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) that should to be set. To set these variables,

   ```
   setparameter(kineticlawObj,'Vm', 'Va');
   setparameter(kineticlawObj,'Km', 'Ka');
   ```

4. Verify that the parameter variables are correct.

   ```
   get (kineticlawObj, 'ParameterVariableNames')
   ```

   MATLAB returns

   ```
   ans =

       'Va'    'Ka'
   ```

See Also

Reaction object property: `ReactionRate`,

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

Kinetic law object property: `SpeciesVariableNames`

Method: `setparameter`.

ParameterVariables

Purpose

Property showing parameters in abstract kinetic law

Description

Description

Indicates the parameter variables that are used in the `Expression` property of the abstract kinetic law object. Used to determine the `ReactionRate` equation in the reaction object. Use the MATLAB function `set` to assign `ParameterVariables` to an abstract kinetic law. For more information see abstract kinetic law.

Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Defined by abstract kinetic law
Access	Read/Write in abstract kinetic law. Read-only in kinetic law.

Example

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

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

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

2. Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`

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

   `reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3. The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables.

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

MATLAB returns

```
ans =

    'Vm' 'Km'
```

See Also

Reaction object property: `ReactionRate`

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

Kinetic law object properties: `SpeciesVariableNames`, `ParameterVariableNames`.

Method: `setparameter`

MATLAB function `set`

Parent

Purpose

Property indicating the parent object

Description

Indicates the parent object for a SimBiology object (read-only). The `Parent` property indicates accessibility of the object. The object is accessible to the `Parent` object and other objects within the `Parent` object. The value of `Parent` depends on the type of object and how it was created.

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

Characteristics

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

See Also

`sbiomodel`, `addkineticlaw`, `addmodel`, `addparameter`, `addreaction`.

Purpose Property to indicate reaction products

Description 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 `[]`.
Access	Read-only

Example

1 Create a model object

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

2 Add reaction objects

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

3 Verify assignment.

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

MATLAB returns

```
Species Object Array

   Index:    Name:    InitialAmount:    InitialAmountUnits:
   1         c        0
```

`2` `d` `0`

See Also `addreaction`, `addspecies`, `addproduct`, `rmproduct`

RandomState

Purpose

Property to set random number generator

Description

`RandomState` sets the random number generator for the stochastic solvers. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object.

SimBiology uses a pseudorandom number generator. The sequence of numbers generated is determined by the state of the generator, which can be specified by the integer `RandomState`. If `RandomState` is set to integer `J`, the random number generator is initialized to its J^{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 resets the state at startup. The default value of `RandomState` is `[]`.

Characteristics

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

Example

Shows how to change `RandomState` settings.

1. Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `expltau`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
set(configsetObj, 'SolverType', 'expltau')
```

2. Change the `Randomstate` to `5`.

```
set(configsetObj.SolverOptions, 'RandomState', 5)
```

```
get(configsetObj.SolverOptions, 'RandomState')


ans =

  5
```

See Also `ErrorTolerance`, `LogDecimation`, `MaxIterations`

Purpose

Property to indicate reaction reactants.

Description

Reactants is a 1-by-n species object array with species in the reaction. If the `Reaction` property is modified to use a different reactant, the `Reactants` property will be updated accordingly.

You can add reactant species to the reaction with the `addreactant` method.

You can remove reactant species from the reaction with the `rmreactant` `method`. You can also update reactants by setting the `Reaction` property with the function `set`.

Characteristics

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

Example

1. Create a model object

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

2. Add reaction objects

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

3. View the reactants for `reactionObj`.

```
get(reactionObj, 'Reactants')
```

MATLAB returns

```
Species Object Array

  Index:    Name:    InitialAmount:    InitialAmountUnits:
  1         a        0
```

```
                  2             b            0
```

See Also `addreaction`, `addspecies`, `addreactant`, `rmreactant`

Purpose

Property to indicate the reaction object reaction

Description

Property to indicate the reaction represented in the reaction object. Indicates the chemical reaction that can change the amount of one or more species, for example: `'A + B > C'`. This property is different from the model object property `Reactions`.

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

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

While the following are valid reactions,

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

reactions that combine species with null are invalid.

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

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

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

Characteristics

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

Example

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

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

2 Verify that the reaction property records the input.

