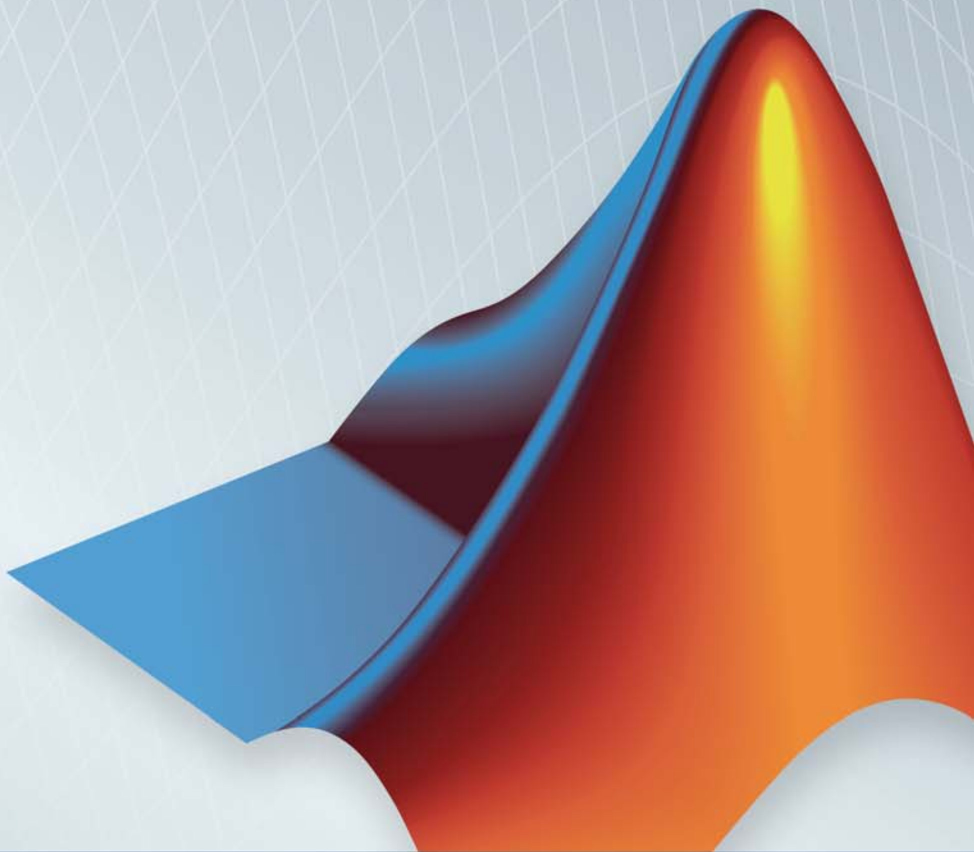
SimBiology®

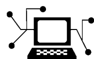
Reference

R2013a



MATLAB®

How to Contact MathWorks



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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

SimBiology® Reference

© COPYRIGHT 2005–2013 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

September 2005	Online only	New for Version 1.0 (Release 14SP3+)
March 2006	Online only	Updated for Version 1.0.1 (Release 2006a)
May 2006	Online only	Updated for Version 2.0 (Release 2006a+)
September 2006	Online only	Updated for Version 2.0.1 (Release 2006b)
March 2007	Online only	Rereleased for Version 2.1.1 (Release 2007a)
September 2007	Online only	Rereleased for Version 2.1.2 (Release 2007b)
October 2007	Online only	Updated for Version 2.2 (Release 2007b+)
March 2008	Online only	Updated for Version 2.3 (Release 2008a)
October 2008	Online only	Updated for Version 2.4 (Release 2008b)
March 2009	Online only	Updated for Version 3.0 (Release 2009a)
September 2009	Online only	Updated for Version 3.1 (Release 2009b)
March 2010	Online only	Updated for Version 3.2 (Release 2010a)
September 2010	Online only	Updated for Version 3.3 (Release 2010b)
April 2011	Online only	Updated for Version 3.4 (Release 2011a)
September 2011	Online only	Updated for Version 4.0 (Release 2011b)
March 2012	Online only	Updated for Version 4.1 (Release 2012a)
September 2012	Online only	Updated for Version 4.2 (Release 2012b)
March 2013	Online only	Updated for Version 4.3 (Release 2013a)

Contents

1 Functions — Alphabetical List

2 Methods — Alphabetical List

3 Properties — Alphabetical List

Index

Functions — Alphabetical List

sbioabstractkineticlaw

Purpose

Create kinetic law definition

Syntax

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

Arguments

Name	Enter a name for the kinetic law definition. `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

`abstkineticlawObj` `= sbioabstractkineticlaw('`*`Name`*`')` creates an abstract kinetic law object, with the name *`Name`* and returns it to *`abstkineticlawObj`*. Use the abstract kinetic law object to specify a *kinetic law definition*.

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

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

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

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

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

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

Method Summary

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

Property Summary

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

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

Examples

1 Create a kinetic law definition.

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

2 Assign the parameter and species variables in the expression.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined

SimBiology Abstract Kinetic Law Array

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

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

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

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

See Also

`addkineticlaw` | `addparameter` | `addreaction` | `sbiomodel`

sbioaccelerate

Purpose

Prepare model object for accelerated simulations

Syntax

```
sbioaccelerate(modelObj)
sbioaccelerate(modelObj,optionObj)

sbioaccelerate(modelObj,csObj,dvObj)

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

Description

`sbioaccelerate(modelObj)` prepares a model object for an accelerated simulation using its active configuration set (configset), and, if available, active variants and active doses. A SimBiology model can contain multiple configsets with only one being active at any given time. The active configset contains the settings to use in model preparation for acceleration.

For accelerated simulations, use `sbioaccelerate` before running `sbiosimulate`. You must use the same model and configset for both functions.

Rerun `sbioaccelerate`, before calling `sbiosimulate`, if you modify this model, other than:

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

`sbioaccelerate(modelObj,optionObj)` uses an option object specified by `optionObj` as one of the following:

- `configset object`
- `variant object`
- `scheduleDose object`
- `repeatDose object`

- array of doses or variants

Currently, a particular dose object can only be accelerated with a single model. You cannot use the same dose object for multiple models to be accelerated. You must create a new copy of the dose for each model.

`sbioaccelerate(modelObj,csObj,dvObj)` uses a configset object `csObj` and dose, variant, or an array of doses or variants specified by `dvObj`. If `csObj` is set to `[]`, then the function uses the active configset object.

`sbioaccelerate(modelObj,csObj,variantObj,doseObj)` uses a configset object `csObj`, variant object or variant array specified by `variantObj` and dose object or dose array specified by `doseObj`.

Requirements:

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

Input Arguments

modelObj - SimBiology model

SimBiology model object

SimBiology model, specified as a SimBiology model object. The model minimally needs one reaction or rate rule to be accelerated for simulations.

optionObj - Option object

configset object | variant object or array of variant objects | dose object or array of dose objects

Option object, specified as one of the following: `configset object`, `variant object`, array of variant objects, `scheduleDose object`, `repeatDose object`, or array of dose objects.

- When you accelerate the model using an array of dose objects, you can simulate the model using any subset of the dose objects from the array.
- You can use any or no variant input arguments when running `sbioaccelerate`.

csObj - Configuration set object

configset object | `[]`

Configuration set object, specified as a `configset object` that stores simulation-specific information. When you specify `csObj` as`[]`, `sbioaccelerate` uses the currently active configset.

dvObj - Dose or variant object

dose object or array of dose objects | variant object or array of variant objects | `[]`

Dose or variant object, specified as one of the following: `scheduleDose object`, `repeatDose object`, array of dose objects, `variant object`, or array of variant objects.

- Use `[]` when you want to explicitly exclude any variant objects from the `sbioaccelerate` function.
- When `dvObj` is a dose object, `sbioaccelerate` uses the specified dose object as well as any active variant objects if available.
- When `dvObj` is a variant object, `sbioaccelerate` uses the specified variant object as well as any active dose objects if available.

variantObj - Variant object

variant object or array of variant objects | `[]`

Variant object, specified as a `variant object` or array of variant objects. Use `[]` when you want to explicitly exclude any variant object from `sbioaccelerate`.

doseObj - Dose object

dose object or array of dose objects | []

Dose object, specified as a `scheduleDose object`, `repeatDose object`, or array of dose objects. A dose object defines additions that are made to species amounts or parameter values. Use `[]` when you want to explicitly exclude any dose objects from `sbioaccelerate`.

Examples

Prepare a Model for Accelerated Simulation

Create a SimBiology model from an SMBL file.

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

Prepare the model for accelerated simulation.

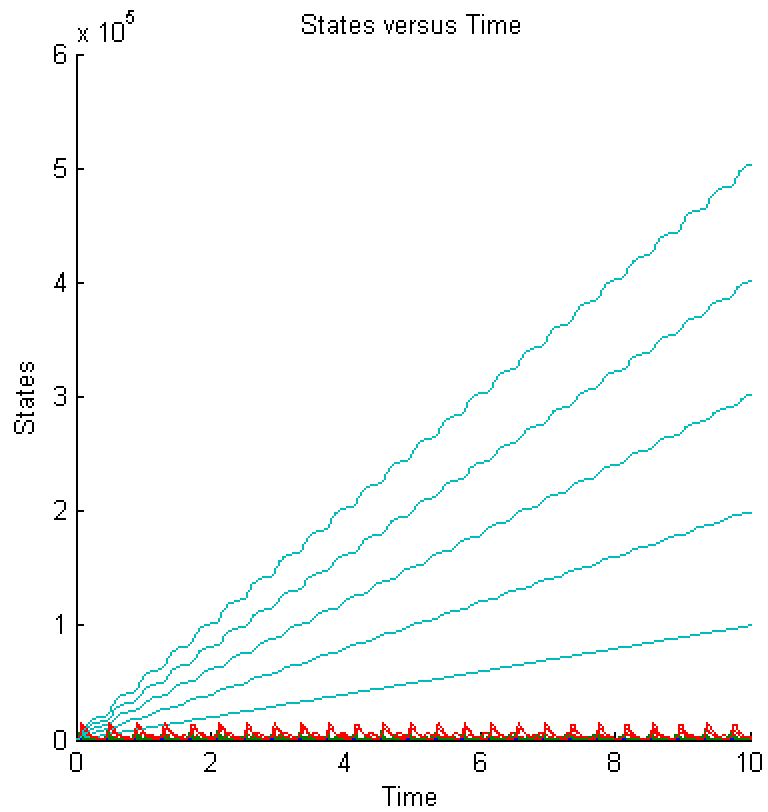
```
sbioaccelerate(m);
```

Simulate the model using different initial amounts of species `x`.

```
x = sbioselect(m,'type','species','name','x');
for i=1:5
 x.initialAmount = i;
 sd(i) = sbiosimulate(m);
end
```

Plot the results.

```
sbioplot(sd)
```



Accelerate Simulation Using a User-Defined Configset Object

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add a new configuration set using a different stop time of 15 seconds.

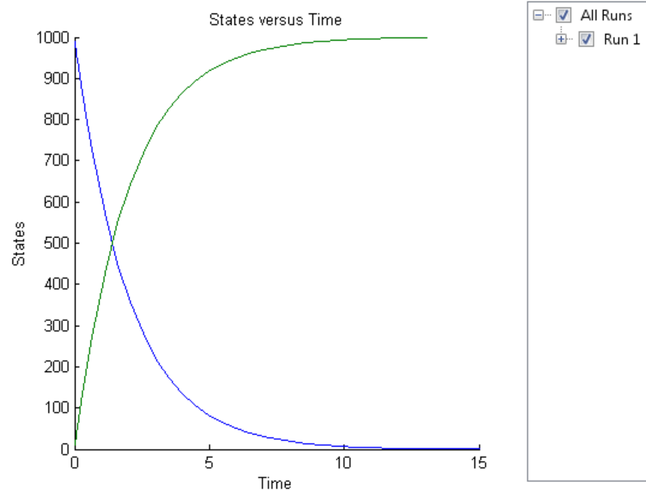
```
csObj = addconfigset(m1,'newStopTimeConfigSet');
csObj.StopTime = 15;
```

Prepare the model for accelerated simulation using the new configset object.

```
sbioaccelerate(m1,csObj);
```

Simulate the model using the same configset object.

```
sim = sbiosimulate(m1,csObj);
sbioplot(sim)
```



Accelerate Simulation Using an Array of Dose Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add two doses of 100 molecules each for species `x`, scheduled at 2 and 4 seconds respectively.

```
dObj1 = adddose(m1,'d1','schedule');
dObj1.Amount = 100;
dObj1.AmountUnits = 'molecule';
```

```
dObj1.TimeUnits = 'second';
dObj1.Time = 2;
dObj1.TargetName = 'unnamed.x';

dObj2 = adddose(m1,'d2','schedule');
dObj2.Amount = 100;
dObj2.AmountUnits = 'molecule';
dObj2.TimeUnits = 'second';
dObj2.Time = 4;
dObj2.TargetName = 'unnamed.x';
```

Prepare the model for accelerated simulation using the array of both doses.

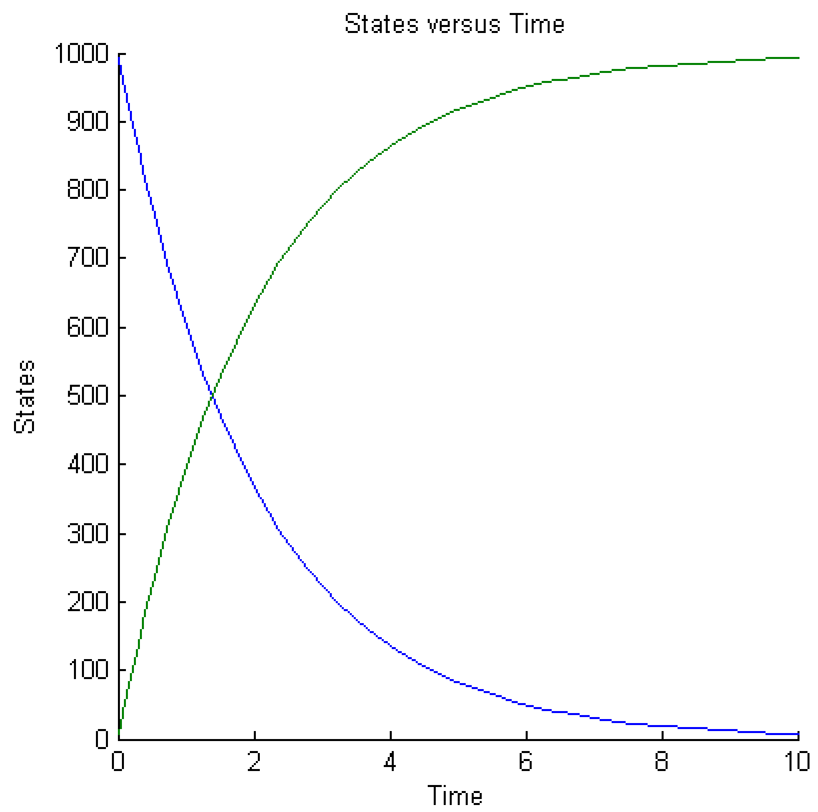
```
sbioaccelerate(m1,[dObj1,dObj2]);
```

Simulate the model using no dose or any subset of the dose array.

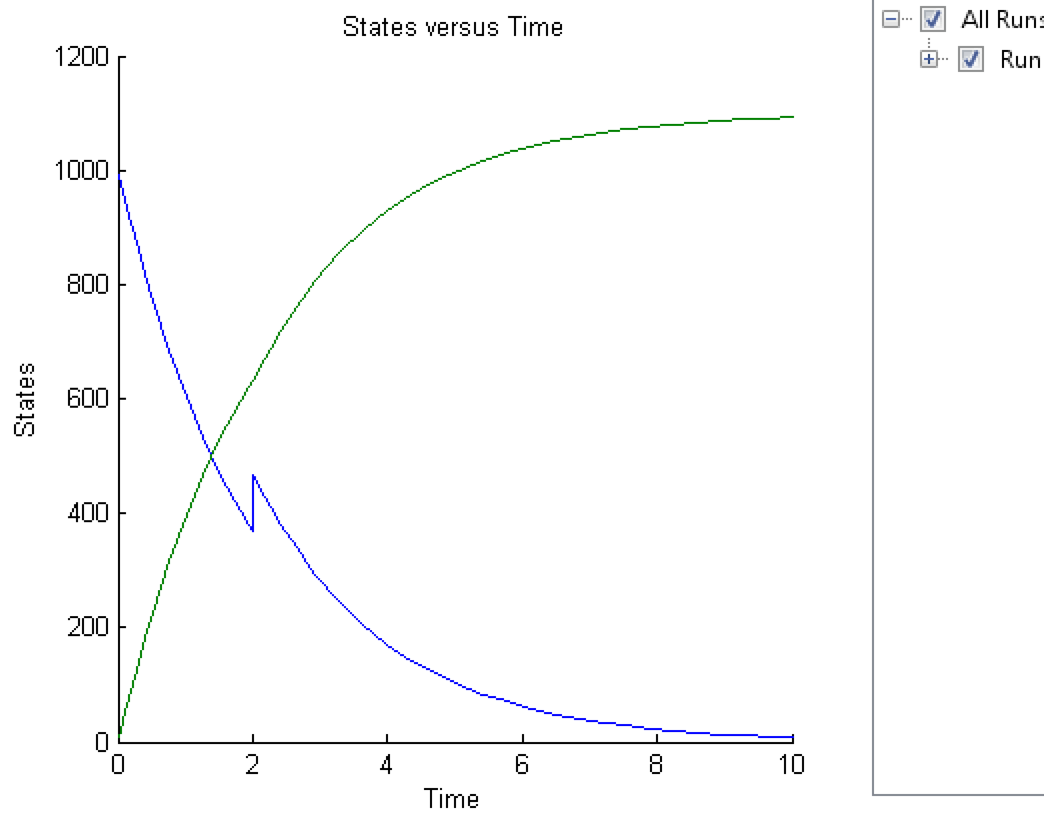
```
sim1 = sbiosimulate(m1);
sim2 = sbiosimulate(m1,dObj1);
sim3 = sbiosimulate(m1,dObj2);
sim4 = sbiosimulate(m1,[dObj1,dObj2]);
```

Plot the results.

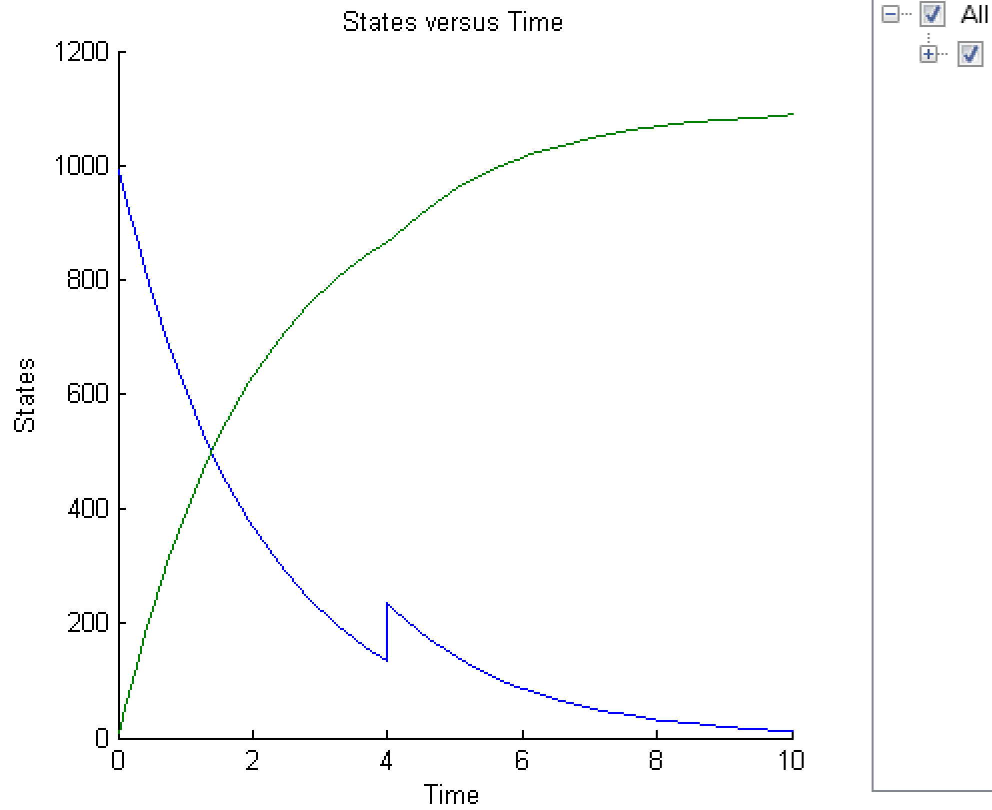
```
sbioplot(sim1)
```



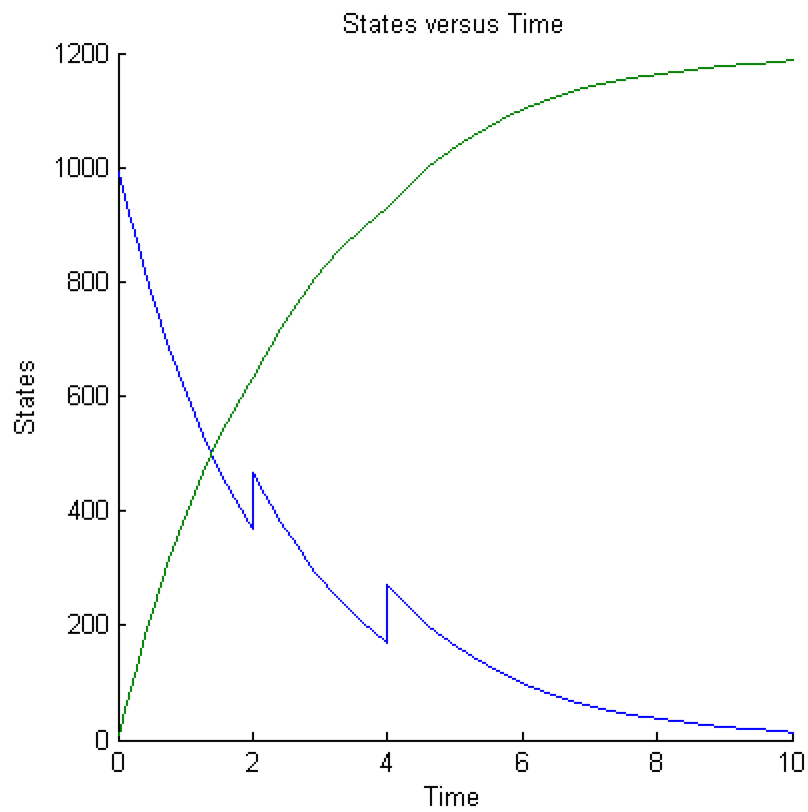

```
sbioplot(sim2)
```




```
sbioplot(sim3)
```




```
sbioplot(sim4)
```



Accelerate Simulation Using Configset and Dose Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Get the default configuration set from the model.

```
defaultConfigSet = getconfigset(m1,'default');
```

Add a scheduled dose of 100 molecules at 2 seconds for species `x`.

```
dObj = adddose(m1,'d1','schedule');
dObj.Amount = 100;
dObj.AmountUnits = 'molecule';
dObj.TimeUnits = 'second';
dObj.Time = 2;
dObj.TargetName = 'unnamed.x';
```

Prepare the model for accelerated simulation using the default configset object and added dose object.

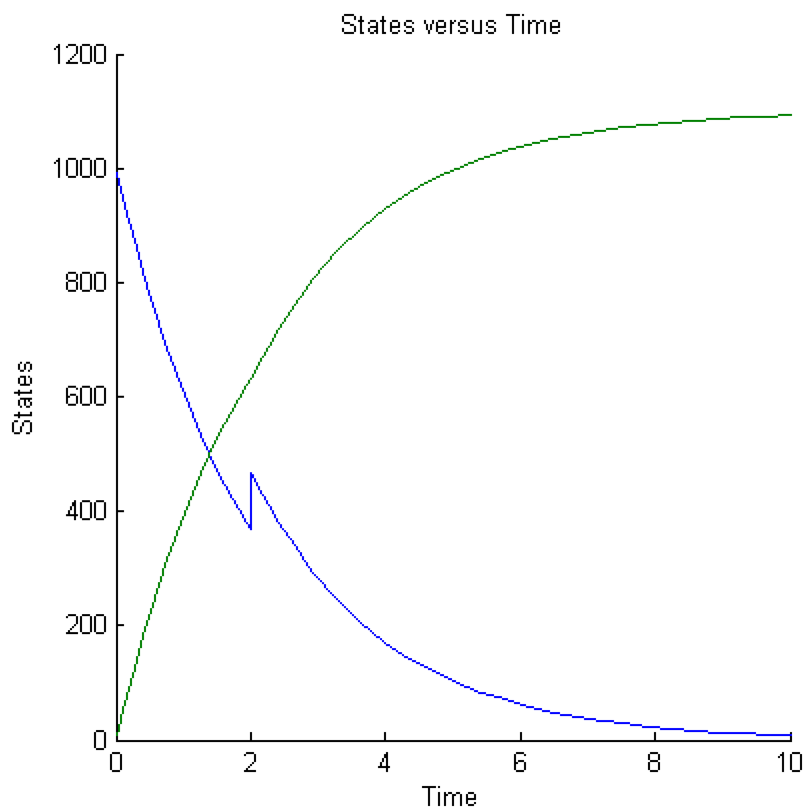
```
sbioaccelerate(m1,defaultConfigSet,dObj);
```

Simulate the model using the same configset and dose objects.

```
sim = sbiosimulate(m1,defaultConfigSet,dObj);
```

Plot the result.

```
sbioplot(sim);
```



Accelerate Simulation Using Configset, Dose, and Variant Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add a new configuration set using a different stop time of 15 seconds.

```
csObj = m1.addconfigset('newStopTimeConfigSet');
csObj.StopTime = 15;
```

Add a scheduled dose of 100 molecules at 2 seconds for species x.

```
dObj = adddose(m1,'d1','schedule');
dObj.Amount = 100;
dObj.AmountUnits = 'molecule';
dObj.TimeUnits = 'second';
dObj.Time = 2;
dObj.TargetName = 'unnamed.x';
```

Add a variant of species x using a different initial amount of 500 molecules.

```
vObj = addvariant(m1,'v1');
addcontent(vObj,{'species','x','InitialAmount',500});
```

Prepare the model for accelerated simulation using the configset, dose, and variant objects. In this case, the third argument of sbioaccelerate must be the variant object.

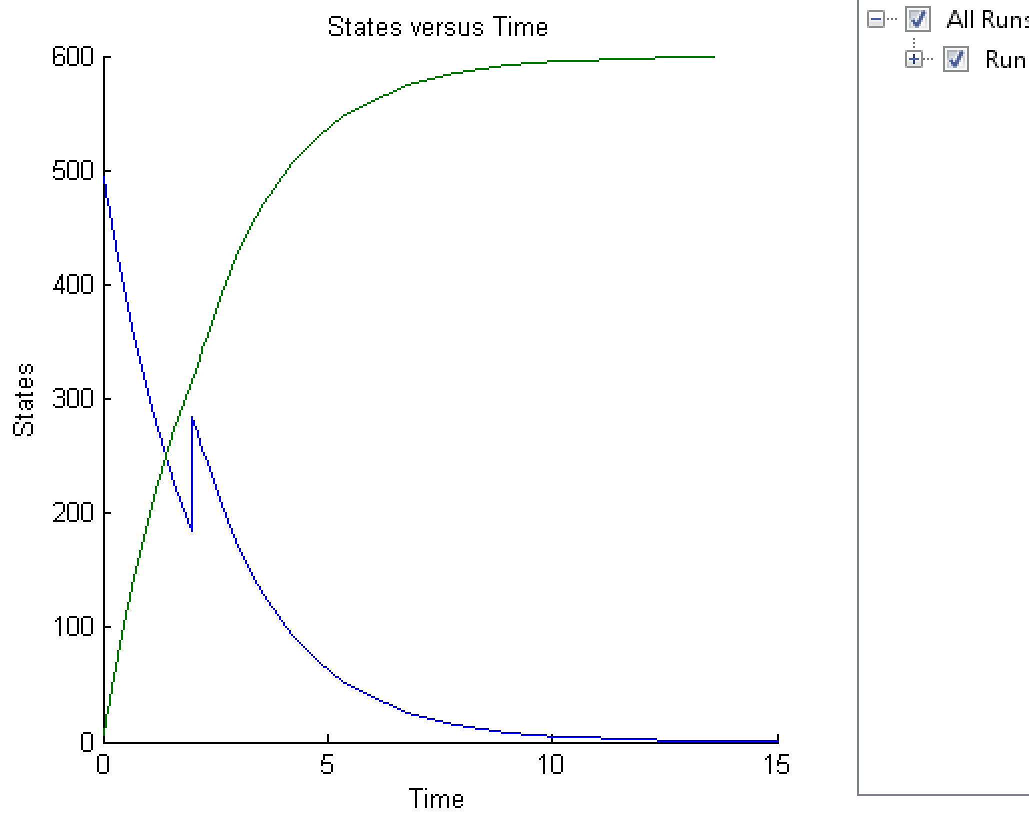
```
sbioaccelerate(m1,csObj,vObj,dObj);
```

Simulate the model using the same configset, variant, and dose objects.