```
get (reactionObj, 'Reaction')
```

MATLAB returns

```
ans =

a + b -> c + d
```

See Also

`sbioreaction`, `addreaction`

Purpose

Property containing the reaction rate equation in reaction object

Description

Defines the reaction rate equation. You can define a `ReactionRate` with or without the `KineticLaw` property. `KineticLaw` defines the type of reaction rate. The `addkineticlaw` function configures the `ReactionRate` based on the `KineticLaw` and the species and parameters specified in the kinetic law object properties `SpeciesVariableNames` and `ParameterVariableNames`.

The reaction takes place in the reverse direction if the `Reversible` property is true. This is reflected in `ReactionRate`. The `ReactionRate` includes the forward and reverse rate if reversible

You can specify `ReactionRate` without `KineticLaw`. Use the `set` function to specify the reaction rate equation. SimBiology adds species variables while creating `reactionObj` using the `addreaction` method. You must add the parameter variables (to the `modelObj` in this case). See the example below.

Once you have specified the `ReactionRate` without `KineticLaw`, if you later configure the `reactionObj` to use `KineticLaw` the `ReactionRate` is unset until you specify `SpeciesVariableNames` and `ParameterVariableNames`.

Characteristics

Applies to	Object: reaction
Data type	`char string`
Data values	Reaction rate string. Default is `''`
Access	Read/Write

Examples

Example 1

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

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

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

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

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

`reactionObj` KineticLaw property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`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 a names `Vm_d, 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, then add a reaction object.

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

2. Specify `ReactionRate` and verify the assignment.

   ```
   set (reactionObj, 'ReactionRate', 'k*a');
   get(reactionObj, 'ReactionRate')
   ```

   MATLAB returns

   ```
   ans =

   k*a
   ```

3. You cannot simulate the model until you add the parameter `k` to the `modelObj`.

   ```
   parameterObj = addparameter(modelObj, 'k');
   ```

   SimBiology adds the parameter to the `modelObj` with default `Value` = `1.0` for the parameter.

See Also

`sbioreaction`, `addreaction`, `sbioparmeter`, `addparameter`, `Reversible`

Purpose

Property with an array of reaction objects.

Description

Property to indicate the reactions in a `Model` object. Read-only array of reaction objects.

A reaction object defines a chemical reaction that occurs between species. The species for the reaction are defined in the `Model` object property `Species`.

You can add a reaction to a model object with the method `addreaction` and you can remove a reaction from the model object with the method `delete`.

Characteristics

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

Example

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

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

2. Verify that the reactions property records the input

```
get (modelObj, 'Reactions')
```

MATLAB returns

```
Reaction Object Array

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

See Also

`sbioreaction`, `addreaction`, `delete`

Purpose Property to specify allowable error relative to component

Description `RelativeTolerance` specifies the allowable error tolerance relative to the state vector at each simulation step. The state vector contains values for all the state variables, for example species amounts for all the species.

`RelativeTolerancet` is a property of `SolverOptions` object. `SolverOptions` is a property of the `configset` object. `RelativeTolerance` is available for the ode solvers (`'ode45'`, `'ode23'`, `'ode113'`, `'ode15s'`, `'ode23s'`, `'ode23t'`, and `'ode23tb'`).

If you set the `RelativeTolerance` at `1e-2` you are specifying that an error of 1% relative to each state value is acceptable at each simulation step.

At each simulation step, the solver estimates the local error e_i in the i^{th} state vector `y`. Simulation converges at that time step if e_i satisfies the following equation:

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

Thus at higher state values, convergence is determined by `RelativeTolerance`. As the state values approach zero, convergence is controlled by `AbsoluteTolerance` The choice of values for `RelativeTolerance` and `AbsoluteTolerance` will vary depending on the problem. The default values should work for first trials of the simulation; however if you want to optimize the solution, consider that there is a trade-off between speed and accuracy. If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy. If the results appear to be inaccurate, you can decrease the tolerance values but this will slow down the solver. If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`
Data values	>0, <1; default is `1e-3`.
Access	Read/Write

Example

Shows how to change `AbsoluteTolerance`.

1 Retrieve the `configset` object from the `modelObj`.

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

2 Change the `AbsoluteTolerance` to `1e-8`.

```
set(configsetObj.SolverOptions, 'RelativeTolerance', 1.0e-6)
get(configsetObj.SolverOptions, 'RelativeTolerance')


ans =

  1.0000e-006
```

See Also

`AbsoluteTolerance`

Purpose

Property to indicate whether a reaction is reversible or irreversible

Description

Defines whether a reaction is reversible or irreversible. The rate of the reaction is defined by the ReactionRate property. For a reversible reaction the reaction rate equation is the sum of the rate of the forward and reverse reactions. The type of reaction rate is defined by the KineticLaw property. If a reaction is changed from reversible to irreversible or vice versa after `KineticLaw` is assigned, the new `ReactionRate` is determined only if `Type` is `MassAction`.. All other `Types` result in unchanged `ReactionRate`. For `MassAction` the first parameter specified is assumed to be the rate of the forward reaction.

Characteristics

Applies to	Object: reaction
Data type	`boolean`
Data values	`true`, `false`. Default value is `false`
Access	Read/Write

Example

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

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

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

2. Set the `Reversible` property for the `reactionObj` to `true` and verify this setting.

```
set (reactionObj, 'Reversible', true)
get (reactionObj, 'Reversible')
```

MATLAB returns

```
ans =
```

```
1
```

MATLAB returns `1` for `true` and `0` for `false`.

In the next steps the example illustrates how the reaction rate equation is assigned for reversible reactions.

3. Create a kinetic law object for the reaction object, of the type `'MassAction'`.

   ```
   kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
   ```

   `reactionObj KineticLaw` property is configured to `kineticlawObj`.

4. The `'MassAction'` kinetic law for reversible reactions has two parameter variables (`'Forward Rate Parameter'` and `'Reverse Rate Parameter'`) that you should set. The species variables for `MassAction` are automatically determined. To set the parameter variables, first create the parameter variables as parameter objects (`parameterObj1, parameterObj2`) with names `Kf, Kr` and assign the object to `kineticlawObj`.

   ```
   parameterObj1 = addparameter(kineticlawObj, 'Kf');
   parameterObj2 = addparameter(kineticlawObj, 'Kr');
   ```

5. Set the variable names for the kinetic law object.

   ```
   set(kineticlawObj,'ParameterVariableNames', {'Kf' 'Kr'});
   ```

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

   ```
   get (reactionObj, 'ReactionRate')
   ```

   MATLAB returns

   ```
   ans =

   Kf*a*b - Kr*c*d
   ```

See Also

`sbioreaction`, `addreaction`, `addparameter`, `ParameterVariableNames`, `ReactionRate`

Purpose

Property to define certain species and parameter interactions

Description

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. `Rule` is a MATLAB expression that defines the change in the species object quantity or a parameter object `Value` when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules see `addrule`, and `RuleType`.

Characteristics

Applies to	Object: rule
Data type	`char string`
Data values	`char string` defined as species or parameter objects. Default is empty.
Access	Read/write

Example

1 Create a model object, then add a reaction object

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

2 Add a rule

```
ruleObj = addrule(modelObj, '10-a+b')
```

MATLAB returns

```
Rule Object Array

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

See Also

`addrule`, `delete`, `sbiorule`

Purpose

Property for defining the type of rule for the rule object.

Description

`RuleType` indicates the type of rule defined by the rule object. A `Rule` object defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species A and species B must be some value. `Rule` is a MATLAB expression that defines the change in the species object quantity or a parameter object `Value` when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules see `addrule`, and `sbiorule`

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

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

  ```
  species1 + species2 - species_total
  ```

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

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

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

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

Write the rule in SimBiology as

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

Characteristics

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

Example

1 Create a model object, then add a reaction object

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

2 Add a rule that specifies the quantity of a species `c`. In the rule expression `k` is the rate constant for `a -> b`

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

3 Change the `RuleType` from the default (`'algebraic'`) to `'rate'`. and verify using the `get` command

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object

```
    Active: 1
Annotation: ''
      Name: ''
     Notes: ''
    Parent: [1x1 SimBiology.Model]
      Rule: 'c = k*(a+b)'
  RuleType: 'rate'
```

```
         Tag: ''
        Type: 'rule'
    UserData: []
```

See Also

`sbiorule`, `addrule`, `delete`

Rules

Purpose

Property showing rules in model object

Description

Indicates the rules in a `Model` object. Read-only array of `SimBiology.Rule` objects.

A **rule** is a mathematical expression that modifies a species amount or a parameter value. A rule defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species `A` and species `B` must be some value.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules see `addrule`, and `RuleType`.

Characteristics

Applies to	Object: model
Data type	Array of rule objects
Data values	Rule object
Access	Read-only

Example

1 Create a model object, then add a reaction object

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

2 Add a rule

```
ruleobj = addrule(modelObj, '10-a+b')
```

MATLAB returns

```
Rule Object Array

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

See Also `addrule`, `delete`, `sbiorule`

Purpose Property holding options for logged species

Description `RuntimeOptions` holds options for species that will be logged during the simulation run. The runtime options object can be accessed through this property.