```
sim = sbiosimulate(m1,csObj,vObj,dObj);
```

Plot the result.

```
sbioplot(sim);
```



See Also `sbiosimulate`

Purpose

Add to user-defined library

Syntax

```
sbioaddtolibrary (abstkineticlawObj)
sbioaddtolibrary (unitObj)
sbioaddtolibrary (unitprefixObj)
```

Arguments

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

Description

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

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

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

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

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

library components become available automatically in future MATLAB sessions.

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

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

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

Examples

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

1 Create a kinetic law definition.

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

2 Assign the parameter and species variables in the expression.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined

SimBiology Abstract Kinetic Law Array

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

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

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

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

See Also

`addkineticlaw` | `sbioabstractkineticlaw` | `sbioremovefromlibrary` | `sbioroot` | `sbiounit` | `sbiounitprefix`

sbioconsmoiety

Purpose

Find conserved moieties in SimBiology model

Syntax

```
[G, Sp] = sbioconsmoiety(modelObj)
[G, Sp] = sbioconsmoiety(modelObj, alg)
H = sbioconsmoiety(modelObj, alg,'p')
H = sbioconsmoiety(modelObj, alg,'p', FormatArg)
[SI, SD, L0, NR, ND] = sbioconsmoiety(modelObj,'link')
```

Arguments

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

SD	Cell array containing the names of dependent species in the model.
L0	Link matrix relating `SI` and `SD`. The link matrix *L0* satisfies `ND = L0*NR`. For the `'link'` functionality, species with their `BoundaryCondition` or `ConstantAmount` properties set to true are treated as having stoichiometry of zero in all reactions. *L0* is a sparse matrix. To convert it to a full matrix, use the `full` function.
NR	Reduced stoichiometry matrices containing one row for each independent species. The concatenated matrix `[NR;ND]` is a row-permuted version of the full stoichiometry matrix of *modelObj*. *NR* is a sparse matrix. To convert it to a full matrix, use the `full` function.
ND	Reduced stoichiometry matrices containing one row for each dependent species. The concatenated matrix `[`*NR*`;`*ND*`]` is a row-permuted version of the full stoichiometry matrix of *modelObj*. *ND* is a sparse matrix. To convert it to a full matrix, use the `full` function.

Description

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

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

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

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

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

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

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

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

Examples

Example 1

This example shows conserved moieties in a cycle.

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

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

2. Look for conserved moieties.

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

   g =

        1     1     1


   sp =

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

Example 2

Explore semipositive conservation relations in the `oscillator` model.

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

 ans =

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

See Also

`getstoichmatrix`

How To

- "Conserved Moiety Determination"

Purpose

Convert unit and unit value to new unit

Syntax

`sbioconvertunits(Obj, 'unit')`

Description

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

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

Examples

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

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

   At the command prompt, type:

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


SimBiology Species Array

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

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

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

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

Units are converted from molecule to mole.

```
get (speciesObj, 'InitialAmountUnits')


ans =

mole
```

The `InitialAmount` value is changed.

```
get (speciesObj, 'InitialAmount')


ans =

  3.8192e-023
```

See Also

`sbioshowunits`

How To

- `sbioshowunits`

Purpose

Copy library to disk

Syntax

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

Description

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

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

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

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

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

Examples

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

1 Create a kinetic law definition.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined

SimBiology Abstract Kinetic Law Array

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

3. Copy the user-defined kinetic law library.

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

4. Verify with `sbiowhos`.

```
sbiowhos -kineticlaw myLibFile
```

See Also

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

Purpose

Open SimBiology desktop for modeling and simulation

Syntax

```
sbiodesktop
sbiodesktop(modelObj)
sbiodesktop(File)
```

Input Arguments

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

Description

`sbiodesktop` opens the SimBiology desktop, which lets you:

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

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

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

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

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

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

Examples

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

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

See Also

`sbioroot` | `sbiofittool` | `simbiology`

Purpose

Construct dose object

Syntax

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

Inputs

DoseName	Name of the dose object.
DoseType	Selects which type of dose object to construct. Enter either `'schedule'` or `'repeat'` • `'schedule'`creates a `ScheduleDose` object and defines the dose with a time array, amount array, and rate array. • `'repeat'`creates a `RepeatDose` object and defines the dose with a dose amount, number of dose repetitions, and the time between doses.

Output Arguments

doseObj	`ScheduleDose` or `RepeatDose` object.

Description

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

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

doseObj = sbiodose(...'*PropertyName*', *PropertyValue*...) defines dose object properties. You can enter the property name/property value pairs in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

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

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

Examples

Construct a repeat dose object:

1. In the MATLAB Command Window, enter:

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

2. Define a repeating dose:

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

Construct a schedule dose object:

1. In the MATLAB Command Window, enter:

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

2. Define a dose schedule:

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

See Also

`adddose` | `getdose` | `removedose` | `copyobj` | `get` | `set`

How To

- `Model object`
- `ScheduleDose object`
- `RepeatDose object`

Purpose

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

Syntax

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

Arguments

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

Description

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

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

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

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

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

Examples

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

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

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

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

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

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

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

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

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

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

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

See Also

`sbioensemblerun` | `sbioensemblestats` | `sbiomodel`

Purpose

Multiple stochastic ensemble runs of SimBiology model

Syntax

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

Arguments

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

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

Description

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

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

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

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

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

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

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

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

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

Examples

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

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

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

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

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

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

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

---

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

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

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

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

See Also

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

Purpose

Get statistics from ensemble run data

Syntax

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

Arguments

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

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

Description

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

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

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

Examples

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

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

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

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

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

3 Perform an ensemble of 20 runs with no interpolation.

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

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

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

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

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

See Also

`sbioensemblerun` | `sbioensembleplot` | `sbiomodel`

sbiofitstatusplot

Purpose

Plot status of sbionlmefit or sbionlmefitsa

Syntax

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

Description

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

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

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

Input Arguments

beta

The current fixed effects.

status

Structure containing several fields.

Field	Value
`inner`	Structure describing the current status of the inner iterations within the ALT and LAP procedures, with the fields: • `procedure` – `'PNLS'`, `'LME'`, or `'none'` when the procedure is `'ALT'` – `'PNLS'`, `'PLM'`, or `'none'` when the procedure is `'LAP'`

Field	Value
	• state — 'init', 'iter', 'done', or 'none' • iteration — Integer starting from 0, or NaN
procedure	'ALT' or 'LAP'
iteration	Integer starting from 0
fval	Current log-likelihood
Psi	Current random-effects covariance matrix
theta	Current parameterization of Psi
mse	Current error variance

state

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

Definitions

Alt

Alternating algorithm for the optimization of the LME or RELME approximations

FO

First-order estimate

FOCE

First-order conditional estimate

LAP

Optimization of the Laplacian approximation for FO or FOCE

LME

Linear mixed-effects estimation

NLME

Nonlinear mixed effects

PLM

Profiled likelihood maximization

PNLS

Penalized nonlinear least squares

RELME

Restricted likelihood for the linear mixed-effects model

Examples

Obtain status information for NLME fitting:

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

See Also

nlmefit | sbionlinfit | sbionlmefit | sbionlmefitsa

How To

- “Obtaining the Status of Fitting”

Purpose

Open SimBiology desktop for population fitting

Syntax

```
sbiofittool
```

Description

`sbiofittool` opens the SimBiology desktop in a state designed for:

- Importing and plotting data for fitting
- Selecting from a library of pharmacokinetic models
- Performing population fit tasks using `sbionlmefit` or `sbionlmefitsa`
- Performing individual fit tasks using `sbionlinfit`

`sbiofittool` opens a simplified configuration of the SimBiology desktop. However, all desktop functionality is available.

If you opened the SimBiology desktop using the `simbiology` function, then `sbiofittool` changes the desktop layout to optimize it for population fitting.

See Also

`simbiology`

sbiogetmodel

Purpose

Get model object that generated simulation data

Syntax

`modelObj = sbiogetmodel(simDataObj)`

Arguments

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

Description

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

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

Examples

Retrieve the model object that generated the SimData object.

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

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

2 Get the model that generated the simulation results.

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

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

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

3 Check that the two models are the same.

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

See Also

`sbiosimulate`

sbiolasterror

Purpose

SimBiology last error message

Syntax

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

Arguments

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

Description

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

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

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

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

Examples

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

1 Import a model.

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

SimBiology Model - RadioactiveDecay

 Model Components:
   Models:           0
```

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

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

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

SimBiology Model - RadioactiveDecay

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

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

```
a.verify
```

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

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

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

4 Retrieve the error diagnostic `struct`.

```
p = sbiolasterror

p =

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

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

```
p(1)

ans =

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

6 Reset the `sbiolasterror`.

```
sbiolasterror([])


ans =

     []
```

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

```
sbiolasterror(p)


ans =
```

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

See Also

`sbiolastwarning` | `verify`

How To

- sbioroot

sbiolastwarning

Purpose

SimBiology last warning message

Syntax

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

Arguments

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

Description

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

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

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

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

See Also

`sbiolasterror` | `verify`

How To

- sbioroot

Purpose

Load project from file

Syntax

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

Description

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

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

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

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

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

See Also

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

How To

- sbiosaveproject
- sbiowhos
- sbioaddtolibrary
- sbioremovefromlibrary

Purpose

Construct model object

Syntax

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

Arguments

NameValue	Required property to specify a unique name for a `model` object. Enter a character string.
PropertyName	Property name for a `Model` object from "Property Summary" on page 1-60.
PropertyValue	Property value. Valid value for the specified property.

Description

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

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

Simulate *modelObj* with the function `sbiosimulate`.

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

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

Method Summary

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

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

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

Property Summary

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

Examples

1. Create a SimBiology model object.

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

2. List all `modelObj` properties and the current values.

```
get(modelObj)
```

MATLAB returns:

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

3. Display a summary of `modelObj` contents.

```
modelObj

  SimBiology Model - cell

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

See Also

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

sbionlinfit

Purpose

Perform nonlinear least-squares regression using SimBiology models

Syntax

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

Description

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

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

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

Following is an alternative to the previous syntax:

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

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

Input Arguments

modelObj

SimBiology model object used to fit observed data.

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

---

pkModelMapObject

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

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

---

pkDataObj

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

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

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

InitEstimates

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

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

optionStruct

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

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

If you have Parallel Computing Toolbox™, you can enable parallel computing for faster data fitting by setting the name-value pair argument 'UseParallel' to 'always' in the statset options structure as follows:

```
matlabpool open; % Open a MATLAB worker pool for parallel computing
opt = statset(...,'UseParallel','always'); % Enable parallel computing
results = sbionlinfit(...,opt); % Perform data fitting
```

Name-Value Pair Arguments

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

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

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

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

'ParamTransform'

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

```
beta = f(estimate)
```

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

- `0` – `beta = estimate`
- `1` – `beta = log(estimate)` (default)
- `2` – `beta = probit(estimate)`
- `3` – `beta = logit(estimate)`

For details, see "Specifying Parameter Transformations".

'ErrorModel'

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

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

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

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

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

'Weights'

Either of the following:

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

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

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

'Pooled'

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

Default: `false`

Output Arguments

results

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

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

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

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

If your model incudes:

- A single response, then `J` is a matrix of Jacobian values associated with time points in the group of data.
- Multiple responses, then `J` is a 3-D array of Jacobian values associated with time points in the group of data, for each response.

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

SimDataI

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

See Also

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

How To

- "Performing Data Fitting with PKPD Models"

Purpose

Estimate nonlinear mixed effects using SimBiology models

Syntax

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

Description

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

---

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

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

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

Following is an alternative to the previous syntax:

results = `sbionlmefit(...,` *optionStruct*`)` specifies *optionStruct*, a structure containing fields and values, that are the name-value pair arguments accepted by `nlmefit`. The defaults for *optionStruct* are the same as the defaults for the arguments used by `nlmefit`, with the exceptions explained in "Input Arguments" on page 1-70.

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

Input Arguments

modelObject

SimBiology model object used to fit observed data.

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

---

pkModelMapObject

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

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

---

pkDataObject

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

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

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

InitEstimates

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

CovModelObj

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

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

---

optionStruct

Structure containing fields and values that are the name-value pairs accepted by the `nlmefit` function. The defaults for *`optionStruct`* are the same as the defaults for the arguments used by `nlmefit`, with the exceptions noted in "Name-Value Pair Arguments" on page 1-72.

If you have Parallel Computing Toolbox, you can enable parallel computing for faster data fitting by setting the name-value pair argument `'UseParallel'` to `'always'` in the `statset` options structure as follows:

```
matlabpool open; % Open a MATLAB worker pool for parallel computing
opt = statset(...,'UseParallel','always'); % Enable parallel computing
results = sbionlmefit(...,'Options',opt); % Perform data fitting
```

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

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

Name-Value Pair Arguments

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

The `sbionlmefit` function uses the name-value pair arguments supported by the `nlmefit` function.

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

- `FEParamsSelect`
- `FEConstDesign`
- `FEGroupDesign`

- FEObsDesign
- REConstDesign
- REGroupDesign
- REObsDesign
- Vectorization

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

- FEGroupDesign
- ParamTransform
- REParamsSelect

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

Be aware that the defaults for these nlmefit name-value pairs differ when used by sbionlmefit:

'FEGroupDesign'

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

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

'ParamTransform'

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

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

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

'OptimFun'

String specifying the optimization function used in maximizing the likelihood.

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

'Options'

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

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

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

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

Output Arguments

results

Structure containing these fields:

- `FixedEffects` — A `dataset` array containing estimated fixed effects, including standard errors.
- `RandomEffects` — A `dataset` array containing sampled random effects for each group in the observed data in *`pkDataObject`*.
- `IndividualParametereEstimates` — A `dataset` array containing estimated parameter values for individuals, including random effects.
- `PopulationParameterEstimates` — A `dataset` array containing estimated parameter values for the population, without random effects.
- `RandomEffectCovarianceMatrix` — A `dataset` array containing the estimated covariance matrix of the random effects.
- `EstimatedParameterNames` — Cell array of strings specifying names of the estimated parameters.
- `CovariateNames` — Cell array of strings specifying names of the covariates in *`CovModelObj`*.
- `FixedEffectsStruct` — Structure containing the values of the estimated fixed effects.
- `stats` — Structure containing information such as `AIC`, `BIC`, and weighted residuals. For details on the fields in this structure, see the `stats` structure in `nlmefit` in the Statistics Toolbox documentation. However, the fields in the `stats` structure returned by `sbionlmefit` vary slightly from those returned by `nlmefit`, namely:

- `ires`, `pres`, `iwres`, `pwres`, and `cwres` each contain a matrix of raw or weighted residuals, with the number of columns equal to the number of responses in the model.
- The `stats` structure returned by `sbionlmefit` includes an additional field, `Observed`. This field contains a string or cell array of strings specifying the measured responses that correspond to the columns in the matrices of the `ires`, `pres`, `iwres`, `pwres`, and `cwres` fields. The `Observed` field is the same as the `Observed` property of the `PKModelMap` input argument.

SimDataI

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

SimDataP

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

See Also

`Model object` | `nlmefit` | `PKData object` | `SimData object` | `PKModelDesign object` | `PKModelMap object` | `sbiofitstatusplot` | `sbionlinfit` | `sbionlmefitsa`

How To

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

Purpose

Estimate nonlinear mixed effects with stochastic EM algorithm

Syntax

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

Description

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

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

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

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

Following is an alternative to the previous syntax:

`results` = `sbionlmefitsa(..., `*`optionStruct`*`)` specifies *`optionStruct`*, a structure containing fields and values, that are the name-value pair arguments accepted by `nlmefitsa`. The defaults for *`optionStruct`* are the same as the defaults for the name-value pair arguments used by `nlmefitsa`, with the exceptions explained in "Input Arguments" on page 1-78.

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

Input Arguments

modelObject

SimBiology model object used to fit observed data.

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

---

pkModelMapObject

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

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

---

pkDataObject

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

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

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

InitEstimates

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

CovModelObj

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

optionStruct

Structure containing fields and values that are name-value pair arguments accepted by the `nlmefitsa` function. The defaults for *`optionStruct`* are the same as the defaults for the arguments used by `nlmefitsa`, with the exceptions noted in "Name-Value Pair Arguments" on page 1-80.

If you have Parallel Computing Toolbox, you can enable parallel computing for faster data fitting by setting the name-value pair argument `'UseParallel'` to `'always'` in the `statset` options structure as follows:

```
matlabpool open; % Open a MATLAB worker pool for parallel computing
opt = statset(...,'UseParallel','always'); % Enable parallel computing
results = sbionlmefitsa(...,'Options',opt); % Perform data fitting
```

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

Name-Value Pair Arguments

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

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

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

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

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

- FEGroupDesign
- ParamTransform
- REParamsSelect

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

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

'FEGroupDesign'

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

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

'ParamTransform'

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

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

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

'OptimFun'

String specifying the optimization function used in maximizing the likelihood.

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

'Options'

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

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

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

Output Arguments

results

Structure containing these fields:

- `FixedEffects` — A `dataset` array containing estimated fixed effects, including standard errors.
- `RandomEffects` — A `dataset` array containing sampled random effects for each group in the observed data in *`pkDataObject`*.
- `IndividualParametereEstimates` — A `dataset` array containing estimated parameter values for individuals, including random effects.
- `PopulationParameterEstimates` — A `dataset` array containing estimated parameter values for the population, without random effects.
- `RandomEffectCovarianceMatrix` — A `dataset` array containing the estimated covariance matrix of the random effects.

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

SimDataI

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

SimDataP

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

See Also

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

How To

- "Performing Data Fitting with PKPD Models"
- "Specifying an Error Model"
- "Specifying a Nonlinear, Mixed-Effects Model"
- "Specifying Parameter Transformations"

Purpose

NONMEM file definition object for sbionmimport

Syntax

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

Description

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

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

Tips

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

Input Arguments

Filename

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

Property Name/Value Pairs

'CompartmentLabel'

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

Default: `''`

'ContinuousCovariateLabels'

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

Default: `{}`

'DateLabel'

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

Default: `''`

'DependentVariableLabel'

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

Default: `''`

'DoseLabel'

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

Default: `''`

'DoseIntervalLabel'

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

Default: `''`

'DoseRepeatLabel'

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

Default: `''`

'EventIDLabel'

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

Default: `''`

'GroupLabel'

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

Default: `''`

'MissingDependentVariableLabel'

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

> **Default:** `''`

'RateLabel'

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

> **Default:** `''`

'TimeLabel'

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

> **Default:** `''`

'Type'

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

Output Arguments

nmdefObj

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

Examples

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

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

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

See Also

sbionmimport

How To

- “Importing Data”

sbionmimport

Purpose

Import NONMEM-formatted data

Syntax

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

Description

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

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

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

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

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

Input Arguments

Filename

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

nmds

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

nmdefObj

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

Output Arguments

simbioDataset

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

PkDataObj

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

See Also

`sbionmfiledef`

How To

- "Importing Data"

sbioparamestim

Purpose

Perform parameter estimation

Syntax

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

Arguments

`k`	Vector of estimated parameter values. For all optimization methods except `'fminsearch'`, the parameters are constrained to be greater than or equal to 0.
`result`	Structure with fields that provide information about the progress of optimization.
`modelObj`	SimBiology model object.
`tspan`	*n*-by-1 vector representing the time span of the target data `xtarget`.
`xtarget`	*n*-by-*m* matrix, where *n* is the number of time samples and *m* is the number of states to match during the simulation. The number of rows in `xtarget` must equal the number of rows in `tspan`.

`observed_array` Either of the following:

- Array of objects (species, compartment, or nonconstant parameter) in `modelObj`, whose values should be matched during the estimation process
- Cell array of object names (species, compartment, or nonconstant parameter) in `modelObj`, whose values should be matched during the estimation process

---

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

The length of `observed_array` must equal the number of columns in `xtarget`. `sbioparamestim` assumes that the order of elements in `observed_array` is the same as the order of columns in `xtarget`.

estimated_array Either of the following:

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

---

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

---

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

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

- String specifying one of the following functions:
  - `'fminsearch'`
  - `'lsqcurvefit'`
  - `'lsqnonlin'`
  - `'fmincon'`
  - `'patternsearch'`
  - `'patternsearch_hybrid'`
  - `'ga'`
  - `'ga_hybrid'`
  - `'pso'`
  - `'pso_hybrid'`

  For descriptions of how `sbioparamestim` uses the previous functions, see the Function Descriptions on page 1-96 table.

- Two-element cell array, with the first element being one of the previous functions, and the second element being an options structure or object. Use an appropriate options structure or object for each method listed next.

Method	Options Structure or Object
`'fminsearch'`	`optimset`
`'fmincon'` `'lsqcurvefit'` `'lsqnonlin'` `'pso'` `'pso_hybrid'`	`optimoptions`
`'patternsearch'` `'patternsearch_hybrid'`	`psoptimset`
`'ga'` `'ga_hybrid'`	`gaoptimset`

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

If you have Parallel Computing Toolbox, you can enable parallel computing for faster data fitting

Function Descriptions

Function	Description
`fminsearch`	`sbioparamestim` uses the default options structure associated with `fminsearch`, except for: `Display = 'off'` `TolFun = 1e-6*` (Initial value of objective function) **Note** `'fminsearch'` is an unconstrained optimization method, which can result in negative values for parameters.

`lsqcurvefit`	Requires Optimization Toolbox.
	`sbioparamestim` uses the default options structure associated with `lsqcurvefit`, except for:
	`Display = 'off'`
	`FinDiffRelStep =` value of the `SolverOptions.RelativeTolerance` property of the configuration set associated with `modelObj`, with a minimum of `eps^(1/3)`
	`TolFun = 1e-6*` (Initial value of objective function)
	`TypicalX = 1e-6*` (Initial values of components to be estimated)

Function Descriptions (Continued)

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

Function Descriptions (Continued)

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

Function Descriptions (Continued)

Function	Description

`ga`	Requires Global Optimization Toolbox.
	
	`sbioparamestim` uses the default options structure associated with `ga`, except for:
	
	`Display = 'off'`
	`TolFun = 1e-6*` (Initial value of objective function)
	`PopulationSize = 10`
	`Generations = 30`
	`MutationFcn = @mutationadaptfeasible`

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

Function Descriptions (Continued)

Function	Description
`pso`	Requires Global Optimization Toolbox. `sbioparamestim` uses the following default options for the PSO (particle swarm optimization) algorithm [2] [3]:

```
                CreationFcn: @psocreationuniform
                    Display: 'off'
            DisplayInterval: 1
                FunValCheck: 'off'
                  HybridFcn: []
               InertiaRange: [0.1000 1.1000]
           InitialPositions: []
                MaxFunEvals: 3000*numberofvariables
                    MaxIter: 30
    MinNeighborhoodFraction: 0.2500
               NumParticles: 10
             ObjectiveLimit: -Inf
                 OutputFcns: []
           PositionInitSpan: 2000
  SelfRecognitionCoefficient: 1.4900
SocialRecognitionCoefficient: 1.4900
             StallIterLimit: 20
             StallTimeLimit: Inf
                  TimeLimit: Inf
                     TolFun: 1e-6*(Initial objective
                UseParallel: 'never'
                 Vectorized: 'off'
```

Description of options

Option	Description	Values
`CreationFcn`	Handle to the function	`@psocreationuniform`The function

Function Descriptions (Continued)

Function	Description		
		that creates the initial population	creates initial positions of particles for the pso function and has the following format: positions=psocreationuni where: • pos is the numParticles*nvars matrix of particle positions. • nvars is the number of variables. • problemStruct is the problem struct with the following fields: solver for the solver name, 'pso', objective for the objective

Function Descriptions (Continued)

Function	Description		
			function, `nvars` for the number of design variables, `lb` for the vector of lower bounds, and `ub` for the vector of upper bounds. • `options` is an options object containing options created by calling `optimoptions('pso',...)`.
	`Display`	Level of display	`'off'|'none'|'final'|'iter'`
	`DisplayInterval`	Interval for iterative display	Positive integer
	`FunValCheck`	Indicator specifying whether to	`'off'|'on'`

Function Descriptions (Continued)

Function	Description		
		check objective function values are valid	
	HybridFcn	Function that is automatically run at the end of iterations of the solver	@fminsearch|@patternsearc 1-by-2 cell array of the form {@solver, hybridoptions}, where solver is: fminsearch, patternsearch, fminunc, or fmincon and hybridoptions is a structure of options for these functions and their values
	InertiaRange	Lower and upper bound of the adaptive inertia	Two-element vector of same-signed values in increasing order
	InitialPositions	Initial particle positions used to initiate the pso function. If any	Matrix

Function Descriptions (Continued)

Function	Description		
		initial particle positions are outside of the bounds, they are truncated to the bounds.	
	MaxFunEvals	Maximum number of objective function evaluations allowed	Positive integer
	MaxIter	Maximum number of iterations allowed	Positive integer
	MinNeighborhoodFraction	Minimum adaptive neighborhood size	Scalar value from 0 and 1
	NumParticles	Number of particles	Positive integer > = 2
	ObjectiveLimit	Minimum objective function value wanted	Scalar
	OutputFcns	Functions that the pso function	Function handle | cell array of function handlesThe

Function Descriptions (Continued)

Function	Description		
		calls at each iteration	function requires three input arguments and has the following form: `[stopflag,options,optchar` `=` `myFcn(optimValues,options` where: • `optimValues` is a structure containing the following fields: - `Iteration` (iteration number) - `StartTime` (identifier returned by `tic` when the algorithm starts) - `LastImprovement` (iteration of last improvement)

Function Descriptions (Continued)

Function	Description		
			- `LastImprovementTime` (time of last improvement, as returned by `toc(StartTime)`) - `FunEval` (total number of objective function evaluations) - `Positions` (matrix of current particle positions) - `Velocities` (matrix of current particle velocities) - `Fvals` (vector of current objective function values seen by

Function Descriptions (Continued)

Function	Description			
				each particle) - `IndividualBestFvals` (vector of the best objective function values seen by each particle) - `IndividualBestPositi` (matrix of the best positions seen by each particle). • `options` is an options object. • `state` is the state of the algorithm with possible values of `'init'` (initialization state), `'iter'`

Function Descriptions (Continued)

Function	Description		
			(iteration state), and 'done' (finished state). • stopflag is a logical indicating whether the optimization should continue or end at the current iteration. • optchanged is a logical indicating whether the options argument was changed.
	PositionInitSpan	Range of initial particle positions for variables that do not have finite bounds	Positive scalar | vector with n elements, where n is the number of design

Function Descriptions (Continued)

Function	Description		
			variables in the model
	`SelfRecognitionCoefficient`	Coefficient that controls the weighting of each particle's best position when updating the velocity	Finite scalar
	`SocialRecognitionCoefficient`	Coefficient that controls the weighting of the neighborhood's best position when updating the velocity	Finite scalar
	`StallIterLimit`	Number of iterations over which average change in objective function value at current point is less than `options.TolFun`	Positive integer
	`StallTimeLimit`	Maximum number of seconds, as measured by `tic/toc`,	Positive scalar

Function Descriptions (Continued)

Function	Description		
		permitted without an improvement in the best known objective function value	
	`TimeLimit`	Maximum number of seconds, as measured by `tic/toc`, allowed for the optimization	Positive scalar
	`TolFun`	Termination tolerance on function value	Positive scalar
	`UseParallel`	Indicator specifying whether to compute objective functions of a particle swarm in parallel	`'always'` | `'never'`
	`Vectorized`	String specifying whether the computation of the objective function is vectorized.	`'off'`

Function Descriptions (Continued)

Function	Description
	Currently, the algorithm does not support any vectorization.
pso_hybrid	Requires Global Optimization Toolbox. sbioparamestim calls the pso function with the additional option HybridFcn = {@objFcn, options}. The objective function, objFcn, is one of these supported functions: patternsearch, fminsearch, fminunc, or fmincon. options is a structure of options for these functions and their values.

Description

[k, result]= sbioparamestim(modelObj, tspan, xtarget, observed_array, estimated_array) estimates the initial values of compartments, species, and parameters of modelObj, a SimBiology model object, specified in estimated_array, so as to match the values of species and nonconstant parameters given by observed_array with the target state, xtarget, whose time variation is given by the time span tspan. If you have Optimization Toolbox installed, sbioparamestim uses the lsqnonlin function as the default method for the parameter estimation. If you do not have Optimization Toolbox installed, sbioparamestim uses the MATLAB function fminsearch as the default method for the parameter estimation.

[___]= sbioparamestim(___ , observed_array, estimated_array, k0) specifies the initial values of compartments, species, and parameters listed in estimated_array.

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

Algorithms

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

Examples

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

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

   ```
   sbioloadproject gprotein_norules m1;
   ```

2. Store the target data in a variable:

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

3. Store all model parameters in an array:

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

4. Store the species that should match target:

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

5 Estimate the parameters:

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

k =

    0.0100
    0.0000
    0.0004
    4.0000
    0.0040
    1.0000
    0.0000
    0.1100


result =

          fval: 1.4193e+06
      residual: [9x1 double]
      exitflag: 2
    iterations: 2
     funccount: 27
     algorithm: 'trust-region-reflective'
       message: [1x413 char]
```