The `LogDecimation` property of the configuration set object defines how often data is logged.

Property Summary

StatesToLog	Property to specify species data recorded
Type	Property to indicate SimBiology object type

Characteristics

Applies to	Object: `configset`
Data type	Object
Data values	Run time options
Access	Read-only

Example

1. Create a model object and retrieve its configuration set.

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

2. Retrieve the `RuntimeOptions` object from the `configset` object.

```
runtimeObj = get(configsetObj, 'RunTimeOptions')


Runtime Settings:

    StatesToLog:         all
```

See Also MATLAB functions `get`, `set`

SolverOptions

Purpose

Property holding the model solver options

Description

`SolverOptions` is an object that holds the model solver options in the `configset` object. Changing the property `SolverType` changes the options specified in the `SolverOptions` object.

Properties of `SolverOptions` are summarized in the property summary on this page.

Property Summary

AbsoluteTolerance	Property to specify largest allowable absolute error
ErrorTolerance	Property specifies explicit or implicit tau error tolerance
LogDecimation	Property to specify recorded simulation output frequency
MaxIterations	Property to specify nonlinear solver maximum iterations in implicit tau
RandomState	Property to set random number generator
RelativeTolerance	Property to specify allowable error relative to component
Type	Property to indicate SimBiology object type

Characteristics

Applies to	Object: `configset`
Data type	Object

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

Example

Illustrates the changes in `SolverOptions` for various `SolverType` settings.

1 Retrieve the `configset` object from the `modelObj`.

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

2 Configure the `SolverType` to `ode45`.

```
set(configsetObj, 'SolverType', 'ode45')
get(configsetObj, 'SolverOptions')


Solver Settings: (ode)

     AbsoluteTolerance:    1.000000e-006
     RelativeTolerance:    1.000000e-003
```

3 Configure the `SolverType` to `ssa`.

```
set(cs, '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`

Purpose

Property to select solver type for simulation

Description

`SolverType` selects a solver for a simulation. The valid `SolverType` values are`'ssa'`, `'expltau'`, `'impltau'`, `'ode45'`, `'ode23'`, `'ode113'`, `'ode15s'`, `'ode23s'`, and `'ode23t'`. The default solver is `ode15s`. For a discussion about these solver types, see “Selecting a Solver”.

Changing the solver type changes the options (properties) specified in the `SolverOptions` property of the `configset` object. If you change any `SolverOptions` these changes are persistent when you switch `SolverType`. For example if you set the `ErrorTolerance` for the `expltau` solver and then change to `impltau` when you switch back to `expltau` the `ErrorTolerance` will have the number you assigned.

Characteristics

Applies to	Object: `configset`
Data type	`enum`
Data values	`'ssa'`, `'expltau'`, `'impltau'`, `'ode45'`, `'ode23'`, `'ode113'`, `'ode15s'`, `'ode23s'`, `'ode23t'`, `'ode23tb'`. Default is `ode15s`.
Access	Read/Write

Example

1. Retrieve the `configset` object from the `modelObj`.

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


Configuration Settings - default (active)
     SolverType:          ode15s
     StopTime:            10.000000

   SolverOptions:
     AbsoluteTolerance:    1.000000e-006
     RelativeTolerance:    1.000000e-003
```

```
   RuntimeOptions:
     StatesToLog:          all

   CompileOptions:
     UnitConversion:       true
     DimensionalAnalysis:  true
```

2 Configure the `SolverType` to `ode45`.

```
set(configsetObj, 'SolverType', 'ode45')
configsetObj

   Configuration Settings - default (active)
     SolverType:           ode45
     StopTime:             10.000000

   SolverOptions:
     AbsoluteTolerance:    1.000000e-006
     RelativeTolerance:    1.000000e-003

   RuntimeOptions:
     StatesToLog:          all

   CompileOptions:
     UnitConversion:       true
     DimensionalAnalysis:  true
```

See Also

`getconfigset`

MATLAB function `set`

Species

Purpose Property showing species in model object

Description Indicates the species in a `Model` object. Read-only array of SimBiology species objects.