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

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

```
[k2,r2] = sbioparamestim(m1,tspan,xtarget,Ga,p_array, ...
          {},'patternsearch');
[k3,r3] = sbioparamestim(m1,tspan,xtarget,Ga,p_array, ...
          {},'ga');
[k4,r4] = sbioparamestim(m1,tspan,xtarget,Ga,p_array, ...
   {},'pso');
```

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

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

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

myopt3 = gaoptimset('PopulationSize',25, 'Generations', 10);
[k3,r3] = sbioparamestim(m1,tspan,xtarget, ...
          Ga,p_array,{},{'ga',myopt3});

myopt4 = optimoptions('pso','Display','iter');
[k4,r4] = sbioparamestim(m1,tspan,xtarget,Ga,p_array,{},{'pso',myopt4});
```

References

[1] Yi, T-M., Kitano, H., and Simon, M.I. (2003) A quantitative characterization of the yeast heterotrimeric G protein cycle. PNAS *100*, 10764–10769.

[2] Iadevaia, S., Lu, Y., Morales, F.C., Mills, G.B., and Ram, P.T. (2010) Identification of Optimal Drug Combinations Targeting Cellular Networks: Integrating Phospho-Proteomics and Computational Network Analysis. Cancer Research *70*, 6704–6714.

[3] Abraham, A., Guo, H., and Liu, H. (2006) Swarm Intelligence: Foundations, Perspectives and Applications. Studies in Computational Intelligence, 3–25.

See Also

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

sbioplot

Purpose

Plot simulation results in one figure

Syntax

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

Arguments

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

Description

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

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

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

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

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

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

Examples

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

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

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

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

See Also

`sbiosubplot` | `SimData object`

sbioremovefromlibrary

Purpose

Remove kinetic law, unit, or unit prefix from library

Syntax

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

Description

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

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

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

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

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

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

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

Examples

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

1 Create a kinetic law definition.

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

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

```
sbioaddtolibrary(abstkineticlawObj);
```

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

```
sbiowhos -kineticlaw -userdefined

SimBiology Abstract Kinetic Law Array

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

3 Remove the kinetic law definition.

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

See Also

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

sbioreset

Purpose

Delete all model objects

Syntax

```
sbioreset
```

Description

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

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

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

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

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

Examples

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

1 Import a model into the workspace.

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

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

```
rootObj = sbioroot
```

```
SimBiology Root Contains:

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

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

```
clear all

rootObj


SimBiology Root Contains:

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

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

```
sbioreset
rootObj


SimBiology Root Contains:

 Models:                            0
```

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

See Also

sbioaddtolibrary | sbioroot | sbiounit | sbiounitprefix

How To

- sbioroot

Purpose

Return SimBiology root object

Syntax

rootObj = sbioroot

Arguments

rootObj	Return `sbioroot` to this object.

Description

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

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

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

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

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

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

Method Summary

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

sbioroot

Property Summary

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

See Also

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

How To

- sbiomodel
- addkineticlaw
- sbioreset

Purpose

Save all models in root object

Syntax

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

Description

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

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

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

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

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

Examples

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

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

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

```
sbiosaveproject myprojectfile modelObj timeseriesObj
```

See Also

`sbioaddtolibrary` | `sbioloadproject` | `sbioremovefromlibrary` | `sbiowhos`

How To

- sbioloadproject
- sbiowhos
- sbioaddtolibrary
- sbioremovefromlibrary

Purpose

Search for objects with specified constraints

Syntax

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

Arguments

Out	Object or array of objects returned by the `sbioselect` function. *Out* might contain a mixture of object types (for example, species and parameters), depending on the selection you specify. If *PropertyValue* is a cell array, then the function returns all objects with the property '*PropertyName*' that matches any element of *PropertyValue*.
Obj	SimBiology object or array of objects to search. If an object is not specified, `sbioselect` searches the root.
PropertyName	Any property of the object being searched.
PropertyValue	Specify *PropertyValue* to include in the selection criteria.
TypeValue	Type of object to include in the selection, for example, `sbiomodel`, `species`, `reaction`, or `kineticlaw`.
Condition	The search condition. See the table under "Description" on page 1-128 for a list of conditions.

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

Description

`sbioselect` searches for objects with specified constraints.

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

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

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

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

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

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

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

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

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

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

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

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

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

Conditions for Numeric Properties	Example Syntax

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

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

Conditions for Numeric Properties	Example Syntax

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

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

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

Conditions for Numeric Properties	Example Syntax

equal_and_same_type	Similar to ==, but in addition requires the property value to be of the same type. Search in the model object (modelObj), and return all objects containing a property of type double and a value equal to 0. (Using '==' would also select objects containing a property with a value of false.)
	
	`zeroObj = sbioselect(modelObj, ...`
	`'Where', '*', 'equal_and_same_type', 0);`

unequal_and_same_type	Similar to ~=, but in addition requires the property value to be of the same type. Select all objects containing a property of type double and value not equal to 0. (Using '~=' would also select objects containing a property with a value of true.)
	
	`nonzeroObj = sbioselect(modelObj, ...`
	`'Where', '*', 'unequal_and_same_type', 0);`

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

Conditions for Properties Names or String Values	Example Syntax

`==`	Search in the model object (`modelObj`), and return species objects named `'Glucose'`.
	
	`speciesObj = sbioselect (modelObj,...`
	` 'Type', 'species', 'Where',...`
	` 'Name', '==', 'Glucose')`

`~=`	Search in the model object (`modelObj`), and return species objects that are not named `'Glucose'`.
	
	`speciesObj = sbioselect (modelObj,...`
	` 'Type', 'species', 'Where',...`
	` 'Name', '~=', 'Glucose')`

`==i`	Same as `==`; in addition, this is case insensitive.

`~=i`	Search in the model object (`modelObj`), and return species objects that are not named `'Glucose'`, ignoring case.
	
	`speciesObj = sbioselect (modelObj,...`
	` 'Type', 'species', 'Where',...`
	` 'Name', '~=i', 'glucose')`

Conditions for Properties Names or String Values	Example Syntax

`regexp`. Supports expressions supported by the functions `regexp` and `regexpi`.	Search in the model object (`modelObj`), and return objects that have `'ese'` or `'ase'` anywhere within the name.
	
	`Obj = sbioselect (modelObj, 'Where',...`
	` 'Name', 'regexp', '[ea]se')`
	
	Search in the root, and return objects that have `kinase` anywhere within the name.
	
	`Obj = sbioselect ('Where',...`
	` 'Name', 'regexp', 'kinase')`
	
	Note that this query could result in a mixture of object types (for example, species and parameters).

`regexpi`	Same as `regexp`; in addition, this is case insensitive.

`~regexp`	Search in the model object (`modelObj`), and return objects that do not have `kinase` anywhere within the name.
	
	`Obj = sbioselect (modelObj, 'Where',...`
	` 'Name', '~regexp', 'kinase')`

`~regexpi`	Same as `~regexp`; in addition, this is case insensitive.
`wildcard`	Supports `DOS-style` wildcards ('?' matches any single character, '*' matches any number of characters, and the pattern must match the entire string). See `regexptranslate` for more information.
`wildcardi`	Same as `wildcard`; in addition, this is case insensitive.

Conditions for Properties Names or String Values	Example Syntax

`~wildcard`	Search in the model object (`modelObj`), and return objects that have names that do not begin with `kin*`.
	
	`Obj = sbioselect (modelObj, 'Where',...`
	` 'Name', '~wildcard', 'kin*')`

`~wildcardi`	Same as `~wildcard`; in addition, this is case insensitive.

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

Condition	Example Syntax

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

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

Condition	Example Syntax

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

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

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

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

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

Examples

1. Import a model.

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

2. Find and return an object named `pA`.

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

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

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

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

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

SimBiology Species Array

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

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

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

See Also

`regexp`

Purpose

Show unit prefixes in library

Syntax

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

Arguments

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

Description

`sbioshowunitprefixes` returns information about unit prefixes in the SimBiology library.

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

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

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

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

Examples

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

See Also

sbioconvertunits | sbioshowunits | sbiounitprefix

Purpose

Show units in library

Syntax

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

Arguments

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

sbioshowunits

Description

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

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

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

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

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

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

Examples

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

See Also

`sbioconvertunits` | `sbioshowunitprefixes` | `sbiounit`

Purpose

Simulate model object

Syntax

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

Description

`[t,x,names] = sbiosimulate(`*`modelObj`*`)` simulates *`modelObj`*, a SimBiology model object, using the active configuration set associated with *`modelObj`*, and returns the simulation results in three outputs, described in "Output Arguments" on page 1-145.

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

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

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

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

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

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

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

Input Arguments

modelObj

SimBiology `Model object`.

configsetObj

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

Note If your model contains events, the `Configset` object cannot specify `'expltau'` or `'impltau'` for the `SolverType` property.

Note If your model contains doses, the `Configset` object cannot specify `'ssa'`, `'expltau'`, or `'impltau'` for the `SolverType` property.

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

---

variantObj

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

doseObj

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

Output Arguments

t

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

x

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

names

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

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

simdataObj

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

Examples

The following examples show how to change solver settings.

Example 1

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

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

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

2. Get the active configuration set for the model.

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

3. Set `SolverType` to `ode23t` and set `StopTime` to `10`.

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

4. Simulate the model.

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

5. Plot the results of the simulation.

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

Example 2

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

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

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

2. Simulate the model.

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

3. Plot the results of the simulation.

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

See Also

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

sbiosubplot

Purpose

Plot simulation results in subplots

Syntax

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

Arguments

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

Description

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

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

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

Examples

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

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

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

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

See Also

`sbioplot`

Purpose

Plot data or simulation results in trellis plot

Syntax

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

Arguments

trellisplot	Object returned by `sbiotrellis`. Use `trellisplot` together with the `plot` method to overlay trellis plots. See "Description" on page 1-151 for information about the `plot` method.
Dataset	A `dataset` array containing grouped data to plot.

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

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

Description

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

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

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

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

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

See Also

`sbioplot` | `sbiosubplot`

sbiounit

Purpose

Create user-defined unit

Syntax

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

Arguments

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

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

Description

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

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

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

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

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

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

```
sbioaddtolibrary(unitObject);
```

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

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

Examples

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

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

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

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

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

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

```
rootObj = sbioroot;

rootObj.UserDefinedLibrary.Units

SimBiology UserDefined Units

Index:  Name:            Composition:         Multiplier:     Offset:

1       firstconstant     1/second            1.000000        0.000000

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

Alternatively, use the sbiowhos command.

```
sbiowhos -userdefined -unit


SimBiology UserDefined Units
```

```
Index:  Name:            Composition:         Multiplier:     Offset:

1       firstconstant     1/second            1.000000        0.000000

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

See Also

sbioaddtolibrary | sbioshowunits | sbiounitprefix | sbiowhos

sbiounitcalculator

Purpose

Convert value between units

Syntax

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

Description

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

Examples

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

See Also

`sbioshowunits`

Purpose

Create user-defined unit prefix

Syntax

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

Arguments

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

Description

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

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

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

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

```
sbioaddtolibrary(unitprefixObject);
```

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

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

Examples

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

1. Create a unit prefix.

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

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

   ```
   sbioaddtolibrary(unitprefixObj1);
   ```

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

   ```
   rootObj = sbioroot;

   rootObj.UserDefinedLibrary.UnitPrefixes

   Unit Prefix Array

      Index:  Library:     Name:            Exponent:

      1       UserDefined  peta              15
   ```

   Alternatively, use the `sbiowhos` command.

   ```
   sbiowhos -userdefined -unitprefix
   ```

```
SimBiology UserDefined Unit Prefixes

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

See Also

`sbioaddtolibrary` | `sbioshowunits` | `sbiounit` | `sbiowhos`

How To

- sbioshowunits

sbiovariant

Purpose Construct variant object

Syntax

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

Arguments

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

Description

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

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

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

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

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

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

Method Summary

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

Property Summary

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

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

Examples

1. Create a variant object.

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

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

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

See Also

`addvariant` | `copyobj` | `getvariant`

Purpose

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

Syntax

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

Description

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

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

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

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

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

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

Examples

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

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

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

% Show all contents of project file.
  sbiowhos myprojectfile

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

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

See Also

`whos`

Purpose

Export SimBiology model to SBML file

Syntax

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

Arguments

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

Description

`sbmlexport(modelObj)` exports a SimBiology model object (`modelObj`) to a file with a Systems Biology Markup Language (SBML) Level 2 Version 4 format. The default file extension is `.xml` and the file name matches the model name.

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

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

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

Examples

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

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

References

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

See Also

`sbiomodel` | `sbiosaveproject` | `sbmlimport`

How To

- sbmlimport
- sbiomodel
- sbiosaveproject

Purpose

Import SBML-formatted file

Syntax

```
modelObj = sbmlimport(File)
```

Description

`modelObj = sbmlimport(File)` imports *`File`*, a Systems Biology Markup Language (SBML)-formatted file, into MATLAB and creates a model object `modelObj`.

`File` is a string specifying a file name or a path and file name supported by your operating system. *`File`* extensions are `.sbml` or `.xml`.*`File`* can also be a URL, if you have the Java® programming language.

`sbmlimport` supports SBML Level 2 Version 4 and earlier.

For functional characteristics and limitations, see "Importing from SBML Files".

Input Arguments

File

String specifying either of the following:

- File name or path and file name supported by your operating system
- URL (if you have Java programming language)

Examples

Import SBML model:

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

References

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

Alternatives

Use the SimBiology desktop to import an SBML-formatted file. For more information, see "Importing and Exporting Models, Tasks, and Data from the Desktop" and "Importing from SBML Files".

See Also

`get` | `sbiosimulate` | `sbmlexport` | `set`

How To

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

Purpose

Open SimBiology desktop for modeling and simulation

Syntax

```
simbiology
simbiology(modelObj)
simbiology(File)
```

Input Arguments

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

Description

`simbiology` opens the SimBiology desktop, which lets you:

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

`simbiology(`*`modelObj`*`)` opens the SimBiology desktop with *`modelObj`*, a SimBiology model object. If a project is open in the desktop, the `simbiology` function adds *`modelObj`* to the project.

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

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

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

Examples

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

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

See Also

`sbioroot` | `sbiofittool`

Superclasses

matlab.mixin.Heterogeneous

Purpose

Exported SimBiology model dose object

Description

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

Construction

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

export (model)	Export SimBiology model

Properties

Amount

Amount of dose, a nonnegative scalar value.

AmountUnits

Dose amount units. This property is read only.

DurationParameterName

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

LagParameterName

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

Name

Name of dose object. This property is read only.

Notes

Text describing dose object. This property is read only.

Parent

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

Rate

Rate of dose, a nonnegative scalar value.

RateUnits

Units for dose rate. This property is read only.

TargetName

Species receiving dose. This property is read only.

TimeUnits

Time units for dosing. This property is read only.

Copy Semantics

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

Examples

Exported SimBiology Model Dose Objects

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

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

Get the editable doses from the exported model object.

```
doses = getdose(em)

doses =

  1x4 RepeatDose array with properties:

    Interval
    RepeatCount
    StartTime
    TimeUnits
```

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

The exported model has 4 repeated dose objects.

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

```
doses(3)
```

```
ans =

  RepeatDose with properties:

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

Change the dosing amount for this dose object.

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

doses(3)

ans =

  RepeatDose with properties:

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

See Also

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

Related Examples

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

Superclasses

SimBiology.export.StochasticSimulationOptions

Purpose

Settings for explicit tau-leaping solver of exported SimBiology model

Description

`SimBiology.export.ExplicitTauSimulationOptions` is the class of simulation options associated with the explicit tau-leaping solver of an export model.

Construction

Explicit tau simulation options are created by the `export` method for SimBiology models with a stochastic `SolverType` set to `'expltau'`.

export (model)	Export SimBiology model

Properties

ErrorTolerance

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

LogDecimation

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

MaximumNumberofLogs

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

MaximumWallClock

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

RandomState

Random number generator, a positive integer value.

SolverType

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

StopTime

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

TimeUnits

Time units for simulation. This property is read only.

Copy Semantics

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

See Also

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

Superclasses

SimBiology.export.StochasticSimulationOptions

Purpose

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

Description

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

Construction

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

export (model)	Export SimBiology model

Properties

ErrorTolerance

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

LogDecimation

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

MaxIterations

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

MaximumNumberofLogs

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

MaximumWallClock

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

RandomState

Random number generator, a positive integer value.

SolverType

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

StopTime

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

TimeUnits

Time units for simulation. This property is read only.

Copy Semantics

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

See Also

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

Purpose

Exported SimBiology model object

Description

Exported SimBiology models are limited-access models that can be simulated and accelerated. You can speedup simulation of exported models using Parallel Computing Toolbox, and deploy exported models using MATLAB Compiler™.

By default, all active dose objects, species, parameters, and compartments export with a SimBiology model, and are editable in the exported model object. You can limit which doses, species, parameters, and compartments are editable using additional options during export. Reactions, rules, and events are never editable in an exported model.

Construction

Use the `export` method to export a SimBiology model.

export (model)	Export SimBiology model

Properties

DependentFiles

Function files the model depends on. This property is read only.

ExportNotes

Text with additional information associated with the exported model. This property is read only.

ExportTime

Creation time of the exported model. This property is read only.

InitialValues

Vector of initial values for modifiable species, compartments, and parameters.

Name

Name of the exported model. This property is read only.

Notes

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

SimulationOptions

`SimBiology.export.SimulationOptions` object specifying simulation options.

ValueInfo

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

Methods

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

Copy Semantics

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

Examples

Export SimBiology Model Object

Load a sample SimBiology model object, and export.

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

```
em =
```

```
  Model with properties:

           Name: 'lotka'
     ExportTime: '12-Dec-2012 15:20:13'
    ExportNotes: ''
```

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

```
em.ValueInfo

ans =

  8x1 ValueInfo array with properties:

    Constant
    InitialValue
    Name
    Parent
    QualifiedName
    Tag
    Type
    Units
```

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

```
{em.ValueInfo.Name}

ans =

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

Display the exported model simulation options.

```
em.SimulationOptions

ans =
```

```
ODESimulationOptions with properties:

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

The exported model has a deterministic solver.

See Also

`SimBiology.export.Dose` | `SimBiology.export.SimulationOptions` | `SimBiology.export.ValueInfo` |

Related Examples

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

Concepts

- Class Attributes
- Property Attributes

Superclasses

SimBiology.export.SimulationOptions

Purpose

Settings for deterministic, ordinary differential equation simulation of exported SimBiology model

Description

`SimBiology.export.ODESimulationOptions` is the class of simulation options associated with determinist, ordinary differential equation (ODE) solvers.

Construction

Deterministic simulation options are created by the `export` method for SimBiology models with a deterministic `SolverType` (for example, `sundials` or `ode15s`).

export (model)	Export SimBiology model

Properties

AbsoluteTolerance

Absolute error tolerance applied to state value during simulation, a positive scalar value.

AbsoluteToleranceScaling

Control scaling of absolute error tolerance, a logical value.

AbsoluteToleranceStepSize

Initial guess for time step size for scaling of absolute error tolerance, `[]` or a scalar value.

MaximumNumberofLogs

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

MaximumWallClock

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

MaxStep

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

OutputTimes

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

RelativeTolerance

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

SolverType

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

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

StopTime

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

TimeUnits

Time units for simulation. This property is read only.

Copy Semantics

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

See Also

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

Superclasses

SimBiology.export.Dose

Purpose

Repeated doses for exported SimBiology model

Description

`SimBiology.export.RepeatDose` is the class for export repeat doses.

Construction

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

export (model)	Export SimBiology model

Properties

Amount

Amount of dose, a nonnegative scalar value.

AmountUnits

Dose amount units. This property is read only.

DurationParameterName

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

Interval

Time between doses, a nonnegative scalar value.

LagParameterName

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

Name

Name of dose object. This property is read only.

Notes

Text describing dose object. This property is read only.

Parent

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

Rate

Rate of dose, a nonnegative scalar value.

RateUnits

Units for dose rate. This property is read only.

RepeatCount

Dose repetitions, a nonnegative integer value.

StartTime

Start time for initial dose, a nonnegative scalar value.

TargetName

Species receiving dose. This property is read only.

TimeUnits

Time units for dosing. This property is read only.

Copy Semantics

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

See Also

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

Superclasses

SimBiology.export.Dose

Purpose

Schedule dose for exported SimBiology model

Description

`SimBiology.export.ScheduleDose` is the class for export schedule doses.

Construction

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

export (model)	Export SimBiology model

Properties

Amount

Amount of dose, a nonnegative scalar value.

AmountUnits

Dose amount units. This property is read only.

DurationParameterName

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

LagParameterName

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

Name

Name of dose object. This property is read only.

Notes

Text describing dose object. This property is read only.

Parent

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

Rate

Rate of dose, a nonnegative scalar value.

RateUnits

Units for dose rate. This property is read only.

TargetName

Species receiving dose. This property is read only.

Time

Schedule dose times, a vector of nonnegative values.

TimeUnits

Time units for dosing. This property is read only.

Copy Semantics

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

See Also

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

Purpose

Simulation settings for exported SimBiology model

Description

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

Construction

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

export (model)	Export SimBiology model

Properties

MaximumNumberofLogs

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

MaximumWallClock

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

StopTime

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

TimeUnits

Time units for simulation. This property is read only.

Copy Semantics

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

See Also

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

SimBiology.export.StochasticSimulationOptions

Superclasses

SimBiology.export.SimulationOptions

Purpose

Settings for stochastic simulation of exported SimBiology model

Description

`SimBiology.export.StochasticSimulationOptions` is the superclass of simulation options associated with stochastic solvers. The subclasses of `SimBiology.export.StochasticSimulationOptions` are `SimBiology.export.ExplicitTauSimulationOptions` and `SimBiology.export.ImplicitTauSimulationOptions`.

Construction

Stochastic simulation options are created by the `export` method for SimBiology models with a stochastic `SolverType` (`ssa`, `expltau`, or `impltau`).

export (model)	Export SimBiology model

Properties

LogDecimation

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

MaximumNumberofLogs

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

MaximumWallClock

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

RandomState

Random number generator, a positive integer value.

SolverType

String indicating solver type to use for simulation. This property is read only. The stochastic solver type is one of:

- 'ssa'
- 'expltau'
- 'impltau'

StopTime

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

TimeUnits

Time units for simulation. This property is read only.

Copy Semantics

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

See Also

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

SimBiology.export.ValueInfo

Purpose

Modifiable species, compartments, or parameters in exported SimBiology model

Description

`SimBiology.export.ValueInfo` is the class that describes the modifiable value components in a `SimBiology.export.Model`, including species, parameters, and compartments.

Construction

`ValueInfo` objects are created by the `export` method for SimBiology models. By default, all model species, parameters, and compartments are `ValueInfo` objects, but you can specify which value components to export using the optional `editvals` input argument to `export`.

export (model)	Export SimBiology model

Properties

Constant

Display whether value is constant or time-varying. This property is read only.

InitialValue

Initial value for the component, a scalar value.

Name

Name of species, compartment, or parameter. This property is read only.

Parent

Name of parent model, compartment, or reaction. This property is read only.

QualifiedName

Qualified name of species, compartment, or parameter. This property is read only.

- For compartments and model-scoped parameters, the qualified name is the same as the name.

- For species, the qualified name is `CompartmentName.SpeciesName`.
- For reaction-scoped parameters, the qualified name is `ReactionName.ParameterName`.

Tag

Label for species, compartment, or parameter. This property is read only.

Type

Type of value (species, parameter, or compartment). This property is read only.

Units

Value units. This property is read only

Copy Semantics

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

See Also

`SimBiology.export.Model` | `export`

Methods — Alphabetical List

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

SimBiology.export.Model.accelerate

Purpose

Prepare exported SimBiology model for acceleration

Syntax

`accelerate(model)`

Description

`accelerate(model)` prepares the exported model for acceleration on the current type of computer.

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

Input Arguments

model

`SimBiology.export.Model` object.

Examples

Accelerate Exported SimBiology Model

Load a sample SimBiology model object, and export.

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

```
em =

  Model with properties:

           Name: 'lotka'
     ExportTime: '12-Dec-2012 15:20:13'
    ExportNotes: ''
```

Accelerate the exported model.

```
accelerate(em);
em.isAccelerated
```

```
ans =

     1
```

The logical value `1` indicates that the exported model is accelerated.

See Also

`SimBiology.export.Model` | `SimBiology.export.Model.isAccelerated` |

Related Examples

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

AbstractKineticLaw object

Purpose

Kinetic law information in library

Description

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

Create your own kinetic law definition and add it to the kinetic law library with the `sbioaddtolibrary` function. You can then use the kinetic law to define a reaction rate. To retrieve a kinetic law definition from the user-defined library, first create a root object using `sbioroot`, then use the command `get(rootObj.UserDefinedLibrary, 'KineticLaws')`.

See "Property Summary" on page 2-5 for links to abstract kinetic law object property reference pages.