Species are entities that take part in reactions. A species object is added to the `Species` property when a reaction is added to the model object with the method `addreaction`. A species object can also be added to the `Species` property with the method `addspecies`.

If you remove a reaction with the method `delete`, and a species is no longer being used by any of the remaining reactions, the species object is *not* removed from the `Species` property. You have to use the `delete` method to remove species.

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

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

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

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

Characteristics

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

See Also

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

SpeciesVariablesNames

Purpose

Property showing cell array of species used in reaction rate equation

Description

`SpeciesVariablesNames` shows the species used by the kinetic law object to determine the `ReactionRate` equation in the reaction object. Use `setspecies` to assign `SpeciesVariableNames`. When you assign species to `SpeciesVariableNames`, SimBiology maps these species names to `SpeciesVariables` in the kinetic law object.

The `ReactionRate` property of a reaction object shows the result of a mapping from an abstract kinetic law. The `ReactionRate` is determined by the kinetic law object `Expression` property by mapping `ParameterVariableNames` to `ParameterVariables` and `SpeciesVariableNames` to `SpeciesVariables`.

Characteristics

Applies to	Object: kinetic law
Data type	Cell array of strings
Data values	Cell array of species names
Access	Read/Write

Example

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

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

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

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`

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

The `reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3. The `'Henri-Michaelis-Menten'` kinetic law has one species variable (`S`) that you should set. To set this variable,

   ```
   setspecies(kineticlawObj,'S', 'a');
   ```

4. Verify that the species variable is correct.

   ```
   get (kineticlawObj, 'SpeciesVariableNames')
   ```

   MATLAB returns

   ```
   ans =

   'a'
   ```

See Also

Reaction object property: `ReactionRate`

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

Kinetic law object property: `ParameterVariableNames`

Method:`setparameter`

Purpose

Property showing species in abstract kinetic law

Description

Property showing species variables that are used in the `Expression` property of the kinetic law object to determine the `ReactionRate` equation in the reaction object. Use the MATLAB function `set` to assign `SpeciesVariables` to an abstract kinetic law. For more information see abstract kinetic law.

Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Defined by abstract kinetic law
Access	Read/Write in abstract kinetic law. Read-only in kinetic law.

Example

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

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

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

2. Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`

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

`reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3. View the species variable for `'Henri-Michaelis-Menten'` kinetic law.

```
get (kineticlawObj, 'SpeciesVariables')
```

MATLAB returns

```
ans =

    'S'
```

See Also

Reaction object property: `ReactionRate`

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

Kinetic law object property: `ParameterVariableNames`.

Method: `setparameter`

MATLAB function `set`

Purpose

Property to specify species data recorded

Description

`StatesToLog` indicates the species data to log during a simulation. This is the data returned in `x` during execution of `(t,x) = sbiosimulate(modelObj)`. By default all species are logged.

Characteristics

Applies to	Object: `RunTimeOptions`
Data type	Object or vector of objects
Data values	Species objects to log. Default is `All`.
Access	Read/Write

Example

Illustrates how to assign species to `StatesToLog`.

1 Create a model object by importing the file `oscillator.xml`.

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

2 Retrieve the first and second species in the `modelObj`.

```
speciesObj1 = modelObj.Species(1);
speciesObj2 = modelObj.Species(2);
```

3 Retrieve the `configsetObj` of `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

4 Set the `StatesToLog` to record three species; two using the retrieved species objects and one using indexing and view the species in `StatesToLog`.

```
set (configsetObj.RuntimeOptions, 'StatesToLog', ...
    [speciesObj1, speciesObj2, modelObj.Species(3)]);
get(configsetObj.RuntimeOptions, 'StatesToLog')
```

```
Species Object Array

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

Purpose

Property that describes species coefficients in a reaction

Description

Specifies the species coefficients in a reaction. Enter an array of `doubles` indicating the stoichiometry of reactants (negative value) and products (positive value). Example: [-1 -1 2].

The `double` specified cannot be `0`. The reactants of the reaction are defined with a negative number. The products of the reaction are defined with a positive number. For example, the reaction 3 H + A-> 2 C + F has the `Stoichiometry` value of [-3 -1 2 1].

When this property is configured the Reaction property updates accordingly. In the above example, if the `Stoichiometry` value was set to [-2 -1 2 3], the Reaction is updated to 2H + A -> 2C + 3F.

The length of the `Stoichiometry` array is the sum of the `Reactants` array and the `Products` array. To remove a product or reactant from a reaction use the `rmproduct` or `rmreactant` functions. Add a product or reactant and set stoichiometry with methods `addproduct` and `addreactant`

ODE solvers support `double` stoichiometry values such as `0.5`. Stochastic solvers and dimensional analysis currently only support integers in `Stoichiometry`, therefore you must balance the reaction equation and specify integer values for these two cases.

`A -> null` has a stoichiometry value of `[-1]`. `null -> B` has a stoichiometry value of `[1]`.

Characteristics

Applies to	Object: reaction
Data type	Double array
Data values	1–by-n double, where n is length (products) + length (reactants). Default [] (empty)
Access	Read/Write

Example

1 Create a reaction object

```
reactionObj = sbioreaction('2 a + 3 b -> d + 2 c');
```

2 Verify the `Reaction` and `Stoichiometry` properties for `reactionObj`.

```
get(reactionObj,'Stoichiometry')
```

MATLAB returns

```
ans =

-2    -3     1     2
```

3 Set stoichiometry to `[-1 -2 2 2]`.

```
set (reactionObj, 'Stoichiometry', [-1 -2 2 2]);
get (reactionObj, 'Stoichiometry')
```

MATLAB returns

```
ans =

 -1    -2     2     2
```

4 Note with `get` that the `Reaction` property updates automatically.

```
get (reactionObj, 'Reaction')
```

MATLAB returns

```
ans =

a + 2 b -> 2 d + 2 c
```

See Also

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

Purpose

Property to set the stop time for a simulation

Description

`StopTime` sets the stop time for a simulation. The type of `StopTime` is specified in the property `StopTimeType`.

Characteristics

Applies to	Object: `configset`
Data type	`double`
Data values	Enter a positive number. Default is `10`.
Access	Read/Write

Example

1. Retrieve the `configset` object from the `modelObj`.

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

2. Configure the `StopTime` to `20`.

```
set(configsetObj, 'StopTime', 20)
get(configsetObj, 'StopTime')


ans =

    20
```

See Also

`StopTimeType`, `TimeUnits`

StopTimeType

Purpose

Property to specify the type of stop time for a simulation

Description

`StopTimeType` sets the type of stop time for a simulation. The stop time is specified in the `StopTime` property of the `configset` object. Valid types are `approxWallTime`, `numberOfLogs`, and `simulationTime`. The default is `simulationTime`.

- `simulationTime`– specify the stop time for the simulation. The solver determines and sets the time steps and the simulation stops when it reaches the specified `StopTime`.
- `approxWallTime`– specify the approximate stop time according to the clock. For example,`10s` of `approxWallTime` is approximately `10s` of real time.
- `numberOfLogs`– specify the total number of simulation steps to be recorded during the simulation. For example if you want to log three simulation steps, the `numberOfLogs` is `3`. The simulation will stop after the specified `numberOfLogs`.

You can change the `StopTimeType` setting with the `set` function.

Characteristics

Applies to	Object: `configset`
Data type	`enum`
Data values	`approxWallTime`, `numberOfLogs`, and `simulationTime`
Access	Read/Write

Example

1 Retrieve the `configset` object from the `modelObj`.

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

2 Configure the `StopTimeType` to `approxWallTime`.

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


ans =

approxWallTime
```

See Also

`StopTime`, `StatesToLog`, `TimeUnits`

MATLAB function `set`

Purpose

Property to specify a label for a SimBiology object

Description

Specifies a label associated with a SimBiology object. Use this property to group objects and then use `sbioselect` to retrieve. For example, use the `Tag` property in reaction objects to group synthesis or degradation reactions. You can then retrieve all synthesis reactions using `sbioselect`. Similarly, for species objects you can enter and store classification information. For example, membrane protein, transcription factor, enzyme classifications, or whether a species is an independent variable. You can also enter the full form of the name of the species. This is useful when viewing the model in the Block Diagram Explorer. For example, the species object `Name` could be `G6P` for convenience, but in the `Tag` you should enter the full name, `Glucose-6phosphate`. The graphical representation of the model in the Block Diagram Explorer (available in `sbiodesktop`) can be sorted by the `Tag` field, and this feature provides a method to view the full name.