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

Constructor Summary

`sbioabstractkineticlaw`	Create kinetic law definition

Method Summary

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

Property Summary

`Expression (AbstractKineticLaw, KineticLaw)`	Expression to determine reaction rate equation
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`ParameterVariables`	Parameters in kinetic law definition
`Parent`	Indicate parent object
`SpeciesVariables`	Species in abstract kinetic law
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

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

addcompartment (model, compartment)

Purpose

Create compartment object

Syntax

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

Arguments

modelObj	`Model` object.
owningCompObj	Compartment object that contains the newly created compartment object.
NameValue	Name for a compartment object. Enter a character string unique to the model object. For information on naming compartments, see `Name`.
CapacityValue	Capacity value for the compartment object. Enter `double`. Positive real number, default = `1`.
PropertyName	Enter the name of a valid property. Valid property names are listed in "Property Summary" on page 2-8.
PropertyValue	Enter the value for the property specified in *PropertyName*. Valid property values are listed on each property reference page.

Description

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

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

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

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

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

`compartmentObj = addcompartment(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). “Property Summary” on page 2-8 lists the properties. The `Owner` property is one exception; you cannot set the `Owner` property in the `addcompartment` syntax because, `addcompartment` requires the owning model or compartment to be specified as the first argument and uses this information to set the `Owner` property. After adding a compartment, you can change the owner using the function `set`.

Method Summary

Methods for compartment objects

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

Property Summary

Properties for compartment objects

`Capacity`	Compartment capacity
`CapacityUnits`	Compartment capacity units
`Compartments`	Array of compartments in model or compartment
`ConstantCapacity`	Specify variable or constant compartment capacity
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Owner`	Owning compartment

`Parent`	Indicate parent object
`Species`	Array of species in compartment object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

Examples

1. Create a model object (`modelObj`).

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

2. Add two compartments to the model object.

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

3. Add a compartment to one of the compartment objects.

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

4. Display the `Compartments` property in the model object.

```
get(modelObj, 'Compartments')

SimBiology Compartment Array

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

5. Display the `Compartments` property in the compartment object.

```
get(compartmentObj2, 'Compartments')
```

```
SimBiology Compartment - matrix

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

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

Purpose

Add compartment to `PKModelDesign` object

Syntax

```
PKCompartmentObj = addCompartment(PKModelDesignObj,
   CompObjName)
PKCompartmentObj = addCompartment(PKModelDesignObj,
   CompObjName, Name, Value)
```

Description

`PKCompartmentObj` `= addCompartment(`*`PKModelDesignObj`*`,` *`CompObjName`*`)` constructs a PK compartment with the specified name and adds it to *`PKModelDesignObj`*, a `PKModelDesign` object.

`PKCompartmentObj` `= addCompartment(`*`PKModelDesignObj`*`,` *`CompObjName`*`,` *`Name`*`,` *`Value`*`)` constructs a PK compartment with the specified name, and with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`PKModelDesignObj`	`PKModelDesign object` to which you want to add a compartment
`CompObjName`	Name of the `PKCompartment` object that is constructed

Name-Value Pair Arguments

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

`DosingType`	String specifying the mechanism for drug absorption. Choices are: • `'Bolus'` • `'Infusion'` • `'ZeroOrder'` • `'FirstOrder'` • `''` (default) For more information, see "Dosing Types".
`EliminationType`	String specifying the mechanism for drug elimination. Choices are: • `'Linear'` • `'Linear-Clearance'` • `'Enzymatic'` • `''` (default) For more information, see "Elimination Types".
`HasResponseVariable`	Logical indicating if the drug concentration in this compartment is reported. Multiple compartments in a model can have this property set to `true`. Default is `false`.

Note If you perform a parameter fit on a model, at least one compartment in the model must have a `HasResponseVariable` property set to `true`.

`HasLag`	Logical indicating if any dose targeting this compartment have a lag associated with them. Default is `false`.

These optional name-value pair arguments set the corresponding property of the `PKCompartment` object. You can also set these properties after creating the `PKCompartment` object by using the following syntax:

```
PKCompartmentObj.PropertyName = Value
```

For example:

```
PKCompartmentObj.DosingType = 'Bolus'
```

Output Arguments

`PKCompartmentObj`	`PKCompartment object`

Method Summary

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

Property Summary

`DosingType`	Drug dosing type in compartment
`EliminationType`	Drug elimination type from compartment
`HasLag`	Lag associated with dose targeting compartment

HasResponseVariable	Compartment drug concentration reported
Name	Specify name of object

See Also

"Creating Pharmacokinetic Model Using the Command Line", HasLag, HasResponseVariable, PKCompartment object, PKModelDesign object

Purpose

Create configuration set object and add to model object

Syntax

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

Arguments

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

Description

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

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

configsetObj = `addconfigset(..., '`*PropertyName*`', `*PropertyValue*`, ...)` constructs a configuration set object, *configsetObj*, and configures *configsetObj* with property value pairs. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The *configsetObj* properties are listed in "Property Summary" on page 2-16.

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

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

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

Method Summary

Methods for configuration set objects

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

Property Summary

Properties for configuration set objects

`Active`	Indicate object in use during simulation
`CompileOptions`	Dimensional analysis and unit conversion options
`MaximumNumberOfLogs`	Maximum number of logs criteria to stop simulation
`MaximumWallClock`	Maximum elapsed wall clock time to stop simulation
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`RuntimeOptions`	Options for logged species
`SensitivityAnalysisOptions`	Specify sensitivity analysis options
`SolverOptions`	Specify model solver options

SolverType	Select solver type for simulation
StartTime	Start time for initial dose time
StopTime	Simulation time criteria to stop simulation
TimeUnits	Show time units for dosing and simulation
Type	Display SimBiology object type

Examples

1. Create a model object by importing the oscillator.xml file, and add a Configset object to the model.

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

2. Configure the simulation stop criteria by setting the StopTime, MaximumNumberOfLogs, and MaximumWallClock properties of the Configset object. Set the stop criteria to a simulation time of 3000 seconds, 50 logs, or a wall clock time of 10 seconds, whichever comes first.

```
set(configsetObj, 'StopTime', 3000, 'MaximumNumberOfLogs', 50,...
    'MaximumWallClock', 10)
get(configsetObj)

                         Active: 0
                 CompileOptions: [1x1 SimBiology.CompileOptions]
                           Name: 'myset'
                          Notes: ''
                 RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
   SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysis
                  SolverOptions: [1x1 SimBiology.ODESolverOptions]
                     SolverType: 'ode15s'
                       StopTime: 3000
            MaximumNumberOfLogs: 50
               MaximumWallClock: 10
```

```
              TimeUnits: 'second'
                   Type: 'configset'
```

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

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

See Also

`get`, `getconfigset`, `removeconfigset`, `set`, `setactiveconfigset`

Purpose

Append content to variant object

Syntax

```
addcontent(variantObj, contents)
addcontent(variantObj1, variantObj2)
```

Arguments

variantObj — Specify the variant object to which you want to append data. The `Content` property is modified to add the new data.

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

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

addcontent (variant)

Description

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

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

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

Examples

1. Create a model containing one species.

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

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

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

See Also

addvariant, rmcontent, sbiovariant

Purpose

Add dose object to model

Syntax

```
doseObj2 = adddose(modelObj, 'DoseName')
doseObj2 = adddose(modelObj, 'DoseName', 'DoseType')
doseObj2 = adddose(modelObj, doseObj)
```

Arguments

modelObj	Model object to which you add a dose object.
DoseName	Name of a dose object to construct and add to a model object. *DoseName* is the value of the dose object property `Name`.
DoseType	Type of dose object to construct and add to a model object. Enter either `'schedule'` or `'repeat'`.
doseObj	Dose object to add to a model object. Created with the constructor `sbiodose`.

Outputs

doseObj2	`ScheduleDose` or `RepeatDose` object. A `RepeatDose` or `ScheduleDose` object defines an increase (dose) to a species amount during a simulation.

Description

Before using a dose object in a simulation, use the `adddose` method to add the dose object to a SimBiology model object. Then, set the `Active` dose object property to `true`.

`doseObj2 = adddose(modelObj, 'DoseName')` constructs a SimBiology RepeatDose object (*doseObj2*), assigns *DoseName* to the property `Name`, adds the dose object to a SimBiology model object (*modelObj*), and assigns *modelObj* to the property `Parent`.

doseObj2 = adddose(*modelObj*, '*DoseName*', '*DoseType*') constructs either a SimBiology ScheduleDose object or RepeatDose object (*doseObj*).

doseObj2 = adddose(*modelObj*, *doseObj*) adds a SimBiology dose object (*doseObj*) to a SimBiology model object (*modelObj*), copies the dose object to a second dose object (*doseObj2*), and assigns *modelObj* to the property Parent.

Example

Add a dose object to a model object.

1 Create a model then add a dose to the model.

```
modelObj = sbiomodel('mymodel');
dose1Obj = adddose(modelObj, 'dose1');
```

2 Define properties of the dose object.

```
dose1Obj.Amount = 5'
dose1Obj.Repeat = 6;
dose1obj.Interval = 24;
dose1Obj.TImeUnits = 'hour'
```

See Also

Model object methods:

- adddose — add a dose object to a model object
- getdose — get dose information from a model object
- removedose — remove a dose object from a model object

Dose object constructor sbiodose.

ScheduleDose object and RepeatDose object methods:

- copyobj — copy a dose object from one model object to another model object
- get — view properties for a dose object
- set — define or modify properties for a dose object

Purpose

Add event object to model object

Syntax

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

Arguments

modelObj	Model object.
TriggerValue	Required property to specify a trigger condition. Must be a MATLAB expression that evaluates to a logical value. Use the keyword `'time'` to specify that an event occurs at a specific time during the simulation. For more information, see `Trigger`.
EventFcnsValue	A string or a cell array of strings, each of which specifies an assignment of the form '*objectname = expression*', where *objectname* is the name of a valid object. Defines what occurs when the event is triggered. For more information, see `EventFcns`.
PropertyName	Property name for an event object from "Property Summary" on page 2-24.
PropertyValue	Property value. For more information on property values, see the property reference for each property listed in "Property Summary" on page 2-24.

Description

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

object (`modelObj`) to the property `Parent`. In the model object, this method appends the event object to the property `Events`.

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

For details on how events are handled during a simulation, see "Event Object".

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

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

Method Summary

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

Property Summary

Property	Description
`Active`	Indicate object in use during simulation
`EventFcns`	Event expression
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object

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

Examples

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

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

2 Get a list of properties for an event object.

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

Or

```
get(eventObj)
```

MATLAB displays a list of event properties.

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

See Also

`Event object`

addkineticlaw (reaction)

Purpose

Create kinetic law object and add to reaction object

Syntax

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

Arguments

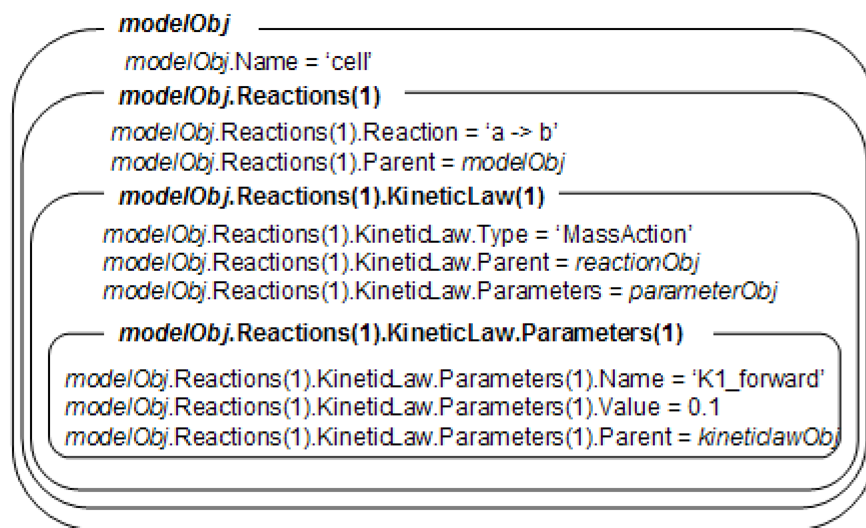
reactionObj	Reaction object. Enter a variable name for a reaction object.
KineticLawNameValue	Property to select the type of kinetic law object to create. For built-in kinetic law, valid values are: `'Unknown'`, `'MassAction'`, `'Henri-Michaelis-Menten'`, `'Henri-Michaelis-Menten-Reversible'`, `'Hill-Kinetics'`, `'Iso-Uni-Uni'`, `'Ordered-Bi-Bi'`, `'Ping-Pong-Bi-Bi'`, `'Competitive-Inhibition'`, `'NonCompetitive-Inhibition'`, and `'UnCompetitive-Inhibition'`. Find valid *KineticLawNameValue*s by using `sbioroot` to create a SimBiology root object, then query the object with the commands `get(rootObj.BuiltInLibrary, 'KineticLaws')` and `get(rootObj.UserDefinedLibrary, 'KineticLaws')`. `sbiowhos -kineticlaw` lists kinetic laws in the SimBiology root, which includes kinetic laws from both the `BuiltInLibrary` and the `UserDefinedLibrary`.

Description

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

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

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



KineticLawNameValue is any valid kinetic law definition. See "Kinetic Law Definition" on page 3-65 for a definition of kinetic laws and more information about how they are used to get the reaction rate expression.

kineticlawObj= addkineticlaw(..., '*PropertyName*', *PropertyValue*, ...) constructs a kinetic law object, *kineticlawObj*, and configures

`kineticlawObj` with property value pairs. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The `kineticlawObj` properties are listed in "Property Summary" on page 2-29.

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

Method Summary

Methods for kinetic law objects

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

Property Summary

Properties for kinetic law objects

Property	Description
`Expression (AbstractKineticLaw, KineticLaw)`	Expression to determine reaction rate equation
`KineticLawName`	Name of kinetic law applied to reaction
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parameters`	Array of parameter objects
`ParameterVariableNames`	Cell array of reaction rate parameters
`ParameterVariables`	Parameters in kinetic law definition
`Parent`	Indicate parent object
`SpeciesVariableNames`	Cell array of species in reaction rate equation
`SpeciesVariables`	Species in abstract kinetic law
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

Examples

Example 1

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

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

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

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

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

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

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

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

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

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

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

The `addparameter` method creates two parameter objects with values that are associated with the kinetic law parameters.

4 Associate kinetic law parameters with the parameters in the kinetic law definition.

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

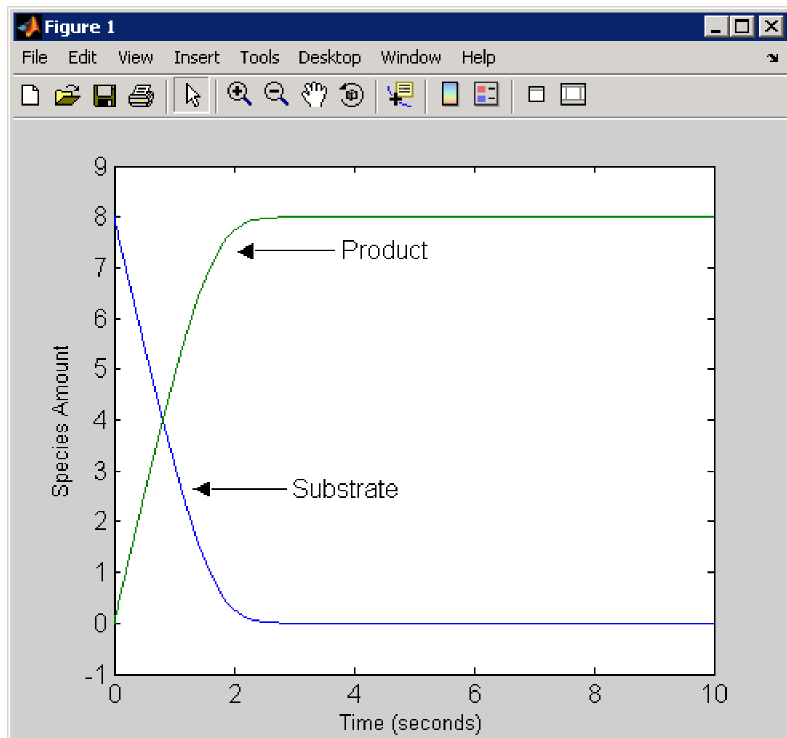
This method associates the parameters in the property `ParameterVariableNames` with the parameters in the property `ParameterVariables` using a one-to-one mapping in the order given.

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

```
get (reactionObj, 'ReactionRate')

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

6 Enter an initial value for the substrate and simulate.

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



Example 2

This example uses the built-in kinetic law `MassAction`.

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

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

2 Define a kinetic law for the reaction object.

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

Notice, the property `Expression` for `MassAction` kinetic law does not show the parameters and species in the reaction rate.

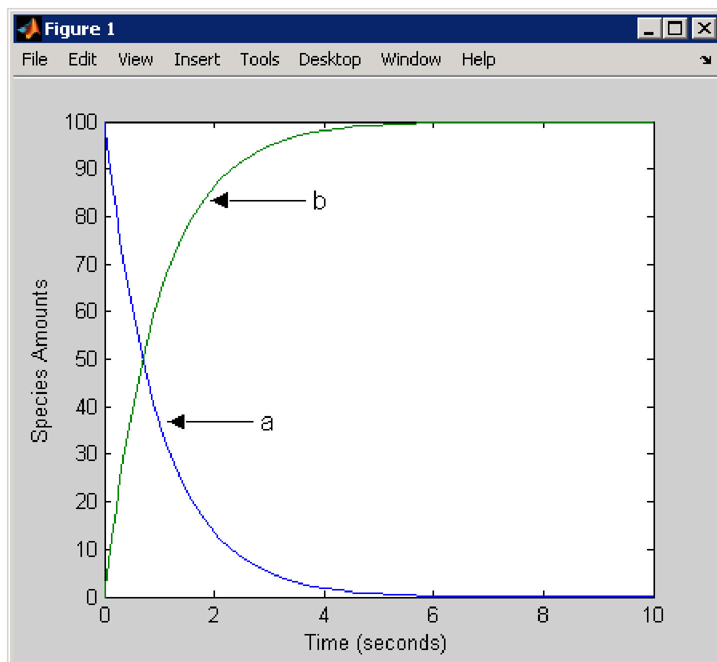
3 Assign the rate constant for the reaction.

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

ans =
    k_forward*a
```

4 Enter an initial value for the substrate and simulate.

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

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



See Also

addreaction, setparameter

addparameter (model, kineticlaw)

Purpose

Create parameter object and add to model or kinetic law object

Syntax

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

Arguments

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

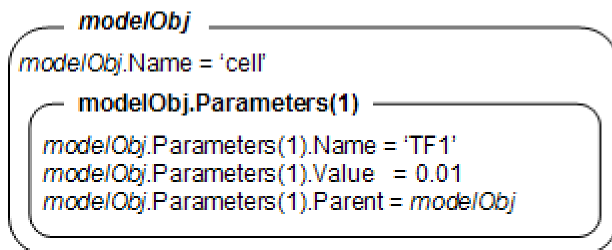
Description

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

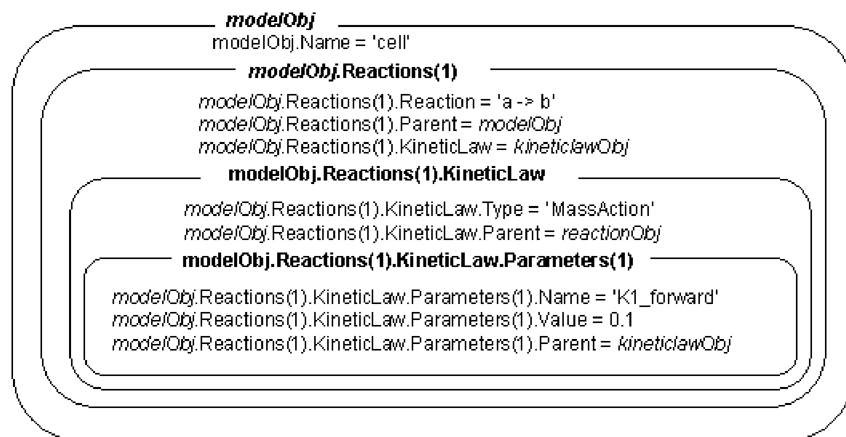
A parameter object defines an assignment that a model or a kinetic law can use. The scope of the parameter is defined by the parameter parent. If a parameter is defined with a kinetic law object, then only the kinetic law object and objects within the kinetic law object can use the

parameter. If a parameter object is defined with a model object as its parent, then all objects within the model (including all rules, events and kinetic laws) can use the parameter.

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




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



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

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

Scope of a parameter — A parameter can be *scoped* to either a model or a kinetic law.

- When a kinetic law searches for a parameter in its expression, it first looks in the parameter list of the kinetic law. If the parameter isn't found there, it moves to the model that the kinetic law object is in and looks in the model parameter list. If the parameter isn't found there, it moves to the model parent.
- When a rule searches for a parameter in its expression, it looks in the parameter list for the model. If the parameter isn't found there, it moves to the model parent. A rule cannot use a parameter that is scoped to a kinetic law. So for a parameter to be used in both a reaction rate equation and a rule, the parameter should be *scoped* to a model.

Additional parameter object properties can be viewed with the `get` command. Additional parameter object properties can be modified with the `set` command. The parameters of `Obj` can be viewed with `get(Obj, 'Parameters')`.

A SimBiology parameter object can be copied to a SimBiology model or kinetic law object with `copyobj`. A SimBiology parameter object can be removed from a SimBiology model or kinetic law object with `delete`.

Method Summary

Methods for parameter objects

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

Property Summary

Properties for parameter objects

`ConstantValue`	Specify variable or constant parameter value
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object
`Value`	Assign value to parameter object
`ValueUnits`	Parameter value units

Example

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

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

2 Define a kinetic law for the reaction object.

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

3 Add a parameter and assign it to the kinetic law object (`kineticlawObj`); add another parameter and assign to the model object (`modelObj`).

```
% Add parameter to kinetic law object
parameterObj1 = addparameter (kineticlawObj, 'K1');

get (kineticlawObj, 'Parameters')
```

MATLAB returns:

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1         K1       1

% Add parameter with value 0.9 to model object
parameterObj1 = addparameter (modelObj, 'K2', 0.9);

get (modelObj, 'Parameters')
```

MATLAB returns:

```
SimBiology Parameter Array

 Index:    Name:    Value:    ValueUnits:
   1         K2       1
```

See Also

`addreaction`

Purpose

Add product species object to reaction object

Syntax

```
speciesObj = addproduct(reactionObj, 'NameValue')
speciesObj = addproduct(reactionObj, speciesObj)
speciesObj = addproduct(reactionObj, 'NameValue',
   Stoichcoefficient)
speciesObj = addproduct(reactionObj, speciesObj,
   Stoichcoefficient)
```

Arguments

reactionObj	Reaction object. Enter a name for the reaction object.
NameValue	Property of a species object that names the object (not the reaction object). Enter a unique character string. For example, `'fructose 6-phosphate'`. A species object can be referenced by other objects using this property. You can use the function `sbioselect` to find an object with a specific `NameValue`.
speciesObj	Species object.
Stoichcoeffieient	Stoichiometric coefficients for products, length of array equal to length of `NameValue`, or length of *speciesObj*.

Description

speciesObj = addproduct(*reactionObj*, '*NameValue*') creates a species object and returns the species object (*speciesObj*). In the species object, this method assigns the value (*NameValue*) to the property `Name`. In the reaction object, this method assigns the species object to the property `Products`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `1` to the property `Stoichiometry`.

When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called `'unnamed'`) in the model and adds the newly created species to that compartment.
- If only one compartment object (`compObj`) exists in the model, the method creates a species object in that compartment.
- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

  For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Create and add a species object to a compartment object with the method `addspecies`.

`speciesObj = addproduct(reactionObj, speciesObj)`, in the species object (`speciesObj`), assigns the parent object of the `reactionObj` to the species property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Products`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `1` to the property `Stoichiometry`.

`speciesObj = addproduct(reactionObj, 'NameValue', Stoichcoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`. If `NameValue` is a cell array of species names, then `Stoichcoefficient` must be a vector of doubles with the same length as `NameValue`.

`speciesObj = addproduct(reactionObj, speciesObj, Stoichcoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species the reaction also uses the new name. You must however configure all other applicable elements such as rules that use the species, and the kinetic law object.

Examples

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'A + C -> U');
```

2. Modify the reaction of the `reactionObj` from `A + C -> U` to `A + C -> U + 2 H`.

```
speciesObj = addproduct(reactionObj, 'H', 2);
```

See Also

`addspecies`

addreactant (reaction)

Purpose

Add species object as reactant to reaction object

Syntax

```
speciesObj = addreactant(reactionObj, 'NameValue')
addreactant(reactionObj, speciesObj, StoichCoefficient)
addreactant(reactionObj, 'NameValue', StoichCoefficient)
```

Arguments

reactionObj	Reaction object.
NameValue	`Name` property of a species object. Enter a unique character string, for example, `'glucose'`. A species object can be referenced by other objects using this property.You can use the function `sbioselect` to find an object with a specific `Name` property value.
speciesObj	Species object or cell array of species objects.
StoichCoefficient	Stoichiometric coefficients for reactants, length of array equal to length of `NameValue` or length of *speciesObj*.

Description

speciesObj = addreactant(*reactionObj*, '*NameValue*') creates a species object and returns the species object (*speciesObj*). In the species object, this method assigns the value (*NameValue*) to the property `Name`. In the reaction object, this method assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `-1` to the property `Stoichiometry`.

When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (`compObj`) exists in the model, the method creates a species object in that compartment.

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

  For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Create and add a species object to a compartment object with the method `addspecies`.

`addreactant(`*`reactionObj`*`, `*`speciesObj`*`, `*`StoichCoefficient`*`)`, in the species object (`speciesObj`), assigns the parent object to the `speciesObj` property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `-1` to the property `Stoichiometry`. If `speciesObj` is a cell array of species objects, then *`StoichCoefficient`* must be a vector of doubles with the same length as `speciesObj`.

`addreactant(`*`reactionObj`*`, '`*`NameValue`*`', `*`StoichCoefficient`*`)`, in addition to the description above, adds the stoichiometric coefficient (*`StoichCoefficient`*) to the property `Stoichiometry`. If *`NameValue`* is a cell array of species names, then *`StoichCoefficient`* must be a vector of doubles with the same length as *`NameValue`*.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species the reaction also uses the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object.

See for more information on species names.

Example

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'A -> U');
```

2 Modify the `reaction` of `the reactionObj` from A -> U to be A + 3 C -> U.

```
speciesObj = addreactant(reactionObj, 'C', 3);
```

See Also

addspecies

Purpose

Create reaction object and add to model object

Syntax

```
reactionObj = addreaction(modelObj,'ReactionValue')
reactionObj = addreaction(modelObj, 'ReactantsValue',
   'ProductsValue')
reactionObj = addreaction(modelObj, 'ReactantsValue',
   RStoichCoefficients, 'ProductsValue', PStoichCoefficients)
reactionObj = addreaction(...'PropertyName', PropertyValue...)
```

Arguments

`modelObj`

SimBiology model object.

`ReactionValue`

Specify the reaction equation. Enter a character string. A hyphen preceded by a space and followed by a right angle bracket (->) indicates reactants going forward to products. A hyphen with left and right angle brackets (<->) indicates a reversible reaction. Coefficients before reactant or product names must be followed by a space.

Examples are `'A -> B'`, `'A + B -> C'`, `'2 A + B -> 2 C'`, and `'A <-> B'`. Enter reactions with spaces between the species.

If there are multiple compartments, or to specify the compartment name, use *`compartmentName.speciesName`*.

Examples are `'cytoplasm.A -> cytoplasm.B'`, `'cytoplasm.A -> nucleus.A'`, and `'cytoplasm.A + cytoplasm.B -> nucleus.AB'`.

`ReactantsValue`

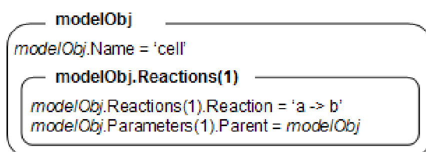
A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments.

ProductsValue	A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments.
RStoichCoefficients	Stoichiometric coefficients for reactants, length of array equal to length of *ReactantsValue*.
PStoichCoefficients	Stoichiometric coefficients for products, length of array equal to length of *ProductsValue*.

Description

reactionObj = addreaction(*modelObj*,'*ReactionValue*') creates a reaction object, assigns a value (*ReactionValue*) to the property Reaction, assigns reactant species object(s) to the property Reactants, assigns the product species object(s) to the property Products, and assigns the model object to the property Parent. In the Model object (modelObj), this method assigns the reaction object to the property Reactions, and returns the reaction object (reactionObj).