Characteristics

Applies to	Objects: abstract kinetic law, kinetic law, model, parameter, reaction, rule, species
Data type	`char string`
Data values	Any `char string`
Access	Read/Write

Example

1. Create a model object.

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

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

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

3. Verify `Tag` assignment.

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

MATLAB returns

```
ans =

    'Synthesis Reaction'
```

See Also

`sbiomodel`, `sbioabstractkineticlaw`, `sbioparameter`, `sbioreaction`, `sbioroot`, `sbiorule`, `sbiospecies`

TimeUnits

Purpose Property to show the stop time units for a simulation

Description `TimeUnits` shows units for the stop time for a simulation. The type of `StopTime` is specified in the property `StopTimeType`. Unit is `seconds`.

Characteristics

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

See Also `StopTimeType`, `StopTime`

Purpose

Property to indicate SimBiology object type

Description

Indicates a SimBiology object type. When you create an object in SimBiology, the value of Type is automatically defined.

For example, when a `Species` object is created, the value of Type is automatically defined as `'species'`.

Characteristics

Applies to	Objects: abstract kinetic law, configuration set, `CompileOptions`, kinetic law, model, parameter, reaction, root, rule, species, `RuntimeOptions`, `SolverOptions`.
Data type	`char string`
Data values	`abstract_kinetic_law`, `configset`, `compileoptions`, `kineticlaw`, `parameter`, `reaction`, `root`, `rule`, `runtimeoptions`, `sbiomodel`, `species`, `solveroptions`.
Access	Read-only

See Also

`sbiomodel`, `sbioabstractkineticlaw`, `sbioparameter`, `sbioreaction`, `sbioroot`, `sbiorule`, `sbiospecies`

UnitConversion

Purpose

Property to indicate whether to perform unit conversion.

Description

`UnitConversion` specifies whether to perform unit conversion for the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object.

When `UnitConversion` is set to `true`, SimBiology converts the matching physical quantities to one consistent unit system in order to resolve them. This conversion is in preparation for correct simulation, but species amounts are returned in the user-specified units.

For example, consider a reaction `a + b > c`. Using mass action kinetics the reaction rate is defined as `a*b*k` where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are 0.01M and 0.005M respectively, then units of `k` are `1/(M*second)`. If you specify `k` with another equivalent unit definition, for example, `1/((molecules/liter)*second)`, `UnitConversion` occurs after `DimensionalAnalysis`.

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

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

Characteristics

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

Example

Shows how to retrieve and set `unitconversion` from the default `true` to `false` in the default configuration set in a model object

1 Import a model.

```
modelObj = sbmlimport('oscillator')


SimBiology Model - Oscillator

   Model Components:
     Models:            0
     Parameters:        0
     Reactions:         42
     Rules:             0
     Species:           23
```

2 Retrieve the `configset` object of the model object.

```
configsetObj = getconfigset(modelObj)

 Configuration Settings - default (active)
     SolverType:            ode15s
     StopTime:              10.000000

   SolverOptions:
     AbsoluteTolerance:     1.000000e-006
     RelativeTolerance:     1.000000e-003

   RuntimeOptions:
     StatesToLog:           all

   CompileOptions:
     UnitConversion:        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 `UnitConversion`.

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

See Also

`getconfigset`, `sbiosimulate`.

MATLAB functions `get` and `set`.

Purpose

Property to specify data to associate with object

Description

Property to specify data that you want to associate with a SimBiology object. The object does not use this data directly, but you can access it using the function `get` or dot notation.

Characteristics

Applies to	Objects: abstract kinetic law, kinetic law, model, parameter, reaction, rule, species
Data type	Any
Data values	Any. Default is empty
Access	Read/Write

See Also

`sbiomodel`, `sbioabstractkineticlaw`, `sbioparameter`, `sbioreaction`, `sbioroot`, `sbiorule`, `sbiospecies`

UserDefinedKineticLaws

Purpose

Property containing user-defined kinetic laws

Description

`UserDefinedKineticLaws` is a SimBiology root object property showing all user-defined abstract kinetic laws. Use the command `sbiowhos -userdefined -kineticlaw` to see the list of user-defined kinetic laws. You can use user-defined kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example:

```
kineticlawObj = addkineticlaw(reactionObj, 'my_kinetic_law');
```

You can add, modify, or delete `UserDefinedKineticLaws`. Create an abstract kinetic law with the command `sbioabstractkineticlaw` and add it to the user-defined kinetic law library with the command `sbioaddtolibrary`. `sbioaddtolibrary` also updates the `UserDefinedKineticLaws` property of the root object.

See "Abstract Kinetic Law" on page 6-27 for a definition and more information.

Characteristics

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

Examples

Example 1

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

```
sbiowhos -userdefined -kineticlaw


Abstract Kinetic Law Object Array
```

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

Example 2

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

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


Abstract Kinetic Law Object Array

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

Example 3

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

1. Create an abstract kinetic law.

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

2. Assign the parameter and species variables to the expression.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined

   Abstract Kinetic Law Object Array

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

Alternatively,

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

   Abstract Kinetic Law Object Array

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

See Also

`BuiltInKineticLaws`, `BuiltInUnits`, `BuiltInUnitPrefixes`

MATLAB functions `get` and `set`

Purpose

Property containing user-defined unit prefixes

Description

`UserDefinedUnitPrefixes` is a SimBiology root object property showing all user-defined unit prefixes. You can specify units with prefixes for species amounts and parameter values, because, SimBiology enables you to do dimensional analysis and unit conversion during simulation. The valid units and unit prefixes are either built-in or user-defined. Use the command `sbiowhos -userdefined -unit` to see the list of user-defined units.

You can add, modify, or delete `UserDefinedUnitPrefixes`. You can define a unit prefix with the command `sbioregisterunitprefix`, which enables you to create the unit and add it to the user-defined unit prefixes library, and also add it to the `UserDefinedUnitPrefixes` property of the root object.

Characteristics

Applies to	Object: root
Data type	`char string`
Data values	Valid unit prefixes
Access	Read/Write

Example

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

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

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

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

```
sbiowhos -userdefined -unitprefix

SimBiology UserDefined Unit Prefixes
```

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

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

```
r = sbioroot
r.UserDefinedUnitPrefixes


ans =

    'peta'
```

See Also `BuiltInUnitPrefixes`, `BuiltInUnits`, `UserDefinedUnits`, `UserDefinedKineticLaws`

Purpose

Property containing user-defined units

Description

`UserDefinedUnits` is a SimBiology root object property showing all user-defined units. You can specify units for species amounts and parameter values, because, SimBiology enables you to do dimensional analysis and unit conversion during simulation. The valid units are either built-in or user-defined. Use the command `sbiowhos -userdefined -unit` to see the list of user-defined units.

You can add, modify, or delete `UserDefinedUnits`. You can define a unit with the command `sbioregisterunit`, which enables you to create the unit and add it to the user-defined units library, and also add it to the `UserDefinedUnits` property of the root object.

Characteristics

Applies to	Object: root
Data type	`char string`
Data values	Valid units
Access	Read/Write

Example

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

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

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

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

```
sbiowhos -userdefined -unit


SimBiology UserDefined Units
```

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

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

```
r = sbioroot
r.UserDefinedUnits


ans =

    'secondorderconstant'
    'firstorderconstant'
```

See Also `BuiltInUnitPrefixes`, `BuiltInUnits`, `UserDefinedUnitPrefixes`, `UserDefinedKineticLaws`

Value

Purpose

Property to assign value to parameter object

Description

The property `Value` is the value of the parameter object. The parameter object defines an assignment that can be used by the model object and/or the kinetic law object. Create parameters and assign `Value` using the method `addparameter`.

Characteristics

Applies to	Object: parameter
Data type	`double`
Data values	Any `double`. Default value is 1.0.
Access	Read/Write

Example

Assign a parameter with value to the model object

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

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

2. Add a parameter to the model object (`modelObj`) with `Value` `0.5`.

```
parameterObj1 = addparameter (modelObj, 'K1', 0.5)
```

MATLAB returns

```
Parameter Object Array

Index:    Name:    Value:    ValueUnits:
1         K2       0.5
```

See Also

`addparameter`, `sbioparameter`

ValueUnits

Purpose

Property with parameter value units

Description

Indicates the unit definition of the parameter object `Value` property. ValueUnits can be one of the builtin units. To get a list of the builtin units use the `sbioshowunits` function. If `ValueUnits` changes from one unit definition to another, the `Value` does not automatically convert to the new units. The `sbioconvertunits` function does this conversion.

You can add a parameter object to a model object or a kinetic law object.

Characteristics

Applies to	Object: parameter
Data type	`char string`
Data values	Unit from units library, default is empty '
Access	Read/Write

Example

Assign a parameter with value to the model object.

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

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

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

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

   MATLAB returns

   ```
   Parameter Object Array

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

See Also

`addparameter`, `sbioparameter`, `sbioshowunits`, `sbioconvertunits`

Index