```
reactionObj = addreaction(modelObj, 'a -> b')
```



When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (compObj) exists in the model, the method creates a species object in that compartment.

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

  For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

You can manually add a species to a compartment object with the method `addspecies`.

You can add species to a reaction object using the methods `addreactant` or `addproduct`. You can remove species from a reaction object with the methods `rmreactant` or `rmproduct`. The property `Reaction` is modified by adding or removing species from the reaction equation.

You can copy a SimBiology reaction object to a model object with the function `copyobj`. You can remove the SimBiology reaction object from a SimBiology model object with the function `delete`.

You can view additional reaction object properties with the `get` command. For example, the reaction equation of `reactionObj` can be viewed with the command `get(`*`reactionObj`*`, 'Reaction')`. You can modify additional reaction object properties with the command `set`.

`reactionObj` `= addreaction(`*`modelObj`*`, '`*`ReactantsValue`*`', '`*`ProductsValue`*`')` creates a reaction object, assigns a value to the property `Reaction` using the reactant (*`ReactantsValue`*) and product (*`ProductsValue`*) names, assigns the species objects to the properties `Reactants` and `Products`, and assigns the model object to the property `Parent`. In the model object (`modelObj`), this method assigns the reaction object to the property `Reactions`, and returns the reaction object (`reactionObj`). The stoichiometric values are assumed to be `1`.

`reactionObj = addreaction(`*`modelObj`*`, '`*`ReactantsValue`*`', `*`RStoichCoefficients`*`, '`*`ProductsValue`*`', `*`PStoichCoefficients`*`)` adds stoichiometric coefficients (*`RStoichCoefficients`*) for reactant species, and stoichiometric coefficients (`PStoichCoefficients`) for product species to the property `Stoichiometry`. The length of `Reactants`

and `RCoefficients` must be equal, and the length of `Products` and `PCoefficients` must be equal.

`reactionObj = addreaction(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

Note If you use the `addreaction` method to create a reaction rate expression that is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" before simulating your model.

Method Summary

Methods for reaction objects

addkineticlaw (reaction)	Create kinetic law object and add to reaction object
addproduct (reaction)	Add product species object to reaction object
addreactant (reaction)	Add species object as reactant to reaction object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmproduct (reaction)	Remove species object from reaction object products

rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

Property Summary

Properties for reaction objects

`Active`	Indicate object in use during simulation
`KineticLaw`	Show kinetic law used for `ReactionRate`
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Products`	Array of reaction products
`Reactants`	Array of reaction reactants
`Reaction`	Reaction object reaction
`ReactionRate`	Reaction rate equation in reaction object
`Reversible`	Specify whether reaction is reversible or irreversible
`Stoichiometry`	Species coefficients in reaction
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

Examples

Create a model, add a reaction object, and assign the expression for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

`reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) and one species variable (`S`) that should to be set. To set these variables, first create the parameter variables as parameter objects (`parameterObj1, parameterObj2`) with names `Vm_d,` and `Km_d`, and assign the objects `Parent` property value to the `kineticlawObj`.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

4 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Vm_d' 'Km_d'});
set(kineticlawObj,'SpeciesVariableNames', {'a'});
```

5 Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
Vm_d*a/(Km_d+a)
```

See Also

addkineticlaw, addproduct, addreactant, rmproduct, rmreactant

addrule (model)

Purpose

Create rule object and add to model object

Syntax

```
ruleObj = addrule(modelObj, Rule)
ruleObj = addrule(modelObj, Rule, RuleType)
ruleObj = addrule(..., 'PropertyName', PropertyValue,...)
```

Arguments

modelObj	Model object to which to add the rule.
Rule	String specifying the rule. For example, enter the algebraic rule `'Va*Ea + Vi*Ei - K2'`.
RuleType	String specifying the type of rule. Choices are: • `'algebraic'` • `'initialAssignment'` • `'repeatedAssignment'` • `'rate'` For more information, see `RuleType`

Description

A rule is a mathematical expression that changes the amount of a species or the value of a parameter. It also defines how species and parameters interact with one another.

`ruleObj = addrule(modelObj, Rule)` constructs and returns *`ruleObj`*, a rule object. In *`ruleObj`*, the rule object, this method assigns the *`modelObj`* input argument to the `Parent` property, assigns the *`Rule`* input argument to the `Rule` property, and assigns `'initialAssignment'` or `'algebraic'` to the `RuleType` property. (This method assigns `'initialAssignment'` for all assignment rules and `'algebraic'` for all other rules.) In *`modelObj`*, the model object, this method assigns *`ruleObj`*, the rule object, to the `Rules` property.

`ruleObj = addrule(modelObj, Rule, RuleType)` in addition to the assignments above, assigns the *`RuleType`* input argument to the

RuleType property. For more information on the types of rules, see RuleType.

ruleObj = addrule(..., '*PropertyName*', *PropertyValue*,...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

View additional rule properties with the function get, and modify rule properties with the function set. Copy a rule object to a model with the function copyobj, or delete a rule object from a model with the function delete.

Note If you use the addrule method to create an algebraic rule, rate rule, or repeated assignment rule, and the rule expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" before simulating your model.

Method Summary

Methods for rule objects

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

Properties for rule objects

`Active`	Indicate object in use during simulation
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Rule`	Specify species and parameter interactions
`RuleType`	Specify type of rule for rule object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

Examples

Add a rule with the default `RuleType`.

1 Create a model object, and then add a rule object.

```
modelObj = sbiomodel('cell');
ruleObj = addrule(modelObj, '0.1*B-A')
```

2 Get a list of properties for a rule object.

```
get(modelObj.Rules(1)) or get(ruleObj)
```

MATLAB displays a list of rule properties.

```
    Active: 1
Annotation: ''
      Name: ''
```

```
      Notes: ''
     Parent: [1x1 SimBiology.Model]
       Rule: '0.1*B-A'
   RuleType: 'algebraic'
        Tag: ''
       Type: 'rule'
   UserData: []
```

Add a rule with the `RuleType` property set to `rate`.

1 Create model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a -> b');
```

2 Add a rule which defines that the quantity of a species `c`. In the rule expression, `k` is the rate constant for `a -> b`.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

3 Change the `RuleType` from default (`'algebraic'`) to `'rate'`, and verify using the `get` command.

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
    Active: 1
Annotation: ''
      Name: ''
     Notes: ''
    Parent: [1x1 SimBiology.Model]
      Rule: 'c = k*(a+b)'
  RuleType: 'rate'
       Tag: ''
      Type: 'rule'
  UserData: []
```

See Also

`copyobj`, `delete`, `sbiomodel`

Purpose

Create species object and add to compartment object within model object

Syntax

```
speciesObj = addspecies(compObj, 'NameValue')
speciesObj = addspecies(compObj, 'NameValue',
   InitialAmountValue)
speciesObj = addspecies(modelObj, 'NameValue')
speciesObj = addspecies(modelObj, 'NameValue',
   InitialAmountValue)
speciesObj = addspecies(...'PropertyName', PropertyValue...)
```

Arguments

compObj	Compartment object.
modelObj	Model object containing zero or one compartment.
NameValue	Name for a species object. Enter a character string unique among species within *modelObj* or *compObj*. Species objects are identified by name within `Event`, `ReactionRate`, and `Rule` property strings. For information on naming species, see `Name`. You can use the function `sbioselect` to find an object with a specific `Name` property value.
InitialAmountValue	Initial amount value for the species object. Enter `double`. Positive real number, default = `0`.
PropertyName	Enter the name of a valid property. Valid property names are listed in "Property Summary" on page 2-60.
PropertyValue	Enter the value for the property specified in *PropertyName*. Valid property values are listed on each property reference page.

Description

speciesObj = addspecies(*compObj*, '*NameValue*') creates speciesObj, a species object, and adds it to *compObj*, a compartment object. In the species object, this method assigns *NameValue* to the Name property, assigns *compObj* to the Parent property, and assigns 0 to the InitialAmount property. In the compartment object, this method adds the species object to the Species property.

speciesObj = addspecies(*compObj*, '*NameValue*', *InitialAmountValue*), in addition to the above, assigns *InitialAmountValue* to the InitialAmount property for the species object.

speciesObj = addspecies(*modelObj*, '*NameValue*') creates speciesObj, a species object, and adds it to *compObj*, the compartment object in *modelObj*, a model object. If *modelObj* does not contain any compartments, it creates *compObj* with a Name property of 'unnamed'. In the species object, this method assigns *NameValue* to the Name property, assigns *compObj* to the Parent property, and assigns 0 to the InitialAmount property. In the compartment object, this method adds the species object to the Species property.

speciesObj = addspecies(*modelObj*, '*NameValue*', *InitialAmountValue*), in addition to the above, assigns *InitialAmountValue* to the InitialAmount property for the species object.

You can also add a species to a reaction using the methods addreactant and addproduct.

A species object must have a unique name at the level at which it is created. For example, a compartment object cannot contain two species objects named H2O. However, another compartment can have a species named H2O.

View properties for a species object with the get command, and modify properties for a species object with the set command. You can view a summary table of species objects in a compartment (compObj) with get(compObj, 'Species') or the properties of the first species with get(compObj.Species(1)).

speciesObj = addspecies(...'*PropertyName*', *PropertyValue*...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs). The property summary on this page shows the list of properties.

If there is more than one compartment object (compObj) in the model, you must qualify the species name with the compartment name. For example, cell.glucose denotes that you want to put the species named glucose into a compartment named cell. Additionally, if the compartment named cell does not exist, the process of adding the reaction creates the compartment and names it cell.

If you change the name of a species you must configure all applicable elements, such as events and rules that use the species, any user-specified ReactionRate, or the kinetic law object property SpeciesVariableNames. Use the method setspecies to configure SpeciesVariableNames.

To update species names in the SimBiology graphical user interface, access each appropriate pane through the **Project Explorer**. You can also use the **Find** feature to locate the names that you want to update. The **Output** pane opens with the results of **Find**. Double-click a result row to go to the location of the model component.

Species names are automatically updated for reactions that use MassAction kinetic law.

Method Summary

Methods for species objects

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties

rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

Property Summary

Properties for species objects

`BoundaryCondition`	Indicate species boundary condition
`ConstantAmount`	Specify variable or constant species amount
`InitialAmount`	Species initial amount
`InitialAmountUnits`	Species initial amount units
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

Examples

Add two species to a model, where one is a reactant and the other is the enzyme catalyzing the reaction.

1. Create a model object named `my_model` and add a compartment object.

   ```
   modelObj = sbiomodel ('my_model');
   compObj = addcompartment(modelObj, 'comp1');
   ```

2 Add two species objects named `glucose_6_phosphate` and `glucose_6_phosphate_dehydrogenase`.

```
speciesObj1 = addspecies (compObj, 'glucose_6_phosphate');
speciesObj2 = addspecies (compObj, ...
                          'glucose_6_phosphate_dehydrogenase');
```

3 Set the initial amount of `glucose_6_phosphate` to `100` and verify.

```
set (speciesObj1, 'InitialAmount',100);
get (speciesObj1, 'InitialAmount')
```

MATLAB returns:

```
ans =

   100
```

4 Use `get` to note that `modelObj` contains the species object array.

```
get(compObj, 'Species')
```

MATLAB returns:

```
SimBiology Species Array

Index: Name:                              InitialAmount: InitialAmountUnits:
 1     glucose_6_phosphate                 100
 2     glucose_6_phosphate_dehydrogenase  0
```

5 Retrieve information about the first species in the array.

```
get(compObj.Species(1))
            Annotation: ''
     BoundaryCondition: 0
        ConstantAmount: 0
         InitialAmount: 100
    InitialAmountUnits: ''
                  Name: 'glucose_6_phosphate'
```

```
      Notes: ''
     Parent: [1x1 SimBiology.Compartment]
        Tag: ''
       Type: 'species'
   UserData: []
```

See Also

addcompartment, addproduct, addreactant, addreaction, get, set

Purpose

Add variant to model

Syntax

```
variantObj = addvariant(modelObj, 'NameValue')
variantObj2 = addvariant(modelObj, variantObj)
```

Arguments

modelObj	Specify the model object to which you want add a variant.
variantObj	Variant object to create and add to the model object.
NameValue	Name of the variant object. *NameValue* is assigned to the Name property of the variant object.

Description

variantObj = addvariant(*modelObj*, '*NameValue*') creates a SimBiology variant object (*variantObj*) with the name *NameValue* and adds the variant object to the SimBiology model object *modelObj*. The variant object Parent property is assigned the value of *modelObj*.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see Variant object.

variantObj2 = addvariant(*modelObj*, *variantObj*) adds a SimBiology variant object (*variantObj*) to the SimBiology model object and returns another variant object *variantObj2*. The variant object *variantObj2* Parent property is assigned the value of *modelObj*.

View properties for a variant object with the get command, and modify properties for a variant object with the set command.

Note Remember to use the addcontent method instead of using the set method on the Content property, because the set method replaces the data in the Content property, whereas addcontent appends the data.

To view the variants stored on a model object, use the `getvariant` method. To copy a variant object to another model, use `copyobj`. To remove a variant object from a SimBiology model, use the `delete` method.

Examples

1 Create a model containing one species.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
speciesObj = addspecies(compObj, 'A');
```

2 Add a variant object that varies the `InitialAmount` property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

See Also

`addcontent`, `commit`, `copyobj`, `delete`, `getvariant`

Purpose

Commit variant contents to model

Syntax

`commit(variantObj, modelObj)`

Arguments

modelObj	Specify the model object to which you want to commit a variant.
variantObj	Variant object to commit to the model object.

Description

`commit(variantObj, modelObj)` commits the `Contents` property of a SimBiology variant object (*variantObj*) to the model object *modelObj*. The property values stored in the variant object replace the values stored in the model.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant object`.

The `Contents` are set on the model object in order of occurrence, with duplicate entries overwriting. If the `commit` method finds an incorrectly specified entry, an error occurs and the remaining properties defined in the `Contents` property are not set.

Examples

1. Create a model containing one species.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
speciesObj = addspecies(compObj, 'A', 10);
```

2. Add a variant object that varies the `InitialAmount` property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

3. Commit the contents of the variant (*variantObj*).

```
commit (variantObj, modelObj);
```

See Also

`addvariant`, `Variant object`

Purpose

Options for compartments

Description

The SimBiology compartment object represents a container for species in a model. Compartment size can vary or remain constant during a simulation. All models must have at least one compartment and all species in a model must be assigned to a compartment. Compartment names must be unique within a model.

Compartments allow you to define the size (`Capacity`) of physically isolated regions that may affect simulation, and associate pools of species within those regions. You can specify or change `Capacity` using rules, events, and variants, similar to species amounts or parameter values.

The model object stores compartments as a flat list. Each compartment stores information on its own organization; in other words a compartment has information on which compartment it lives within (`Owner`) and who it contains (`Compartments`).

The flat list of compartments in the model object lets you vary the way compartments are organized in your model without invalidating any expressions.

To add species that participate in reactions, add the reaction to the model using the `addreaction` method. When you define a reaction with a new species:

- If no compartment objects exist in the model, the `addreaction` method creates a compartment object (called `'unnamed'`) in the model and adds the newly created species to that compartment.
- If only one compartment object exists in the model, the method creates a species object in that compartment.
- If there is more than one compartment object in the model, you must qualify the species name with the compartment name.

  For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Alternatively, create and add a species object to a compartment object, using the `addspecies` method at the command line.

When you use the SimBiology desktop to create a new model, it adds an empty compartment (*`unnamed`*), to which you can add species.

You can specify reactions that cross compartments using the syntax *`compartment1Name.species1Name`* –> *`compartment2Name.species2Name`*. If you add a reaction that contains species from different compartments, and the reaction rate dimensions are concentration/time, all reactants should be from the same compartment.

In addition, if the reaction is reversible then there are two cases:

- If the kinetic law is `MassAction`, and the reaction rate reaction rate dimensions are concentration/time, then the products must be from the same compartment.
- If the kinetic law is not `MassAction`, then both reactants and products must be in the same compartment.

See "Property Summary" on page 2-69 for links to compartment property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addcompartment (model, compartment)	Create compartment object

Method Summary

Methods for compartment objects

addcompartment (model, compartment)	Create compartment object
addspecies (model, compartment)	Create species object and add to compartment object within model object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties

Property Summary

Properties for compartment objects

`Capacity`	Compartment capacity
`CapacityUnits`	Compartment capacity units
`Compartments`	Array of compartments in model or compartment
`ConstantCapacity`	Specify variable or constant compartment capacity
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Owner`	Owning compartment

`Parent`	Indicate parent object
`Species`	Array of species in compartment object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`AbstractKineticLaw object`, `Configset object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`

Purpose

Solver settings information for model simulation

Description

The SimBiology configset object, also known as the configuration set object, contains the options that the solver uses during simulation of the model object. The configuration set object contains the following options for you to choose:

- Type of solver
- Stop time for the simulation
- Solver error tolerances, and for ode solvers — the maximum time step the solver should take
- Whether to perform sensitivity analysis during simulation
- Whether to perform dimensional analysis and unit conversion during simulation
- Species and parameter input factors for sensitivity analysis

A SimBiology model can contain multiple configsets with one being active at any given time. The active configset contains the settings that are used during the simulation. Use the method `setactiveconfigset` to define the active configset. Use the method `getconfigset` to return a list of configsets contained by a model. Use the method `addconfigset` to add a new configset to a model.

See "Property Summary" on page 2-72 for links to configset object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

Constructor Summary

addconfigset (model)	Create configuration set object and add to model object

Configset object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
set (any object)	Set object properties

Property Summary

`Active`	Indicate object in use during simulation
`CompileOptions`	Dimensional analysis and unit conversion options
`MaximumNumberOfLogs`	Maximum number of logs criteria to stop simulation
`MaximumWallClock`	Maximum elapsed wall clock time to stop simulation
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`RuntimeOptions`	Options for logged species
`SensitivityAnalysisOptions`	Specify sensitivity analysis options
`SolverOptions`	Specify model solver options
`SolverType`	Select solver type for simulation
`StartTime`	Start time for initial dose time
`StopTime`	Simulation time criteria to stop simulation

`TimeUnits`	Show time units for dosing and simulation
`Type`	Display SimBiology object type

See Also

`AbstractKineticLaw object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`, `Species object`

construct (PKModelDesign)

Purpose

Construct SimBiology model from `PKModelDesign` object

Syntax

```
[modelObj, pkModelMapObject] =
construct(pkModelDesignObject)
[modelObj, pkModelMapObject,
   CovModelObj] = construct(pkModelDesignObject)
```

Arguments

`modelObj`	SimBiology model object specifying a pharmacokinetic model.
`pkModelMapObject`	Defines the roles of the components in *`modelObj`*. For details, see `PKModelMap object`.
`CovModelObj`	Defines the relationship between parameters and covariates. For details, see `CovariateModel object`.

Description

`[`*`modelObj`*`, `*`pkModelMapObject`*`] = construct(`*`pkModelDesignObject`*`)` constructs a SimBiology model object, *`modelObj`*, containing the model components (such as compartments, species, reactions, and rules) required to represent the pharmacokinetic model specified in *`pkModelDesignObject`*. It also constructs *`pkModelMapObject`*, a `PKModelMap` object, which defines the roles of the model components.

The newly constructed model object, *`modelObj`*, is named `'Generated Model'` (which you can change). It contains one compartment for each compartment specified in the `PKCompartment` property of *`pkModelDesignObject`*. Each compartment contains a species that represents a drug concentration. The compartments are connected with reversible reactions that models flux between compartments.

`[`*`modelObj`*`, `*`pkModelMapObject`*`, `*`CovModelObj`*`] = construct(`*`pkModelDesignObject`*`)` constructs *`CovModelObj`*, a `CovariateModel` object, which defines the relationship between parameters and covariates. Within the `Expression` property of

`CovModelObj`, each parameter being estimated has an expression of the form `parameterName = exp(theta1 + eta1)` (without covariate dependencies), where `theta1` is a fixed effect, and `eta1` is a random effect. You can modify the expressions to add covariate dependencies. For details, see `CovariateModel` object.

See Also

`PKModelDesign object` | `PKModelMap object` | `CovariateModel object`

How To

- "Creating Pharmacokinetic Model Using the Command Line"
- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- "Specifying a Covariate Model"

constructDefaultFixedEffectValues (covmodel)

Purpose

Create initial estimate vector needed for fit

Syntax

```
FEInitEstimates =
constructDefaultFixedEffectValues(CovModelOb
   j)
```

Description

FEInitEstimates = constructDefaultFixedEffectValues(*CovModelObj*) creates *FEInitEstimates*, a structure containing the initial estimates for the fixed effects in *CovModelObj*, a CovariateModel object. These initial estimates are set to a default of zero, but you can edit these estimates. The number and names of the fields in the *FEInitEstimates* structure matches the number and names of fixed effects (theta values) in the Expression property of *CovModelObj*.

Tip After creating the *FEInitEstimates* structure, you can edit it and use it to change the FixedEffectValues property of *CovModelObj*, before using the object as an input argument to sbionlmefit or sbionlmefitsa.

See Also

CovariateModel | Expression | FixedEffectValues | sbionlmefit | sbionlmefitsa

How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- "Specifying a Covariate Model"

copyobj (any object)

Purpose

Copy SimBiology object and its children

Syntax

```
copiedObj = copyobj(Obj, parentObj)
copiedObj = copyobj(modelObj)
```

Arguments

Obj — Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, reaction, rule, species, or variant object.

parentObj

If *copiedObj* is...	*parentObj* must be...
configuration set, event, reaction, rule, or variant object	model object
compartment object	compartment or model object
species object	compartment object
parameter object	model or kinetic law object
kinetic law object	reaction object
model object or abstract kinetic law object	`sbioroot`

modelObj — Model object to be copied.

copiedObj — Output returned by the `copyobj` method with the parent set as specified in input argument (*parentObj*).

Description

copiedObj = copyobj(*Obj*, *parentObj*) makes a copy of a SimBiology object (*Obj*) and returns a pointer to the copy (*copiedObj*). In the copied object (*copiedObj*), this method assigns a value (*parentObj*) to the property `Parent`.

`copiedObj = copyobj(modelObj)` makes a copy of a model object (*modelObj*) and returns the copy (*copiedObj*). In the copied model object (*copiedObj*), this method assigns the root object to the property `Parent`.

Note When the `copyobj` method copies a model, it resets the `StatesToLog` property to the default value. Similarly, the `Inputs` and `Outputs` properties are not copied but rather left empty. Thus, when you simulate a copied model you see results for the default states, unless you manually update these properties.

Examples

Create a reaction object separate from a model object, and then add it to a model.

1. Create a model object and add a reaction object.

   ```
   modelObj1 = sbiomodel('cell');
   reactionObj = addreaction(modelObj1, 'a -> b');
   ```

2. Create a copy of the `reaction object` and assign it to another `model object`.

   ```
   modelObj2 = sbiomodel('cell2');
   reactionObjCopy = copyobj(reactionObj, modelObj2);
   modelObj2.Reactions

   SimBiology Reaction Array

   Index:    Reaction:
    1         a -> b
   ```

See Also

`sbiomodel`, `sbioroot`

Purpose

Define relationship between parameters and covariates

Description

`CovariateModel` defines the relationship between estimated parameters and covariates.

Tip Use a `CovariateModel` object as an input argument to `sbionlmefit` or `sbionlmefitsa` to fit a model with covariate dependencies. Before using the `CovariateModel` object with either fitting function, set the `FixedEffectValues` property to specify the initial estimates for the fixed effects.

Construction

`CovModelObj` `= CovariateModel` creates an empty `CovariateModel` object.

`CovModelObj` `= CovariateModel(`*`Expression`*`)` creates a `CovariateModel` object with its `Expression` property set to *`Expression`*, a string or cell array of strings, where each string represents the relationship between a parameter being estimated and one or more covariates. *`Expression`* must denote fixed effects with the prefix `theta`, and random effects with the prefix `eta`. Each string in *`Expression`* must be in the form:

```
parameterName = relationship
```

This example of an expression string defines the relationship between a parameter (`volume`) and a covariate (`weight`), with fixed effects, but no random effects:

```
Expression = {'volume = theta1 + theta2*weight'};
```

This table illustrates expression formats for some common parameter-covariate relationships.

Parameter-Covariate Relationship	Expression Format
Linear with random effect	`Cl = theta1 + theta2*WEIGHT + eta1`
Exponential without random effect	`Cl = exp(theta_Cl + theta_Cl_WT*WEIGHT)`
Exponential, WEIGHT centered by mean, has random effect	`Cl = exp(theta1 + theta2*(WEIGHT - mean(WEIGHT)) + eta1)`
Exponential, log(WEIGHT), which is equivalent to power model	`Cl = exp(theta1 + theta2*log(WEIGHT) + eta1)`
Exponential, dependent on WEIGHT and AGE, has random effect	`Cl = exp(theta1 + theta2*WEIGHT + theta3*AGE + eta1)`

Tip To simultaneously fit data from multiple dose levels, use a `CovariateModel` object as an input argument to `sbionlmefit`, and omit the random effect (`eta`) from the `Expression` property in the `CovariateModel` object.

Tip Use the `getCovariateData` method of a `PKData` object to view the covariate data when writing equations for the *`Expression`* input argument.

Note You can also construct a `CovariateModel` object using the `construct` method of a `PKModelDesign` object. However, the `Expression` property of the `CovariateModel` object does not include covariate dependencies. You can modify the expressions to add covariate dependencies. For details, see `Expression`.

Method Summary

constructDefaultFixedEffectValues (covmodel)	Create initial estimate vector needed for fit
verify (covmodel)	Check covariate model for errors

Property Summary

`CovariateLabels (CovariateModel)`	Labels for covariates in `CovariateModel` object
`Expression (CovariateModel)`	Define relationship between parameters and covariates
`FixedEffectDescription (CovariateModel)`	Descriptions of fixed effects in `CovariateModel` object
`FixedEffectNames (CovariateModel)`	Names of fixed effects in `CovariateModel` object
`FixedEffectValues (CovariateModel)`	Values for initial estimates of fixed effects in `CovariateModel` object
`ParameterNames (CovariateModel)`	Names of parameters in `CovariateModel` object
`RandomEffectNames (CovariateModel)`	Names of random effects in `CovariateModel` object

Examples

Create a `CovariateModel` object and set the `Expression` property to define the relationship between two parameters (clearance and volume)

and two covariates (weight and age) using fixed effects (`thetas`) and random effects (`etas`):

```
covModelObj = CovariateModel
covModelObj.Expression = {'CL = theta1 + theta2*WT + eta1', 'V = theta3 +
```

See Also

`construct` | `getCovariateData` | `PKData object` | `PKModelDesign object` | `sbionlmefit` | `sbionlmefitsa`

How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”

Purpose

Delete SimBiology object

Syntax

`delete(Obj)`

Arguments

Obj	abstract kinetic law, configuration set, event, kinetic law, model, parameter, reaction, rule, SimData, species, unit, unit prefix, or variant object.

Description

`delete(Obj)` removes an object (*Obj*) from its parent.

- If *Obj* is a model object, the model is deleted from the root object. `delete` removes all references to the model both at the command line and in the SimBiology desktop.
- If *Obj* is a species object that is being used by a reaction object, this method returns an error and the species object is not deleted. You need to delete the reaction or remove the species from the reaction before you can delete the species object.
- If *Obj* is a parameter object being used by a kinetic law object, there is no warning when the object is deleted. However, when you try to simulate your model, a error occurs because the parameter cannot be found.
- If *Obj* is a reaction object, this method deletes the object, but the species objects that were being used by the reaction object are not deleted.
- If *Obj* is an abstract kinetic law object and there is a kinetic law object referencing it, this method returns an error.
- If *Obj* is a SimBiology configuration set object, and it is the active configuration set object, this method, after deleting the object, makes the default configuration set object active. Note that you cannot delete the default configuration set.
- You cannot delete the SimBiology `root`.

You can also delete all model objects from the root with one call to the `sbioreset` function.

Examples

Example 1

Delete a reaction from a model. Notice the species objects are not deleted with the reaction object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'a -> b');
delete(reactionObj)
```

Example 2

Delete a single model from the root object.

```
modelObj1 = sbiomodel('cell');
modelObj2 = sbiomodel('virus');
delete(modelObj2)
```

See Also

`sbiomodel`, `sbioreset`, `sbioroot`

Purpose

Display summary of SimBiology object

Syntax

`display(Obj)`

Arguments

Obj	SimBiology object: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, rule, species, or unit.

Description

Display the SimBiology object array. `display(Obj)` is called for the SimBiology object, ***Obj*** when the semicolon is not used to terminate a statement. The display of ***Obj*** gives a brief summary of the ***Obj*** configuration. You can view a complete list of ***Obj*** properties with the command `get`. You can modify all ***Obj*** properties that can be changed, with the command `set`.

Examples

```
modelObj = sbiomodel('cell')
reactionObj = addreaction(modelObj, 'A + B -> C')
```

Event object

Purpose

Store event information

Description

Events are used to describe sudden changes in model behavior. An event lets you specify discrete transitions in model component values that occur when a user-specified condition become true. You can specify that the event occurs at a particular time, or specify a time-independent condition.

For details on how events are handled during a simulation, see "Event Object".

See "Property Summary" on page 2-87 for links to event property reference pages.

Properties define the characteristics of an object. For example, an event object includes properties that allow you to specify the conditions to trigger an event (`Trigger`), and what to do after the event is triggered (`EventFcn`). Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

Constructor Summary

addevent (model)	Add event object to model object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

`Active`	Indicate object in use during simulation
`EventFcns`	Event expression
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Tag`	Specify label for SimBiology object
`Trigger`	Event trigger
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`AbstractKineticLaw object`, `Configset object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`, `Species object`

export (model)

Purpose

Export SimBiology model

Syntax

```
exportedModel = export(model)
exportedModel = export(model,editvals)
exportedModel = export(model,editvals,editdoses)
```

Arguments

`model`	Model object.
`editvals`	Vector of species, parameter, and compartment objects that are editable in the exported model.
`editdoses`	Vector of dose objects that are editable in the exported model.

Description

`exportedModel = export(model)` exports a model, including all its active doses, and returns an exported model object, `SimBiology.export.Model`. By default, all species, parameters, compartments, and doses are editable in the exported model.

The exported SimBiology model object is a subclass of `hgsetget`. Thus, you can view properties for an exported model object with the `get` command, and modify properties for an exported model object with the `set` command.

`exportedModel = export(model,editvals)` specifies a vector of species, parameter, and compartment objects that are editable in the exported model. All doses in the exported model are editable.

`exportedModel = export(model,editvals,editdoses)` additionally specifies a vector of dose objects that are editable in the exported model.

Method Summary

Methods for exported model objects

accelerate	Prepare exported SimBiology model for acceleration
getdose	Return exported SimBiology model dose object
getIndex	Get indices into `ValueInfo` and `InitialValues` properties
isAccelerated	Determine whether an exported SimBiology model is accelerated
simulate	Simulate exported SimBiology model

Examples

Export a SimBiology model object.

```
modelObj = sbmlimport('lotka');
exportedModel = export(modelObj)

exportedModel =

  Model with properties:

           Name: 'lotka'
     ExportTime: '13-Dec-2012 13:52:11'
    ExportNotes: ''
```

Display the editable values (species, parameters, and compartments) information for the exported model object.

```
{exportedModel.ValueInfo.Name}

ans =

    'unnamed'    'x'    'y1'    'y2'    'z'    'c1'    'c2'    'c3'
```

There are 8 editable values in the exported model. Export the model again, allowing only the parameters (c1, c2, and c3) to be editable.

```
parameters = sbioselect(modelObj,'Type','parameter');
exportedModelParam = export(modelObj,parameters);
{exportedModelParam.ValueInfo.Name}
```

```
ans =

    'c1'    'c2'    'c3'
```

Export the model a third time, allowing the parameters and species to be editable.

```
PS = sbioselect(modelObj,'Type',{'species','parameter'});
exportedModelPS = export(modelObj,PS);
{exportedModelPS.ValueInfo.Name}
```

```
ans =

    'x'    'y1'    'y2'    'z'    'c1'    'c2'    'c3'
```

See Also

SimBiology.export.Model |

Related Examples

- "PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics"
- "Deploy a SimBiology Model"

Purpose

Get object properties

Syntax

```
PropertyValue = get(Obj, 'PropertyName')
objProperties = get(Obj)
```

Arguments

PropertyValue	Value defined for '*PropertyName*'
Obj	Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, `PKCompartment`, `PKData`, `PKModelDesign` `PKModelMap`, reaction, rule, `SimData`, species, or variant object.
'*PropertyName*'	Name of the property to get.
objProperties	`Struct` containing properties and values for the object, *Obj*.

Description

PropertyValue = get(*Obj*, '*PropertyName*') gets the value '*PropertyValue*' of the object, *Obj*'s *PropertyName* property.

objProperties = get(*Obj*) gets the properties for the object, *Obj*, and returns it to *objProperties*.

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2. Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

3. Set the `ConstantValue` property of the parameter object to `false` and verify.

   MATLAB returns `1` for `true` and `0` for `false`.

```
set (parameterObj, 'ConstantValue', false);
get(parameterObj, 'ConstantValue')
```

   MATLAB returns

```
ans =

     0
```

See Also

`getadjacencymatrix`, `getconfigset`, `getdata`, `getparameters`, `getsensmatrix`, `getspecies`, `getstoichmatrix`, `set`

Purpose

Get adjacency matrix from model object

Syntax

```
M = getadjacencymatrix(modelObj)
[M, Headings] = getadjacencymatrix(modelObj)
[M, Headings, Mask] = getadjacencymatrix(modelObj)
```

Arguments

M	Adjacency matrix for *modelObj*.
modelObj	Specify the model object.
Headings	Return row and column headings. If species are in multiple compartments, species names are qualified with the compartment name in the form `compartmentName.speciesName`. For example, `nucleus.DNA`, `cytoplasm.mRNA`.
Mask	Return `1` for the species object and `0` for the reaction object to *Mask*.

Description

`getadjacencymatrix` returns the adjacency matrix for a model object.

M = `getadjacencymatrix(`*modelObj*`)` returns an adjacency matrix for the model object (*modelOBJ*) to *M*.

An adjacency matrix is defined by listing all species contained by *modelObj* and all reactions contained by *modelObj* column-wise and row-wise in a matrix. The reactants of the reactions are represented in the matrix with a 1 at the location of [row of species, column of reaction]. The products of the reactions are represented in the matrix with a 1 at the location of [row of reaction, column of species]. All other locations in the matrix are 0.

[*M*, *Headings*] = `getadjacencymatrix(`*modelObj*`)` returns the adjacency matrix to *M* and the row and column headings to *Headings*. *Headings* is defined by listing all `Name` property values of species contained by *modelObj* and all `Name` property values of reactions contained by *modelObj*.

getadjacencymatrix (model)

`[M, Headings, Mask] = getadjacencymatrix(modelObj)` returns an array of 1s and 0s to *`Mask`*, where a 1 represents a species object and a 0 represents a reaction object.

Examples

1. Read in`m1`, a model object, using `sbmlimport`:

```
m1 = sbmlimport('lotka.xml');
```

2. Get the adjacency matrix for `m1`:

```
[M, Headings] = getadjacencymatrix(m1)
```

See Also

`getstoichmatrix`

Purpose

Get configuration set object from model object

Syntax

```
configsetObj = getconfigset(modelObj, 'NameValue')
configsetObj = getconfigset(modelObj)
configsetObj = getconfigset(modelObj,'active')
```

Arguments

modelObj	Model object. Enter a variable name for a model object.
NameValue	Name of the `configset` object.
configsetObj	Object holding the simulation-specific information.

Description

configsetObj = getconfigset(*modelObj*, '*NameValue*') returns the configuration set attached to `modelObj` that is named *NameValue*, to *configsetObj*.

configsetObj = getconfigset(*modelObj*) returns a vector of all attached configuration sets, to *configsetObj*.

configsetObj = getconfigset(*modelObj*,'active') retrieves the active configuration set.

A configuration set object stores simulation-specific information. A SimBiology model can contain multiple `configsets` with one being active at any given time. The active configuration set contains the settings that are used during the simulation.

Use the `setactiveconfigset` function to define the active `configset`. *modelObj* always contains at least one `configset` object with the name configured to `'default'`. Additional `configset` objects can be added to *modelObj* with the method `addconfigset`.

Examples

1 Retrieve the default `configset` object from the `modelObj`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
```

```
Configuration Settings - default (active)
  SolverType:                   ode15s
  StopTime:                     10

SolverOptions:
  AbsoluteTolerance:            1.000000e-06
  RelativeTolerance:            1.000000e-03
  SensitivityAnalysis:          false

RuntimeOptions:
  StatesToLog:                  all

CompileOptions:
  UnitConversion:               false
  DimensionalAnalysis:          true

SensitivityAnalysisOptions:
  Inputs:                       0
  Outputs:                      0
```

2 Configure the `SolverType` to `ssa`.

```
set(configsetObj, 'SolverType', 'ssa')
get(configsetObj)

                        Active: 1
                CompileOptions: [1x1 SimBiology.CompileOptions]
                          Name: 'default'
                         Notes: ''
                RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
    SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOpti
                 SolverOptions: [1x1 SimBiology.SSASolverOptions]
                    SolverType: 'ssa'
                      StopTime: 10
            MaximumNumberOfLogs: Inf
              MaximumWallClock: Inf
                     TimeUnits: 'second'
```

```
Type: 'configset'
```

See Also

`addconfigset`, `removeconfigset`, `setactiveconfigset`

getCovariateData (pkdata)

Purpose

Create design matrix needed for fit

Syntax

```
CovData = getCovariateData(PKDataObj)
```

Description

CovData = getCovariateData(*PKDataObj*) creates *CovData*, a dataset array containing only the covariate data from the data set in *PKDataObj*, a PKData object. *CovData* contains one row for each individual and one column for each covariate.

Tip Use the getCovariateData method to view the covariate data when writing equations for the Expression property of a CovariateModel object.

See Also

CovariateModel | Expression | PKData

How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”

Purpose Get data from SimData object array

Syntax

```
[t, x, names] = getdata(simDataObj)
[Out] = getdata(simDataObj, 'FormatValue')
```

Arguments

Output Arguments

t	An n-by-1 vector of time points.
x	An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
names	An m-by-1 cell array of names.
Metadata	When used with the 'nummetadata' input argument, *Metadata* contains a cell array of metadata structures. The elements of *Metadata* label the columns of *x*.
Out	Data returned in the format specified in '*FormatValue*', shown in "Input Arguments" on page 2-99. Depending on the specified '*FormatValue*', *Out* contains one of the following: • Structure array • SimData object • Time series object • Combined time series object from an array of SimData objects

Input Arguments

simDataObj	SimData object. Enter a variable name for a SimData object.
FormatValue	Choose a format from the following table.

FormatValue	Description
`'num'`	Specifies the format that lets you return data in numeric arrays. This is the default when `getdata` is called with multiple output arguments.
`'nummetadata'`	Specifies the format that lets you return a cell array of metadata structures in *`metadata`* instead of names. The elements of *`metadata`* label the columns of *`x`*.
`'numqualnames'`	Specifies the format that lets you return qualified names in *`names`* to resolve ambiguities.
`'struct'`	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
`'simdata'`	Specifies the format that lets you return data in a new SimData object. This format is more useful for SimData methods other than `getdata`.

FormatValue	Description
'ts'	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in simDataObj.
'tslumped'	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

Description

[*t*, *x*, *names*] = getdata(*simDataObj*) gets simulation time and state data from the SimData object simDataObj. When simDataObj contains more than one element, the outputs *t*, *x*, *names* are cell arrays in which each cell contains data for the corresponding element of simDataObj.

[*Out*] = getdata(*simDataObj*, '*FormatValue*') returns the data in the specified format. Valid formats are listed in "Input Arguments" on page 2-99.

Examples

Simulating and Retrieving Data

1. The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace and simulate the model.

```
sbioloadproject('radiodecay');
simDataObj = sbiosimulate(m1);
```

2. Get all the simulation data from the SimData object.

```
[t x names] = getdata(simDataObj);
```

Retrieving Data for Ensemble Runs

1. The project file, `radiodecay.sbproj`, contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

   ```
   sbioloadproject('radiodecay');
   ```

2. Change the solver to use during the simulation and perform an ensemble run.

   ```
   csObj  = getconfigset(m1);
   set(csObj, 'SolverType', 'ssa');
   simDataObj = sbioensemblerun(m1, 10);
   ```

3. Get all the simulation data from the SimData object.

   ```
   tsObjs = getdata(simDataObj(1:5), 'ts');
   ```

See Also

`display`, `get`, `resample`, `select`, `selectbyname`, `setactiveconfigset`

MATLAB function `struct`

Purpose

Return exported SimBiology model dose object

Syntax

```
doses = getdose(model)
doses = getdose(model,doseName)
```

Description

`doses = getdose(model)` returns all the `SimBiology.export.Dose` objects associated with the exported model.

`doses = getdose(model,doseName)` returns the export dose object with the `Name` property matching `doseName`.

Input Arguments

model

`SimBiology.export.Model` object.

doseName

String containing a dose name to match against the `Name` property of the export dose objects in `model`.

Default: All dose objects.

Output Arguments

doses

Export dose objects in `model`, or the export dose object with `Name` property `doseName`.

Examples

Retrieve SimBiology Model Dose Objects

Open a sample SimBiology model project, and export the included model object.

```
sbioloadproject('AntibacterialPKPD')
em = export(m1);
```

Display the editable doses in the exported model object.

```
doses = getdose(em)
```

```
doses = 

  1x4 RepeatDose array with properties:

    Interval
    RepeatCount
    StartTime
    TimeUnits
    Amount
    AmountUnits
    DurationParameterName
    LagParameterName
    Name
    Notes
    Parent
    Rate
    RateUnits
    TargetName
```

The exported model has 4 repeated dose objects. Display the dose names.

```
{doses.Name}
```

```
ans = 

    '250 mg bid'    '250 mg tid'    '500 mg bid'    '500 mg tid'
```

Extract only the 3rd dose object from the exported model object.

```
dose3 = getdose(em,'500 mg bid')
```

```
dose3 = 

  RepeatDose with properties:

               Interval: 12
            RepeatCount: 27
```

```
              StartTime: 0
              TimeUnits: 'hour'
                 Amount: 500
            AmountUnits: 'milligram'
  DurationParameterName: 'TDose'
       LagParameterName: ''
                   Name: '500 mg bid'
                  Notes: ''
                 Parent: 'Antibacterial'
                   Rate: 0
              RateUnits: ''
             TargetName: 'Central.Drug'
```

See Also

SimBiology.export.Model | SimBiology.export.Dose |

Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

getdose (model)

Purpose

Return SimBiology dose object

Syntax

```
doseObj = getdose(modelObj)
doseObj = getdose(modelObj, 'DoseName')
```

Arguments

modelObj	Selects a model object that contains a dose object.
DoseName	Name of a dose object contained in a model object. *DoseName* is from the dose object property, `Name`.

Outputs

doseObj	`ScheduleDose` or `RepeatDose` object retrieved from a model object. A `RepeatDose` or `ScheduleDose` object defines an increase (dose) to a species amount during a simulation.

Description

doseObj = `getdose(`*modelObj*`)` returns a Simbiology dose object (*doseObj*) contained in a Simbiology model object (*modelObj*).

doseObj = `getdose(`*modelObj*, '*DoseName*'`)` returns a SimBiology dose object (*modelObj*) with the name *DoseName*.

Examples

Get a dose object from a model object.

1. Create a model object, and then add a dose object to the model object.

```
modelObj = sbiomodel('mymodel');
dose1Obj = adddose(modelObj, 'dose1');
```

2. Get the `dose` object from a model object.

```
myModelDose = getdose(modelObj);
```

See Also

`Model object` methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

`ScheduleDose object` and `RepeatDose object` methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

Model.getequations

Purpose

Return system equations for model object

Syntax

```
equations = getequations(modelobj)
equations = getequations(modelobj,configsetobj,variantobj,
   doseobj)
```

Description

`equations = getequations(modelobj)` returns `equations`, a string containing the system of equations that represent `modelobj`, a `Model` object. The active `Configset` object is used to generate the equations and must specify a deterministic solver.

`equations = getequations(modelobj,configsetobj,variantobj,doseobj)` returns the system of equations that represent the model specified by a `Model` object, `Variant` objects, and dose objects (`RepeatDose` or `ScheduleDose`). The `Configset` object, `configsetobj`, is used to generate the equations and must specify a deterministic solver.

Tips

Use `getequations` to see the system of equations that represent a model for:

- Publishing purposes
- Model debugging

Input Arguments

modelobj

Object of the `Model` class.

Note If using `modelobj` as the only input argument, the active `Configset` object must specify a deterministic solver.

---

configsetobj

Object of the `Configset` class. This object must specify a deterministic solver.

Default: `[]` (Empty, which specifies the active `Configset` object for `modelobj`)

variantobj

Object or array of objects of the `Variant` class.

Default: `[]` (Empty, which specifies no variant object)

doseobj

Object or array of objects of the `RepeatDose` or `ScheduleDose` class.

Default: `[]` (Empty, which specifies no dose object)

Output Arguments

equations

String containing the system of equations that represent a model. This string includes equations for reactions, rules, events, variants, and doses.

Examples

View System of Equations for Simple Model

View system of equations that represent a simple model, containing only reactions.

Import the lotka model, included with SimBiology, into a variable named `model1`:

```
model1 = sbmlimport('lotka');
```

View all equations that represent the `model1` model and its active configset:

```
m1equations = getequations(model1)
```

```
m1equations =

ODEs:
```

```
d(y1)/dt = 1/unnamed*(ReactionFlux1 - ReactionFlux2)
d(y2)/dt = 1/unnamed*(ReactionFlux2 - ReactionFlux3)
d(z)/dt = 1/unnamed*(ReactionFlux3)

Fluxes:
ReactionFlux1 = c1*y1*x
ReactionFlux2 = c2*y1*y2
ReactionFlux3 = c3*y2

Parameter Values:
c1 = 10
c2 = 0.01
c3 = 10
unnamed = 1

Initial Conditions:
x = 1
y1 = 900
y2 = 900
z = 0
```

MATLAB displays the ODEs, fluxes, parameter values, and initial conditions for the reactions in `model1`.

View System of Equations for Model and Dose

View system of equations that represent a model, containing only reactions, and a repeated dose.

Import the lotka model, included with SimBiology, into a variable named `model1`:

```
model1 = sbmlimport('lotka');
```

Add a repeated dose to the model:

```
doseObj1 =  adddose(model1,'dose1','repeat');
```

Set the properties of the dose to administer 3 mg, at a rate of 10 mg/hour, 6 times, at an interval of every 24 hours, to species `y1`:

```
doseObj1.Amount = 0.003;
doseObj1.AmountUnits = 'gram';
doseObj1.Rate = 0.010;
doseObj1.RateUnits = 'gram/hour';
doseObj1.Repeat = 6;
doseObj1.Interval = 24;
doseObj1.TimeUnits = 'hour';
doseObj1.TargetName = 'y1';
```

View all equations that represent the `model1` model, its active configset, and the repeated dose:

```
m1_with_dose_equations = getequations (model1,[],[],doseObj1)

m1_with_dose_equations =

ODEs:
d(y1)/dt = 1/unnamed*(ReactionFlux1 - ReactionFlux2) + dose1
d(y2)/dt = 1/unnamed*(ReactionFlux2 - ReactionFlux3)
d(z)/dt = 1/unnamed*(ReactionFlux3)

Fluxes:
ReactionFlux1 = c1*y1*x
ReactionFlux2 = c2*y1*y2
ReactionFlux3 = c3*y2

Parameter Values:
c1 = 10
c2 = 0.01
c3 = 10
unnamed = 1

Initial Conditions:
y1 = 900
y2 = 900
```

```
z = 0
x = 1

Doses:
Variable                         Type                Units
dose1                            repeatdose          gram
```

MATLAB displays the ODEs, fluxes, parameter values, and initial conditions for the reactions and the dose in `model1`.

See Also

`Model object` | `Configset object` | `Variant object` | `RepeatDose object` | `ScheduleDose object`

Purpose

Get indices into `ValueInfo` and `InitialValues` properties

Syntax

```
indices = getIndex(model,name)
indices = getIndex(model,name,type)
```

Description

`indices = getIndex(model,name)` returns the indices of all `ValueInfo` objects in a `SimBiology.export.Model` object that have a `QualifiedName` or `Name` property that match the specified `name` input argument.

- `getIndex` first tries to match the `QualifiedName` property. If there are matches, then `getIndex` returns their indices.
- If there are no matches based on `QualifiedName`, then `getIndex` tries to match the `Name` property. If there are matches, then `getIndex` returns their indices.
- If there are no matches based on `QualifiedName` or `Name`, then `getIndex` returns `[]`.

`indices = getIndex(model,name,type)` returns indices for only the `ValueInfo` objects with a `Type` property that matches the `type` input argument.

Input Arguments

model

`SimBiology.export.Model` object.

name

String containing a name to match against the `QualifiedName`, then `Name`, properties of the `ValueInfo` objects in `model`.

type

String containing a name to match against the `Type` property of the `ValueInfo` objects in `model`.

Default: All types.

Output Arguments

indices

Vector of indices indicating which `ValueInfo` objects in a `SimBiology.export.Model` object match on the specified `name` and `type`.

Examples

Index Exported SimBiology Editable Values

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');
em = export(modelObj);
```

Get the index of the editable value with name `y1`.

```
ix = getIndex(em,'y1')
```

```
ix =

     3
```

Display the type of value.

```
em.ValueInfo(ix).Type
```

```
ans =

species
```

The name `y1` corresponds to an editable species.

See Also

`SimBiology.export.Model` | `SimBiology.export.ValueInfo` |

Related Examples

- "PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics"
- "Deploy a SimBiology Model"

Purpose

Get specific parameters in kinetic law object

Syntax

```
parameterObj = getparameters(kineticlawObj)
parameterObj = getparameters(kineticlawObj,
   'ParameterVariablesValue')
```

Arguments

kineticlawObj	Retrieve parameters used by the kinetic law object.
ParameterVariablesValue	Retrieve parameters used by the kinetic law object corresponding to the specified parameter in the `ParameterVariables` property of the kinetic law object.

Description

parameterObj = getparameters(*kineticlawObj*) returns the parameters used by the kinetic law object `kineticlawObj` to `parameterObj`.

parameterObj = getparameters(*kineticlawObj*, '*ParameterVariablesValue*') returns the parameter in the `ParameterVariableNames` property that corresponds to the parameter specified in the `ParameterVariables` property of `kineticlawObj`, to `parameterObj`. `ParameterVariablesValue` is the name of the parameter as it appears in the `ParameterVariables` property of `kineticlawObj`. `ParameterVariablesValue` can be a cell array of strings.

If you change the name of a parameter, you must configure all applicable elements such as rules that use the parameter, any user-specified `ReactionRate`, or the kinetic law object property `ParameterVariableNames`. Use the method `setparameter` to configure `ParameterVariableNames`.

Examples

Create a model, add a reaction, and assign the `ParameterVariableNames` for the reaction rate equation.

1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Add two parameter objects.

```
parameterObj1 = addparameter(kineticlawObj,'Va');
parameterObj2 = addparameter(kineticlawObj,'Ka');
```

4 The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) that should to be set. To set these variables:

```
setparameter(kineticlawObj,'Vm', 'Va');
setparameter(kineticlawObj,'Km', 'Ka');
```

5 To retrieve a parameter variable:

```
parameterObj3 = getparameters(kineticlawObj, 'Vm')
```

MATLAB returns:

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
  1         Va       1

parameterObj4 = getparameters (kineticlawObj, 'Km')
```

See Also

`addparameter`, `getspecies`, `setparameter`

Purpose

Get 3-D sensitivity matrix from SimData array

Syntax

```
[T, R, Outputs, InputFactors] = getsensmatrix(simDataObj)
[T, R, Outputs, InputFactors] = getsensmatrix(simDataObj,
   OutputNames,InputFactorNames)
```

Arguments

T	*T* is an `m-by-1` array specifying time points for the sensitivity data in R.
R	*R* is an `m-by-n-by-p` array of sensitivity data with times, outputs, and input factors corresponding to its first, second, and third dimensions respectively. `R(:,i,j)` is the time course for the sensitivity of state *`Outputs{i}`* to the input factor *`InputFactors{j}`*.
Outputs	Name of the output factors, where output factors are the names of the states for which you want to calculate sensitivity.
InputFactors	Name of the input factors, where input factors are the names of the states with respect to which you want to calculate sensitivity.

Description

`[T, R, Outputs, InputFactors] = getsensmatrix(simDataObj)` gets time and sensitivity data from the SimData object (*`simDataObj`*).

When *`simDataObj`* contains more than one element, the output arguments are cell arrays in which each cell contains data for the corresponding element of *`simDataObj`*.

The `getsensmatrix` method can only return sensitivity data that is contained in the SimData object. The sensitivity data that is logged in a SimData object is set at simulation time by the configuration set used during the simulation. This is typically the model's active configuration set. For an explanation of how to set up a sensitivity calculation using the configuration set, see "Sensitivity Analysis". Note in particular that

the sensitivity data `R` returned by `getsensmatrix` may be normalized, as specified at simulation time.

`[`*`T, R, Outputs, InputFactors`*`] = getsensmatrix(`*`simDataObj,OutputNames,InputFactorNames`*`)` gets sensitivity data for the outputs specified by *`OutputNames`* and the input factors specified by *`InputFactorNames`*.

`OutputNames` and *`InputFactorNames`* can both be any one of the following:

- Empty array
- Single name
- Cell array of names

Pass an empty array for *`OutputNames`* or *`InputFactorNames`* to ask for sensitivity data on all output factors or input factors contained in *`simDataObj`*, respectively. You can also use qualified names such as `'`*`CompartmentName.SpeciesName`*`'` or `'`*`ReactionName.ParameterName`*`'` to resolve ambiguities.

Examples

This example shows how to retrieve sensitivity data from a SimData object.

1 Set up the simulation:

a Import the radio decay model from SimBiology examples.

```
modelObj  = sbmlimport('radiodecay');
```

b Retrieve the configuration settings and the sensitivity analysis options from the `modelObj`.

```
configsetObj = getconfigset(modelObj);
sensitivityObj = get(configsetObj, 'SensitivityAnalysisOptions');
```

c Specify the species for which you want sensitivity data in the `Outputs` property. All model species are selected in this example.

Use the sbioselect function to retrieve the species objects from the model.

```
allSpeciesObj = sbioselect(modelObj, 'Type', 'species');
set(sensitivityObj, 'Outputs', allSpeciesObj);
```

d Specify species and parameters with respect to which you want to calculate the sensitivities in the Inputs property.

e Enable SensitivityAnalysis.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true)
get(configsetObj.SolverOptions, 'SensitivityAnalysis')

ans =

  1
```

f Simulate and return the results in a SimData object.

```
simDataObj = sbiosimulate(modelObj)
```

2 Extract and plot sensitivity data from the SimData object.

a Use getsensmatrix to retrieve sensitivity data.

```
[t R outs ifacs] = getsensmatrix(simDataObj);
```

b Plot sensitivity values.

```
plot(t, R(:,:,2));
legend(outs);
title(['Sensitivities of species relative to ' ifacs{2}]);
```

See Also

display, get, getdata, resample, selectbyname

MATLAB function struct

getspecies (kineticlaw)

Purpose

Get specific species in kinetic law object

Syntax

```
speciesObj = getspecies(kineticlawObj)
speciesObj = getspecies(kineticlawObj,
   'SpeciesVariablesValue')
```

Arguments

kineticlawObj	Retrieve species used by the kinetic law object.
SpeciesVariablesValue	Retrieve species used by the kinetic law object corresponding to the specified species in the `SpeciesVariables` property of the kinetic law object.

Description

`speciesObj = getspecies(kineticlawObj)` returns the species used by the kinetic law object `kineticlawObj` to `speciesObj`.

`speciesObj = getspecies(kineticlawObj, 'SpeciesVariablesValue')` returns the species in the `SpeciesVariableNames` property to `speciesObj`.

`SpeciesVariablesValue` is the name of the species as it appears in the `SpeciesVariables` property of `kineticlawObj`. `SpeciesVariablesValue` can be a cell array of strings.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the name of a species, the reaction updates to use the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object `SpeciesVariableNames`. Use the method `setspecies` to configure `SpeciesVariableNames`.

Examples

Create a model, add a reaction, and then assign the `SpeciesVariableNames` for the reaction rate equation.

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

`reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has one species variable (`S`) that should to be set. To set this variable:

```
setspecies(kineticlawObj,'S', 'a');
```

4 Retrieve the species variable using `getspecies`.

```
speciesObj = getspecies (kineticlawObj, 'S')
```

MATLAB returns:

```
SimBiology Species Array

Index:  Compartment: Name: InitialAmount: InitialAmountUnits:
   1      unnamed     a        0
```

See Also

`addspecies`, `getparameters`, `setparameter`, `setspecies`

getstoichmatrix (model)

Purpose

Get stoichiometry matrix from model object

Syntax

```
M = getstoichmatrix(modelObj)
[M,objSpecies] = getstoichmatrix(modelObj)
[M,objSpecies,objReactions] = getstoichmatrix(modelObj)
```

Arguments

M	Adjacency matrix for *modelObj*.
modelObj	Specify the model object *modelObj*.
objSpecies	Return the list of *modelObj* species by `Name` property of the species. If the species are in multiple compartments, species names are qualified with the compartment name in the form `compartmentName.speciesName`. For example, `nucleus.DNA`, `cytoplasm.mRNA`.
objReactions	Return the list of *modelObj* reactions by the `Name` property of reactions.

Description

`getstoichmatrix` returns a stoichiometry matrix for a model object.

M = `getstoichmatrix(`*modelObj*`)` returns a stoichiometry matrix for a SimBiology model object (*modelObj*) to *M*.

A stoichiometry matrix is defined by listing all reactions contained by *modelObj* column-wise and all species contained by *modelObj* row-wise in a matrix. The species of the reaction are represented in the matrix with the stoichiometric value at the location of [row of species, column of reaction]. Reactants have negative values. Products have positive values. All other locations in the matrix are 0.

For example, if *modelObj* is a model object with two reactions with names `R1` and `R2` and Reaction values of `2 A + B -> 3 C` and `B + 3 D -> 4 A`, the stoichiometry matrix would be defined as:

```
       R1    R2
A      -2     4
B      -1    -1
C       3     0
D       0    -3
```

[*M*,*objSpecies*] = getstoichmatrix(*modelObj*) returns the stoichiometry matrix to *M* and the species to *objSpecies*. *objSpecies* is defined by listing all Name property values of species contained by *Obj*. In the above example, *objSpecies* would be {'A', 'B', 'C', 'D'};.

[*M*,*objSpecies*,*objReactions*] = getstoichmatrix(*modelObj*) returns the stoichiometry matrix to *M* and the reactions to *objReactions*. *objReactions* is defined by listing all Name property values of reactions contained by *modelObj*. In the above example, *objReactions* would be {'R1', 'R2'}.

Examples

1. Read in m1, a model object, using sbmlimport:

```
m1 = sbmlimport('lotka.xml');
```

2. Get the stoichiometry matrix for the m1:

```
[M,objSpecies,objReactions] = getstoichmatrix(m1)
```

See Also

getadjacencymatrix, “Determining the Stoichiometry Matrix for a Model”

getvariant (model)

Purpose

Get variant from model

Syntax

```
variantObj = getvariant(modelObj)
variantObj = getvariant(modelObj, 'NameValue')
```

Arguments

variantObj	Variant object returned by the `getvariant` method.
modelObj	Model object from which to get the variant.
'*NameValue*'	Name of the variant to get from the model object *modelObj*.

Description

variantObj = getvariant(*modelObj*) returns SimBiology variant objects contained by the SimBiology model object *modelObj* to *variantObj*.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant object`.

variantObj = getvariant(*modelObj*, '*NameValue*') returns the SimBiology variant object with the name *NameValue*, contained by the SimBiology model object, *modelObj*.

View properties for a variant object with the `get` command, and modify properties for a variant object with the `set` command.

Note Remember to use the `addcontent` method instead of using the `set` method on the `Content` property, because the `set` method replaces the data in the `Content` property whereas `addcontent` appends the data.

To copy a variant object to another model, use `copyobj`. To remove a variant object from a SimBiology model, use the `delete` method.

Examples

1. Create a model containing several variants.

```
modelObj = sbiomodel('mymodel');
variantObj1 = addvariant(modelObj, 'v1');
variantObj2 = addvariant(modelObj, 'v2');
```

2. Get all variants in the model.

```
vObjs = getvariant(modelObj)

SimBiology Variant Array

   Index:  Name:              Active:
   1       v1                 false
   2       v2                 false
```

3. Get the variant object named `'v2'` from the model.

```
vObjv2 = getvariant(modelObj, 'v2');
```

See Also

`addvariant`, `removevariant`

SimBiology.export.Model.isAccelerated

Purpose

Determine whether an exported SimBiology model is accelerated

Syntax

```
tf = isAccelerated(model)
tf = isAccelerated(model,computerType)
```

Description

`tf = isAccelerated(model)` returns `true` if `model` is accelerated for the current type of computer, and `false` otherwise.

`tf = isAccelerated(model,computerType)` returns `true` if `model` is accelerated for the specified computer type.

Input Arguments

model

`SimBiology.export.Model` object.

computerType

String specifying a computer type. You can specify any valid `archstr` supported by the function `computer`.

Output Arguments

tf

Logical value `true` if `model` is accelerated for the current computer type, or computer type specified by `computerType`. Logical value `false` if the exported model is not accelerated for the specified computer type.

Examples

Accelerate Exported SimBiology Model

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');
em = export(modelObj)
```

```
em =

  Model with properties:

          Name: 'lotka'
```

```
     ExportTime: '12-Dec-2012 15:20:13'
    ExportNotes: ''
```

Accelerate the exported model.

```
accelerate(em);
em.isAccelerated
```

```
ans =

     1
```

The logical value 1 indicates that the exported model is accelerated.

See Also

SimBiology.export.Model | SimBiology.export.Model.accelerate | computer

Related Examples

- "PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics"
- "Deploy a SimBiology Model"

Purpose

Kinetic law information for reaction

Description

The kinetic law object holds information about the abstract kinetic law applied to a reaction and provides a template for the reaction rate. In the model, the SimBiology software uses the information you provide in a fully defined kinetic law object to determine the `ReactionRate` property in the reaction object.

When you first create a kinetic law object, you must specify the name of the abstract kinetic law to use. The SimBiology software fills in the `KineticLawName` property and the `Expression` property in the kinetic law object with the name of the abstract kinetic law you specified and the mathematical expression respectively. The software also fills in the `ParameterVariables` property and the `SpeciesVariables` property of the kinetic law object with the values found in the corresponding properties of the abstract kinetic law object.

To obtain the reaction rate, you must fully define the kinetic law object:

1. In the `ParameterVariableNames` property, specify the parameters from the model that you want to substitute in the expression (`Expression` property).

2. In the `SpeciesVariableNames` property, specify the species from the model that you want to substitute in the expression.

   The SimBiology software substitutes in the expression, the names of parameter variables and species variables in the order specified in the `ParameterVariables` and `SpeciesVariables` properties respectively.

   The software then shows the substituted expression as the reaction rate in the `ReactionRate` property of the reaction object. If the kinetic law object is not fully defined, the `ReactionRate` property remains `' '` (empty).

For links to kinetic law object property reference pages, see "Property Summary" on page 2-133.

Properties define the characteristics of an object. Use the get and set commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

For an explanation of how relevant properties relate to one another, see "Command Line" on page 2-129.

The following sections use a kinetic law example to show how you can fully define your kinetic law object to obtain the reaction rate in the SimBiology desktop and at the command line.

The Henri-Michaelis-Menten kinetic law is expressed as follows:

$$V_{\mathrm{m}} * S / (K_m + S)$$

In the SimBiology software Henri-Michaelis-Menten is a built-in abstract kinetic law, where V_m and K_m are defined in the ParameterVariables property of the abstract kinetic law object, and S is defined in the SpeciesVariables property of the abstract kinetic law object.

SimBiology Desktop

To fully define a kinetic law in the SimBiology desktop, define the names of the species variables and parameter variables that participate in the reaction rate.

Command Line

To fully define the kinetic law object at the command line, define the names of the parameters in the ParameterVariableNames property of the kinetic law object, and define the species names in the SpeciesVariableNames property of the kinetic law object. For example, to apply the Henri-Michaelis-Menten abstract kinetic law to a reaction

```
A -> B
where Vm = Va, Km = Ka
and S = A
```

Define Va and Ka in the ParameterVariableNames property to substitute the variables that are in the ParameterVariables property (Vm and Km). Define A in the SpeciesVariableName property to be used to substitute the species variable in the SpeciesVariables property (S). Specify the order of the model parameters to be used for substitution in the same order that the parameter variables are listed in the ParameterVariables property. Similarly, specify species order if more than one species variable is represented.

```
% Find the order of the parameter variables
% in the kinetic law expression.

get(kineticlawObj, 'ParameterVariables')

ans =

    'Vm'    'Km'

% Find the species variable in the
% kinetic law expression

get(kineticlawObj, 'SpeciesVariables')
ans =

    'S'

% Specify the parameters and species variables
% to be used in the substitution.
% Remember to specify order, for example Vm = Va
% Vm is listed first in 'ParameterVariables',
% therefore list Va first in 'ParameterVariableNames'.

set(kineticlawObj,'ParameterVariableNames', {'Va' 'Ka'});
set(kineticlawObj,'SpeciesVariableNames', {'A'});
```

The rate equation is assigned in the reaction object as follows:

Va*A/(Ka + A)

For a detailed procedure, see "Examples" on page 2-134.

The following table summarizes the relationships between the properties in the abstract kinetic law object and the kinetic law object in the context of the above example.

Property	Property Purpose	Abstract Kinetic Law Object	Kinetic Law Object
Name (abstract kinetic law object) KineticLawName (kinetic law object)	Name of abstract kinetic law applied to a reaction. For example: Henri-Michaelis-Menten	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only
Expression	Mathematical expression used to determine the reaction rate equation. For example: $V_m * S/(K_m + S)$	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only; depends on abstract kinetic law applied to reaction.
ParameterVariables	Variables in Expression that are parameters. For example: Vm and Km	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only; depends on abstract kinetic law applied to reaction.

Property	Property Purpose	Abstract Kinetic Law Object	Kinetic Law Object
SpeciesVariables	Variables in Expression that are species. For example: S	Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law.	Read-only; depends on abstract kinetic law applied to reaction.
ParameterVariableNames	Variables in ReactionRate that are parameters. For example: Va and Ka	Not applicable	Define these variables corresponding to ParameterVariables.
SpeciesVariablesNames	Variables in ReactionRate that are species. For example: A	Not applicable	Define these variables corresponding to SpeciesVariables.

Constructor Summary

addkineticlaw (reaction)	Create kinetic law object and add to reaction object

Method Summary

addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object

display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getparameters (kineticlaw)	Get specific parameters in kinetic law object
getspecies (kineticlaw)	Get specific species in kinetic law object
set (any object)	Set object properties
setparameter (kineticlaw)	Specify specific parameters in kinetic law object
setspecies (kineticlaw)	Specify species in kinetic law object

Property Summary

`Expression (AbstractKineticLaw, KineticLaw)`	Expression to determine reaction rate equation
`KineticLawName`	Name of kinetic law applied to reaction
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parameters`	Array of parameter objects
`ParameterVariableNames`	Cell array of reaction rate parameters
`ParameterVariables`	Parameters in kinetic law definition
`Parent`	Indicate parent object

SpeciesVariableNames	Cell array of species in reaction rate equation
SpeciesVariables	Species in abstract kinetic law
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

Examples

This example shows how to define the reaction rate for a reaction.

1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'A -> B');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Query the parameters and species variables defined in the kinetic law.

```
get(kineticlawObj, 'ParameterVariables')

ans =

    'Vm'    'Km'

get(kineticlawObj, 'SpeciesVariables')
ans =

    'S'
```

4 Define Va and Ka as ParameterVariableNames, which correspond to the ParameterVariables Vm and Km. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with the names Va and Ka, and then add them to kineticlawObj. The species object with Name A is created when reactionObj is created and need not be redefined.

```
parameterObj1 = addparameter(kineticlawObj, 'Va');
parameterObj2 = addparameter(kineticlawObj, 'Ka');
```

5 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Va' 'Ka'});
set(kineticlawObj,'SpeciesVariableNames', {'A'});
```

6 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Va*A/(Ka+A)
```

See Also

AbstractKineticLaw object, Configset object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object

SimBiology property Expression(AbstractKineticLaw, KineticLaw)

Model object

Purpose

Model and component information

Description

The SimBiology model object represents a *model*, which is a collection of interrelated reactions and rules that transform, transport, and bind species. The model includes model components such as compartments, reactions, parameters, rules, and events. Each of the components is represented as a property of the model object. A model object also has a default configuration set object to define simulation settings. You can also add more configuration set objects to a model object.

See "Property Summary" on page 2-138 for links to model property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

You can retrieve SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object. The root object contains a list of model objects that are accessible from the MATLAB command line and from the SimBiology desktop. Because both the command line and the desktop point to the same model object in the `Root object`, any changes you make to the model at the command line are reflected in the desktop, and vice versa.

Constructor Summary

`sbiomodel`	Construct model object

Method Summary

addcompartment (model, compartment)	Create compartment object
addconfigset (model)	Create configuration set object and add to model object
adddose (model)	Add dose object to model

addevent (model)	Add event object to model object
addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object
addreaction (model)	Create reaction object and add to model object
addrule (model)	Create rule object and add to model object
addspecies (model, compartment)	Create species object and add to compartment object within model object
addvariant (model)	Add variant to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
export (model)	Export SimBiology model
get (any object)	Get object properties
getadjacencymatrix (model)	Get adjacency matrix from model object
getconfigset (model)	Get configuration set object from model object
getdose (model)	Return SimBiology dose object
getequations	Return system equations for model object
getstoichmatrix (model)	Get stoichiometry matrix from model object
getvariant (model)	Get variant from model

removeconfigset (model)	Remove configuration set from model
removedose (model)	Add dose object to model
removevariant (model)	Remove variant from model
reorder (model, compartment)	Reorder component lists
set (any object)	Set object properties
setactiveconfigset (model)	Set active configuration set for model object
verify (model, variant)	Validate and verify SimBiology model

Property Summary

`Compartments`	Array of compartments in model or compartment
`Events`	Contain all event objects
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parameters`	Array of parameter objects
`Parent`	Indicate parent object
`Reactions`	Array of reaction objects
`Rules`	Array of rules in model object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`AbstractKineticLaw object`, `Configset object`, `KineticLaw object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`, `Species object`

Purpose

Parameter and scope information

Description

The parameter object represents a *parameter*, which is a quantity that can change or can be constant. SimBiology parameters are generally used to define rate constants. You can add parameter objects to a model object or a kinetic law object. The scope of a parameter depends on where you add the parameter object: If you add the parameter object to a model object, the parameter is available to all reactions in the model and the `Parent` property of the parameter object is `SimBiology.Model`. If you add the parameter object to a kinetic law object, the parameter is available only to the reaction for which you are using the kinetic law object and the `Parent` property of the parameter object is `SimBiology.KineticLaw`.

See "Property Summary" on page 2-141 for links to parameter object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addparameter (model, kineticlaw)	Create parameter object and add to model or kinetic law object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties

rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

Property Summary

ConstantValue	Specify variable or constant parameter value
Name	Specify name of object
Notes	HTML text describing SimBiology object
Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object
Value	Assign value to parameter object
ValueUnits	Parameter value units

See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Reaction object, Root object, Rule object, Species object

PKCompartment object

Purpose

Used by PKModelDesign to create SimBiology model

Description

The PKCompartment object is used by the PKModelDesign object to construct a SimBiology model for pharmacokinetic modeling. PKCompartment holds the following information:

- Name of the compartment
- Dosing type
- Elimination type
- Whether the drug concentration in this compartment is reported

The PKCompartment class is a subclass of the hgsetget class which is a subclass of the handle class. For more information on the inherited methods, see hgsetget, and handle.

Construction

addCompartment (PKModelDesign)	Add compartment to PKModelDesign object

Method Summary

get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

DosingType	Drug dosing type in compartment
EliminationType	Drug elimination type from compartment
HasLag	Lag associated with dose targeting compartment
HasResponseVariable	Compartment drug concentration reported
Name	Specify name of object

See Also

"Creating Pharmacokinetic Models" in the SimBiology User's Guide, `PKModelDesign object`

PKData object

Purpose

Define roles of data set columns

Description

The properties of the `PKData` object specify what each column in the data represents. The `PKData` object specifies which columns in the data set represent the following:

- The grouping variable
- The independent and dependent variables
- The dose
- The rate (only if infusion is the dosing type)
- The covariates

This information is used by the fitting functions, `sbionlmefit` and `sbionlinfit`.

To create the `PKData` object specify:

```
pkDataObject = PKData(data);
```

Where `data` is the imported data set.

The `PKData` class is a subclass of the `hgsetget` class, which is a subclass of the `handle` class. For more information on the inherited methods, see `hgsetget` and `handle`.

Construction

`PKData`	Create `PKData` object

Method Summary

get (any object)	Get object properties
getCovariateData (pkdata)	Create design matrix needed for fit
set (any object)	Set object properties

Property Summary

`CovariateLabels`	Identify covariate columns in data set
`DataSet`	`Dataset` object containing imported data
`DependentVarLabel`	Identify dependent variable column in data set
`DependentVarUnits`	Response units in PKData object
`DoseLabel`	Dose column in data set
`DoseUnits`	Dose units in PKData object
`GroupID`	Integer identifying each group in data set
`GroupLabel`	Identify group column in data set
`GroupNames`	Unique values from `GroupLabel` in data set
`IndependentVarLabel`	Identify independent variable column in data set
`IndependentVarUnits`	Time units in PKData object
`RateLabel`	Rate of infusion column in data set
`RateUnits`	Units for dose rate

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKModelDesign object`

PKModelDesign object

Purpose

Helper object to construct pharmacokinetic model

Description

Use the `PKModelDesign` object to construct a SimBiology model for PK modeling. The `PKModelDesign` object lets you specify the number of compartments, the type of dosing, and method of elimination which you then use to construct the SimBiology model object with the necessary compartments, species, reactions, rules, and events.

```
pkm = PKModelDesign;
```

Use the `addCompartment` method to add a compartment with a specified dosing and elimination. `addCompartment` adds each subsequent compartment and connects it to the previous compartment using a reversible reaction. This reaction models the flux between compartments in a PK model.

The `construct` method uses the `PKModelDesign` object to create a SimBiology model object.

The `PKModelDesign` class is a subclass of the `hgsetget` class, which is a subclass of the `handle` class. For more information on the inherited methods see `hgsetget` and `handle`.

Construction

`PKModelDesign`	Create `PKModelDesign` object

Method Summary

addCompartment (PKModelDesign)	Add compartment to `PKModelDesign` object
construct (PKModelDesign)	Construct SimBiology model from `PKModelDesign` object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

`PKCompartments`	Hold compartments in PK model

See Also

"Creating Pharmacokinetic Models" in the SimBiology User's Guide, `PKCompartment object`

PKModelMap object

Purpose

Define SimBiology model components' roles

Description

The `PKModelMap` object holds information about the dosing type, and defines which components of a SimBiology model represent the observed response, the dose, and the estimated parameters.

The `PKModelMap` class is a subclass of the `hgsetget` class which is a subclass of the `handle` class. For more information on the inherited methods see, `hgsetget`, and `handle`.

Construction

`PKModelMap`	Create `PKModelMap` object

Method Summary

get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

`Dosed`	Dosed object name
`DosingType`	Drug dosing type in compartment
`Estimated`	Names of parameters to estimate
`LagParameter`	Parameter specifying time lag for doses
`Observed`	Measured response object name
`ZeroOrderDurationParameter`	Zero-order dose absorption duration

See Also

"Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" in the SimBiology User's Guide, `PKModelDesign object`

Purpose

Options for model reactions

Description

The reaction object represents a *reaction*, which describes a transformation, transport, or binding process that changes one or more species. Typically, the change is the amount of a species. For example:

```
Creatine + ATP <-> ADP + phosphocreatine

glucose + 2 ADP + 2 Pi -> 2 lactic acid + 2 ATP + 2 H2O
```

Spaces are required before and after species names and stoichiometric values.

See "Property Summary" on page 2-150 for links to reaction object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addreaction (model)	Create reaction object and add to model object

Method Summary

addkineticlaw (reaction)	Create kinetic law object and add to reaction object
addproduct (reaction)	Add product species object to reaction object
addreactant (reaction)	Add species object as reactant to reaction object
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object

display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmproduct (reaction)	Remove species object from reaction object products
rmreactant (reaction)	Remove species object from reaction object reactants
set (any object)	Set object properties

Property Summary

`Active`	Indicate object in use during simulation
`KineticLaw`	Show kinetic law used for `ReactionRate`
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Products`	Array of reaction products
`Reactants`	Array of reaction reactants
`Reaction`	Reaction object reaction
`ReactionRate`	Reaction rate equation in reaction object
`Reversible`	Specify whether reaction is reversible or irreversible
`Stoichiometry`	Species coefficients in reaction
`Tag`	Specify label for SimBiology object

Type	Display SimBiology object type
UserData	Specify data to associate with object

See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Root object, Rule object, Species object

removeconfigset (model)

Purpose

Remove configuration set from model

Syntax

```
removeconfigset(modelObj, 'NameValue')
removeconfigset(modelObj, configsetObj)
```

Arguments

modelObj	Model object from which to remove the configuration set.
NameValue	Name of the configuration set.
configsetObj	Configuration set object that is to be removed from the model object.

Description

`removeconfigset(`*`modelObj`*`, '`*`NameValue`*`')` removes the configset object with the name *`NameValue`* from the SimBiology model object *`modelObj`*. A configuration set object stores simulation-specific information. A SimBiology model can contain multiple configuration sets with one being active at any given time. The active configuration set contains the settings that are used during the simulation. *`modelObj`* always contains at least one configuration set object with name configured to `'default'`. You cannot remove the default configuration set from *`modelObj`*. If the active configuration set is removed from *`modelObj`*, then the default configuration set will be made active.

`removeconfigset(`*`modelObj`*`, `*`configsetObj`*`)` removes the configuration set object, *`configsetObj`*, from the SimBiology model, *`modelObj`*. The configuration set is not deleted; if you want to delete `configsetObj`, use the `delete` method.

If however, there is no MATLAB variable holding the configset, `removeconfigset(`*`modelObj`*`, '`*`NameValue`*`')` removes the configset from the model and deletes it.

Examples

1. Create a model object by importing the file `oscillator.xml` and add a configset.

   ```
   modelObj  = sbmlimport('oscillator');
   ```

```
configsetObj = addconfigset(modelObj, 'myset');
```

2. Remove the configset from `modelObj` by name or alternatively by indexing.

```
% Remove the configset with name 'myset'.
removeconfigset(modelObj, 'myset');

% Get all configset objects and remove the second.
configsetObj = getconfigset(modelObj);
removeconfigset(modelObj, configsetObj(2));
```

See Also

`addconfigset`, `getconfigset`, `setactiveconfigset`

removedose (model)

Purpose

Add dose object to model

Syntax

```
doseObj2 = removedose(modelObj, 'DoseName')
doseObj2 = removedose(modelObj, doseObj)
```

Arguments

modelObj	Model object from which you remove a dose object.
DoseName	Name of the dose object to remove from a model object. *DoseName* is the value of the dose object property `Name`.
doseObj	Dose object to remove from a model object.

Outputs

doseObj2	ScheduleDose or RepeatDose object.

Description

`doseObj2 = removedose(`*`modelObj`*`, '`*`DoseName`*`')` removes a SimBiology ScheduleDose or RepeatDose object with the name *DoseName* from a model object (*modelObj*). returns the dose object (*doseObj*), and assigns [] to the dose object property `Parent`.

You can add a removed dose object back to a model object using the method `adddose`.

`doseObj2 = removedose(`*`modelObj`*`, `*`doseObj`*`)` removes a SimBiology ScheduleDose or RepeatDose object *doseObj*.

Examples

Remove a dose object from a model object.

1. Create model and dose objects, and then add dose to model.

   ```
   modelObj = sbiomodel('mymodel');
   dose1Obj = adddose(modelObj, 'dose1');
   ```

2. Remove dose object from model object.

```
removedose(mymodel, 'dose1');
```

Get all dose objects from a model object, and then remove the second dose object.

```
AllDoseObjects = getdose(mymodel);
removedose(mymodel, AllDoseObjects(2));
```

See Also

`Model object` methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

`ScheduleDose object` and `RepeatDose object` methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

removevariant (model)

Purpose

Remove variant from model

Syntax

```
variantObj = removevariant(modelObj, 'NameValue')
variantObj = removevariant(modelObj, variantObj)
```

Arguments

modelObj	Specify the model object from which you want to remove the variant.
variantObj	Specify the variant object to return from the model object.

Description

variantObj = removevariant(*modelObj*, '*NameValue*') removes a SimBiology variant object with the name *NameValue* from the model object *modelObj* and returns the variant object to *variantObj*. The variant object `Parent` property is assigned `[]` (empty).

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant object`.

variantObj = removevariant(*modelObj*, *variantObj*) removes a SimBiology variant object (*variantObj*) and returns the variant object *variantObj*.

To view the variants stored on a model object, use the `getvariant` method. To copy a variant object to another model, use `copyobj`. To add a variant object to a SimBiology model, use the `addvariant` method.

Examples

1. Create a model containing several variants.

   ```
   modelObj = sbiomodel('mymodel');
   variantObj1 = addvariant(modelObj, 'v1');
   variantObj2 = addvariant(modelObj, 'v2');
   variantObj3 = addvariant(modelObj, 'v3');
   ```

2. Remove a variant object using its name.

```
removevariant(modelObj, 'v1');
```

3 Remove a variant object using its index number.

a Get the index number of the variant in the model.

```
vObjs = getvariant(modelObj)

SimBiology Variant Array

   Index:  Name:              Active:
   1       v2                 false
   2       v3                 false
```

b Remove the variant object.

```
removevariant(modelObj, vObjs(2));
```

See Also

addvariant, getvariant

rename (compartment, parameter, species)

Purpose

Rename object and update expressions

Syntax

`rename(Obj, 'NewNameValue')`

Arguments

Obj	Compartment, parameter, or species object.
'*NewNameValue*'	Specify the new name.

Description

`rename(Obj, 'NewNameValue')`, changes the `Name` property of the object, *Obj* to *NewNameValue* and updates any expressions in the model (such as `Rule` or `ReactionRate`) to use the new name.

If the new name is already being used by another model component, the new name will be qualified to ensure that it is unique. For example if you change a species named `A` to `K`, and a parameter with the name `K` exists, the species will be qualified as *CompartmentName*`.K` to indicate that the reference is to the species. If you are referring to an object by it's qualified name, for example *CompartmentName*`.A` and you change the species name, the reference will contain the qualified name in it's updated form, for example, *CompartmentName*`.K`

When you want to change the name of a compartment, parameter, or species object, use this method instead of `set`. The `set` method only changes the `Name` property of the object, except for species objects where the species object's `Name` property and any reaction strings which refer to species are updated to use the new name.

Examples

1. Create a model object that contains a species `A` in a rule.

   ```
   m = sbiomodel('cell');
   s = addspecies(m, 'A');
   r = addrule(m, 'A = 4');
   ```

2. Rename the species to `Y`

   ```
   rename(s, 'Y');
   ```

3 See that the rule expression is now updated.

```
r

SimBiology Rule Array

Index:    RuleType:            Rule:
1         initialAssignment    Y = 4
```

See Also

`set`

reorder (model, compartment)

Purpose

Reorder component lists

Syntax

`modelObj = reorder(Obj, NewOrder)`

Arguments

Obj	Model object or compartment. Enter a variable name.
NewOrder	Object vector in the new order. If *Obj* is a model object, *NewOrder* can be an array of compartments, events, parameters, reactions or rules objects. If *Obj* is a compartment object, *NewOrder* must be an array of species objects.

Description

`modelObj = reorder(Obj, NewOrder)` reorders the component vector *NewOrder*, to be in the order specified.

You can use this method to reorder any of the component vectors, such as events, parameters, rules, and species. The vector of components, when reordered, must contain the same objects as the original list of objects but they can be in a different order.

Examples

1 Import a model.

```
modelObj = sbmlimport('lotka');
```

2 Display the order of the reactions in the model.

```
get(modelObj.Reactions);


   SimBiology Reaction Array

   Index:    Reaction:
   1         x + y1 -> 2 y1 + x
   2         y1 + y2 -> 2 y2
   3         y2 -> z
```

3 Reverse the order of the reactions in the model.

```
reorder(modelObj, modelObj.Reactions([3 2 1]))
```

RepeatDose object

Purpose

Define drug dosing protocol

Description

A `RepeatDose` object defines a series of doses to the amount of a species during a simulation. The `TargetName` property of a dose object defines the species that receives the dose.

Each dose is the same amount, as defined by the `Amount` property, and given at equally spaced times, as defined by the `Interval` property. The `RepeatCount` property defines the number of injections in the series, excluding the initial injection. The `Rate` property defines how fast each dose is given.

To use a dose object in a simulation you must add the dose object to a model object and set the `Active` property of the dose object to true. Set the `Active` property to true if you always want the dose to be applied before simulating the model.

When there are multiple active RepeatDose objects on a model and if there are duplicate specifications for a property value, the last occurrence for the property value in the array of dose, is used during simulation. You can find out which dose is applied last by looking at the indices of the variant objects stored on the model.

See "Property Summary" on page 2-213 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

`sbiodose`	Construct dose object

Method Summary

Methods for RepeatDose objects

copyobj (any object)	Copy SimBiology object and its children
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

Properties for RepeatDose objects

`Active`	Indicate object in use during simulation
`Amount`	Amount of dose
`AmountUnits`	Dose amount units
`DurationParameterName`	arameter length of time
`Interval`	Time between doses
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`ParameterName`	arameter
`Parent`	Indicate parent object
`Rate`	of dose
`RateUnits`	Units for dose rate
`RepeatCount`	Dose repetitions
`StartTime`	Start time for initial dose time
`Tag`	Specify label for SimBiology object
`TargetName`	Species receiving dose

`TimeUnits`	Show time units for dosing and simulation
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`Model object`, `ScheduleDose object`, `sbiodose`, `sbiosimulate`

Purpose

Resample SimData object array onto new time vector

Syntax

```
newSimDataObj = resample(simDataObj)
newSimDataObj = resample(simDataObj, timevector)
newSimDataObj = resample(simDataObj, timevector, method)
```

Arguments

newSimDataObj	Resampled SimData object array.
simDataObj	SimData object array that you want to resample.
timevector	Real numeric array of time points onto which you want to resample the data.
method	Method to use during resampling. Can be one of the following:

- `'interp1q'` — Uses the MATLAB function `interp1q`.
- — To use the MATLAB function `interp1`, specify one of the following methods:
  - `'nearest'`
  - `'linear'`
  - `'spline'`
  - `'pchip'`
  - `'cubic'`
  - `'v5cubic'`
- `'zoh'` — specifies zero-order hold.

Description

`newSimDataObj = resample(simDataObj)` resamples the simulation data contained in every element of the SimData object array *simDataObj* onto a common time vector, producing a new SimData array *newSimDataObj*. By default, the common time vector is taken from the element of *simDataObj* with the earliest stopping time.

newSimDataObj = resample(*simDataObj*, *timevector*) resamples the SimData array *simDataObj* onto the time vector *timevector*. *timevector* must either be a real numeric array or the empty array []. If you use an empty array, resample uses the default time vector as described above.

newSimDataObj = resample(*simDataObj*, *timevector*, *method*) uses the interpolation method specified in *method*.

If the specified *timevector* includes time points outside the time interval encompassed by one or more SimData objects in *simDataObj*, the resampling will involve extrapolation and you will see a warning. See the help for the MATLAB function corresponding to the interpolation method in use for information on how the function performs the extrapolation.

Examples

Simulating and Resampling Data

1. The project file, radiodecay.sbproj contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject('radiodecay');
simDataObj = sbiosimulate(m1);
```

2. Resample the data.

```
newSimDataObj = resample(simDataObj, [1:5], 'linear');
```

Resampling Data for Ensemble Runs

1. The project file, radiodecay.sbproj, contains a model stored in a variable called m1. Load m1 into the MATLAB workspace.

```
sbioloadproject('radiodecay');
```

2. Change the solver to use during the simulation and perform an ensemble run.

```
csObj  = getconfigset(m1);
```

```
set(csObj, 'SolverType', 'ssa');
simDataObj = sbioensemblerun(m1, 10);
```

3 Interpolate the time steps.

```
newSimDataObj = resample(simDataObj, [1:10], 'linear');
```

4 View the time steps in the SimData object arrays.

```
newSimDataObj(1).Time
simDataObj(1).Time
```

See Also

`sbioensemblerun`, `sbioensemblestats`, `sbiosimulate`, `SimData object`

MATLAB functions `interp1`, `interp1q`

reset (root)

Purpose

Delete all model objects from root object

Syntax

```
reset(sbioroot)
```

Description

`reset(sbioroot)` deletes all SimBiology model objects contained by the root object. This call is equivalent to `sbioreset`.

The root object contains a list of model objects, available units, unit prefixes, and kinetic laws.

To add a kinetic law to the user-defined library, use the `sbioaddtolibrary` function. To add a unit to the user-defined library, use `sbiounit` followed by `sbioaddtolibrary`. To add a unit prefix to the user-defined library, use `sbiounitprefix` followed by `sbioaddtolibrary`.

Examples

1 Query `sbioroot`, which has two model objects.

```
sbioroot

   SimBiology Root Contains:

    Models:                           2
    Builtin Abstract Kinetic Laws:    3
    User Abstract Kinetic Laws:       1
    Builtin Units:                    54
    User Units:                       0
    Builtin Unit Prefixes:            13
    User Unit Prefixes:               0
```

2 Call `reset`.

```
sbioroot

   SimBiology Root Contains:

    Models:                           0
    Builtin Abstract Kinetic Laws:    3
```

```
User Abstract Kinetic Laws:       1
Builtin Units:                    54
User Units:                       0
Builtin Unit Prefixes:            13
User Unit Prefixes:               0
```

See Also

`sbioaddtolibrary`, `sbioreset`, `sbioroot`, `sbiounit`, `sbiounitprefix`

rmcontent (variant)

Purpose

Remove contents from variant object

Syntax

```
rmcontent(variantObj, contents)
rmcontent(variantObj, idx)
```

Arguments

variantObj — Specify the variant object from which you want to remove data. The `Content` property is modified to remove the new data.

contents — Specify the data you want to remove from a variant object. Contents can either be a cell array or an array of cell arrays. A valid cell array should have the form {'*Type*', '*Name*', '*PropertyName*', *PropertyValue*}, where *PropertyValue* is the new value to be applied for the *PropertyName*. Valid *Type*, *Name*, and *PropertyName* values are as follows.

'Type'	*'Name'*	*'PropertyName'*
'species'	Name of the species. If there are multiple species in the model with the same name, specify the species as [`compartmentName.speciesName`], where `compartmentName` is the name of the compartment containing the species.	'InitialAmount'
'parameter'	If the parameter scope is a model, specify the parameter name. If the parameter scope is a kinetic law, specify [`reactionName.parameterName`].	'Value'
'compartment'	Name of the compartment.	'Capacity'

idx — Specify the `ContentIndex` or indices of the data to be removed. To display the `ContentIndex`, enter the object name and press **Enter**.

Description

`rmcontent(variantObj, contents)` removes the data stored in the variable *`contents`* from the variant object (*`variantObj`*).

`rmcontent(variantObj, idx)` removes the data specified by the indices *`idx`* (also called `ContentIndex`) from the `Content` property of the variant object.

Examples

1 Create a model containing three species in one compartment.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
A = addspecies(compObj, 'A');
B = addspecies(compObj, 'B');
C = addspecies(compObj, 'C');
```

2 Add a variant object that varies the species' `InitialAmount` property.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {{'species','A', 'InitialAmount', 5}, ...
{'species', 'B', 'InitialAmount', 10}, ...
{ 'species', 'C', 'InitialAmount', 15}});% Display the variant
variantObj

SimBiology Variant - v1 (inactive)

   ContentIndex:     Type:        Name:             Property:           Value:
   1                 species      A                 InitialAmount       5
   2                 species      B                 InitialAmount       10
   3                 species      C                 InitialAmount       15
```

3 Use the `ContentIndex` number to remove a species from the `Content` property of the variant object.

```
rmcontent(variantObj, 2);
variantObj

SimBiology Variant - v1 (inactive)
```

```
   ContentIndex:     Type:        Name:            Property:          Value:
   1                 species      A                InitialAmount      5
   2                 species      C                InitialAmount      15
```

4 (Alternatively) Remove a species from the contents of the variant object using detailed reference to the species.

```
rmcontent(variantObj, {'species','A', 'InitialAmount', 5});
% Display variant object
variantObj
SimBiology Variant - v1 (inactive)

   ContentIndex:     Type:        Name:            Property:          Value:
   1                 species      C                InitialAmount      15
```

See Also

`addvariant`, `rmcontent`, `sbiovariant`

Purpose

Remove species object from reaction object products

Syntax

```
rmproduct(reactionObj, SpeciesName)
rmproduct(reactionObj, speciesObj)
```

Arguments

reactionObj	Reaction object.
SpeciesName	Name for a model object. Enter a species name or a cell array of species names.
speciesObj	Species object. Enter a species object or an array of species objects.

Description

`rmproduct(`*`reactionObj`*`, `*`SpeciesName`*`)`, in a reaction object (`reactionObj`), removes a species object with a specified name (`SpeciesName`) from the property `Products`, removes the species name from the property `Reaction`, and updates the property `Stoichiometry` to exclude the species coefficient.

`rmproduct(`*`reactionObj`*`, `*`speciesObj`*`)` removes a species object as described above using a MATLAB variable for a species object.

The species object is not removed from the parent model property `Species`. If the species object is no longer used by any reaction, you can use the function `delete` to remove it from the parent object.

If one of the species specified does not exist as a product, a warning is returned.

Examples

Example 1

This example shows how to remove a product that was previously added to a reaction. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP -> creatine + ATP + Pi');
rmproduct(reactionObj, 'Pi')
```

```
SimBiology Reaction Array

 Index:  Reaction:
  1      Phosphocreatine + ADP -> creatine + ATP
```

Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A -> B + C');
reactionObj.Reaction
 ans =
    A -> B + C

rmproduct(reactionObj, modelObj.Species(2));
reactionObj.Reaction
  ans =
    A -> C
```

See Also

rmreactant

Purpose

Remove species object from reaction object reactants

Syntax

```
rmreactant(reactionObj, SpeciesName)
rmreactant(reactionObj, speciesObj)
```

Arguments

reactionObj	Reaction object.
SpeciesName	Name for a species object. Enter a species name or a cell array of species names.
speciesObj	Species object. Enter a species object or an array of species objects.

Description

`rmreactant(`*`reactionObj`*`, `*`SpeciesName`*`)`, in a reaction object (`reactionObj`), removes a species object with a specified name (`SpeciesName`) from the property `Reactants`, removes the species name from the property `Reaction`, and updates the property `Stoichiometry` to exclude the species coefficient.

`rmreactant(`*`reactionObj`*`, `*`speciesObj`*`)` removes a species object as described above using a MATLAB variable for a species object, or a model index for a species object.

The species object is not removed from the parent model property `Species`. If the species object is no longer used by any reaction, you can use the method `delete` to remove it from the parent object.

If one of the species specified does not exist as a reactant, a warning is returned.

Examples

Example 1

This example shows how to remove a reactant that was added to a reaction by mistake. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP + Pi -> creatine + ATP');
```

```
rmreactant(reactionObj, 'Pi')

SimBiology Reaction Array

 Index:    Reaction:
  1        Phosphocreatine + ADP -> creatine + ATP
```

Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A -> B + C');

reactionObj.Reaction
ans =
    A + B -> C

rmreactant(reactionObj, modelObj.Species(1));
reactionObj.Reaction

ans =
    A -> C
```

See Also

delete, rmproduct

Purpose

Hold models, unit libraries, and abstract kinetic law libraries

Description

The SimBiology root object contains a list of the SimBiology model objects and SimBiology libraries. The components that the libraries contain are: all available units, unit prefixes, and available abstract kinetic law objects. There are two types of libraries: one contains components that are built in (`BuiltinLibrary`), and the other contains components that are user defined (`UserdefinedLibrary`).

You can retrieve SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object.

See "Property Summary" on page 2-178 for links to root object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

Constructor Summary

`sbioroot`	Return SimBiology root object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
get (any object)	Get object properties
reset (root)	Delete all model objects from root object
set (any object)	Set object properties

Property Summary

`BuiltInLibrary`	Library of built-in components
`Models`	Contain all model objects
`Type`	Display SimBiology object type
`UserDefinedLibrary`	Library of user-defined components

See Also

`AbstractKineticLaw object`, `Configset object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Rule object`, `Species object`

Purpose

Hold rule for species and parameters

Description

The SimBiology rule object represents a *rule*, which is a mathematical expression that modifies a species amount or a parameter value. For a description of the types of SimBiology rules, see `RuleType`.

See "Property Summary" on page 2-179 for links to rule property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addrule (model)	Create rule object and add to model object

Method Summary

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

`Active`	Indicate object in use during simulation
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object

Parent	Indicate parent object
Rule	Specify species and parameter interactions
RuleType	Specify type of rule for rule object
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Species object

Purpose

Define drug dosing protocol

Description

A `ScheduleDose` object defines a series of doses to the amount of a species during a simulation. The `TargetName` property defines the species that receives the dose.

Each dose can have a different amount, as defined by an amount array in the `Amount` property. Each dose can be given at specified times, as defined by a time array in the `Time` property. A rate array in the `Rate` property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

To use a dose object in a simulation, you must add the dose object to a model object and set the `Active` property of the dose object to `true`.

When there are multiple active `ScheduleDdose` objects on a model, and there are duplicate specifications for a property value, the simulation uses the last occurrence for the property value in the array of doses. You can find out which dose youapplied last by looking at the indices of the dose objects stored on the model.

See the "Property Summary" on page 2-213 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` command to list object properties and the `set` command to change their values at the command line. Use can graphically change object properties in the graphical user interface

Constructor Summary

`sbiodose`	Construct dose object

Method Summary

Methods for variant objects

copyobj (any object)	Copy SimBiology object and its children
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

Properties for variant objects

`Active`	Indicate object in use during simulation
`Amount`	Amount of dose
`AmountUnits`	Dose amount units
`DurationParameterName`	arameter length of time
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`ParameterName`	arameter
`Parent`	Indicate parent object
`Rate`	of dose
`RateUnits`	Units for dose rate
`Tag`	Specify label for SimBiology object
`TargetName`	Species receiving dose
`Time`	Simulation time steps or schedule dose times
`TimeUnits`	Show time units for dosing and simulation
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`Model object`, `RepeatDose object`, `sbiodose`, `sbiosimulate`

Purpose

Select data from SimData object

Syntax

```
[t,x,names] = select(simDataObj, Query)
[Out] = select(simDataObj, Query, 'Format', 'FormatValue')
```

Arguments

Output Arguments

t	An n-by-1 vector of time points.
x	An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
names	An m-by-1 cell array of names.
Out	Data returned in the format specified in *'FormatValue'*, shown in "Input Arguments" on page 2-183. Depending on the specified *'FormatValue'*, *Out* contains one of the following: • Structure array • SimData object • Time series object • Combined time series object from an array of SimData objects

Input Arguments

simDataObj	SimData object array. Enter a variable name for a SimData object.

Query	A cell array of arguments consisting of some combination of property name/property value pairs and/or `'Where'` clauses. For a more complete description of the query syntax, including `'Where'` clauses and their supported condition types, see `sbioselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a SimData object in a query. These include `'Type'`, `'Name'`, `'Units'`, `'Compartment'` (species only), or `'Reaction'` (parameter only).
FormatValue	Choose a format from the following table.

FormatValue	**Description**
`'num'`	Specifies the format that lets you return data in numeric arrays. This is the default when `select` is called with multiple output arguments.
`'nummetadata'`	Specifies the format that lets you return a cell array of metadata structures in *metadata* instead of names. The elements of *metadata* label the columns of *x*.
`'numqualnames'`	Specifies the format that lets you return qualified names in *names* to resolve ambiguities.
`'struct'`	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
`'simdata'`	Specifies the format that lets you return data in a new SimData object. This is the default format when `select` is called with zero or one output argument.
`'ts'`	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in `simDataObj`.
`'tslumped'`	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

Description

`[t,x,names] = select(simDataObj, Query)` returns simulation time and state data from the SimData object (`simDataObj`) that matches the query argument *`Query`*.

In a SimData object *`simDataObj`*, the columns of the data matrix `simDataObj.Data` are labeled by the cell array of metadata structures given by `simDataObj.DataInfo`. The `select` method enables you to pick out columns of the data matrix based on their metadata labels. For example, to extract data for all parameters logged in a SimData object *`simDataObj`*, use the syntax `[t, x, names] = select (simDataObj, {'Type', 'parameter'}]`.

`[Out] = select(simDataObj, Query, 'Format', 'FormatValue')` returns the data in the specified format. Valid formats are listed in "Input Arguments" on page 2-183.

Examples

This example shows how to extract data of interest from your simulation data with the `select` method.

1 The project file `radiodecay.sbproj` contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

```
sbioloadproject gprotein_norules m1
```

2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj  = getconfigset(m1);
set(csObj, 'SolverType', 'ssa');
simDataObj = sbioensemblerun(m1, 10);
```

3 Select all species data logged in the SimData array `sdarray`.

```
[t x n] = select(simDataObj, {'Type','species'});
```

4 Select data for the parameters with name `'Kd'` and return the results in a new SimData object array.

```
newsd = select(simDataObj, {'Type','parameter','name', 'Kd'});
```

5 This selects all data from simDataObj with a name that matches the pattern 'G' and returns time series objects.

```
ts = select(simDataObj, {'Where','Name','regexp','G'}, ...
                    'Format','ts');
```

See Also

getdata, sbioselect, sbiosimulate, selectbyname, Simdata object

Purpose

Select data by name from SimData object array

Syntax

```
[t,x,n] = selectbyname(simDataObj, 'NameValue')
Out = selectbyname(simDataObj, NameValue, 'Format', Format)
```

Arguments

Output Arguments

t	An `n-by-1` vector of time points.
x	An `n-by-m` data array. *t* and *names* label the rows and columns of *x* respectively.
n	An `m-by-1` cell array of names.
Out	Data returned in the format as specified in '*FormatValue*', shown in "Input Arguments" on page 2-187. Depending on the specified '*FormatValue*', *Out* contains one of the following: • Structure array • SimData object • Time series object • Combined time series object from an array of SimData objects

Input Arguments

simDataObj	SimData object array. Enter a variable name for a SimData object.
NameValue	Names of the states for which you want to select data from *simDataObj*. Must be either a string or a cell array of strings.

Query	A cell array of arguments consisting of some combination of property name/property value pairs and/or `'Where'` clauses. For a more complete description of the query syntax, including `'Where'` clauses and their supported condition types, see `sbioselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a SimData object. These include `'Type'`, `'Name'`, `'Units'`, `'Compartment'` (species only), or `'Reaction'` (parameter only).
FormatValue	Choose a format from the following table.

FormatValue	Description
`'num'`	Specifies the format that lets you return data in numeric arrays. This is the default when `selectbyname` is called with multiple output arguments.
`'nummetadata'`	Specifies the format that lets you return a cell array of metadata structures in *metadata* instead of names. The elements of *metadata* label the columns of *x*.
`'numqualnames'`	Specifies the format that lets you return qualified names in *names* to resolve ambiguities.
`'struct'`	Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.
`'simdata'`	Specifies the format that lets you return data in a new SimData object. This is the default format when `selectbyname` is called with zero or one output argument.

FormatValue	Description
'ts'	Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in simDataObj.
'tslumped'	Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.

Description

The selectbyname method allows you to select data from a SimData object array by name. [*t*,*x*,*n*] = selectbyname(*simDataObj*, '*NameValue*') returns time and state data from the SimData object *simDataObj* for states with names '*NameValue*'.

In a SimData object *simDataObj*, the names labeling the columns of the data matrix *simDataObj*.Data are given by *simDataObj*.DataNames. A name specified in '*NameValue*' can match more than one data column, for example, when *simDataObj* contains data for a species and parameter both named 'k'. To resolve ambiguities, use qualified names in '*NameValue*', such as '*CompartmentName.SpeciesName*' or '*ReactionName.ParameterName*'. selectbyname returns qualified names in the output argument *names* when there are ambiguities.

Out = selectbyname(*simDataObj*, *NameValue*, 'Format', *Format*) returns the data in the specified format. Valid formats are listed in "Input Arguments" on page 2-187.

Examples

```
% Get data for the species 'glucose' from the simdata array sdarray.
[t x n] = selectbyname(sdarray,'glucose');

% Get data for multiple states and return the results in a struct array.
s = selectbyname(sdarray,{'RexGFP';'nuc.GFP';'cytosol.GFP'},...
                          'Format','struct');
```

See Also

getdata, sbioselect, sbiosimulate

set (any object)

Purpose

Set object properties

Syntax

```
set(Obj, 'PropertyName', PropertyValue)
set(Obj, 'PropertyName1', PropertyValue1, 'PropertyName2',
   PropertyValue2...)
```

Arguments

Obj	Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, `PKCompartment`, `PKData`, `PKModelDesign` `PKModelMap`, reaction, rule, `SimData`, species, or variant object.
'*PropertyName*'	Name of the property to set.
PropertyValue	Specify the value to set. Property values depend on the property being set. See the reference page for an object property for values that can be specified.

Description

`set(`*`Obj`*`, '`*`PropertyName`*`', `*`PropertyValue`*`)` sets the property '*PropertyName*' of the object *Obj*, to *PropertyValue*.

`set(`*`Obj`*`, '`*`PropertyName1`*`', `*`PropertyValue1`*`, '`*`PropertyName2`*`',` *`PropertyValue2`*`...)` sets the properties '*PropertyName1*' and '*PropertyName2*' to *PropertyValue1* and *PropertyValue2* respectively, and so on in sequence. You can specify multiple *PropertyName*, *PropertyValue* pairs.

When you want to change the name of a compartment, parameter, or species object, use the `rename` method instead of `set`. The `rename` method allows you to change the name and update the expressions in which these components are used.

Examples

1. Create a model object.

   ```
   modelObj = sbiomodel ('my_model');
   ```

2. Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

3. Set the `ConstantValue` property of the parameter object to `false` and verify.

   MATLAB returns `1` for `true` and `0` for `false`.

   ```
   set (parameterObj, 'ConstantValue', false);
   get(parameterObj, 'ConstantValue')
   ```

   MATLAB returns

   ```
   ans =

        0
   ```

See Also

`get` , `rename`, `setactiveconfigset`

setactiveconfigset (model)

Purpose

Set active configuration set for model object

Syntax

```
configsetObj = setactiveconfigset(modelObj, 'NameValue')
configsetObj2 = setactiveconfigset(modelObj, configsetObj1)
```

Description

configsetObj = setactiveconfigset(*modelObj*, '*NameValue*') sets the configuration set *NameValue* to be the active configuration set for the model *modelObj* and returns to *configsetObj*.

configsetObj2 = setactiveconfigset(*modelObj*, *configsetObj1*) sets the configset *configsetObj1* to be the active configset for *modelObj* and returns to *configsetObj2*. Any change in one of these two configset objects *configsetObj1* and *configsetObj2* is reflected in the other. To copy over a configset object from one model object to another, use the copyobj method.

The active configuration set contains the settings that are be used during a simulation. A default configuration set is attached to any new model.

Examples

1 Create a model object by importing the oscillator.xml file, and add a Configset object to the model.

```
modelObj  = sbmlimport('oscillator');
configsetObj = addconfigset(modelObj, 'myset');
```

2 Configure the simulation stop criteria by setting the StopTime, MaximumNumberOfLogs, and MaximumWallClock properties of the Configset object. Set the stop criteria to a simulation time of 3000 seconds, 50 logs, or a wall clock time of 10 seconds, whichever comes first.

```
set(configsetObj, 'StopTime', 3000, 'MaximumNumberOfLogs', 50,...
    'MaximumWallClock', 10)
get(configsetObj)

                    Active: 0
            CompileOptions: [1x1 SimBiology.CompileOptions]
```

```
                         Name: 'myset'
                        Notes: ''
               RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
   SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisO
                SolverOptions: [1x1 SimBiology.ODESolverOptions]
                   SolverType: 'ode15s'
                     StopTime: 3000
          MaximumNumberOfLogs: 50
             MaximumWallClock: 10
                    TimeUnits: 'second'
                         Type: 'configset'
```

3 Set the new `Configset` object to be active, simulate the model using the new `Configset` object, and plot the result.

```
 setactiveconfigset(modelObj, configsetObj);
[t,x] = sbiosimulate(modelObj);
plot (t,x)
```

See Also

`addconfigset`, `getconfigset`, `removeconfigset`

setparameter (kineticlaw)

Purpose

Specify specific parameters in kinetic law object

Syntax

```
setparameter(kineticlawObj, 'ParameterVariablesValue',
   'ParameterVariableNamesValue')
```

Arguments

ParameterVariableValue	Specify the value of the parameter variable in the kinetic law object.
ParameterVariableNamesValue	Specify the parameter name with which to configure the parameter variable in the kinetic law object. Determines parameters in the `ReactionRate` equation.

Description

Configure `ParameterVariableNames` in the kinetic law object.

`setparameter(`*`kineticlawObj`*`, 'ParameterVariablesValue', 'ParameterVariableNamesValue')` configures the `ParameterVariableNames` property of the kinetic law object (`kineticlawObj`). `ParameterVariableValue` corresponds to one of the strings in `kineticlawObj ParameterVariables` property. The corresponding element in the `kineticlawObjParameterVariableNames` property is configured to `ParameterVariableNamesValue`. For example, if `ParameterVariables` is `{'Vm', 'Km'}` and `ParameterVariablesValue` is specified as `Vm`, then the first element of the `ParameterVariableNames` cell array is configured to `ParameterVariableNamesValue`.

Examples

Create a model, add a reaction, and then assign the `ParameterVariableNames` for the reaction rate equation.

1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
```

```
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2. Create a kinetic law object for the reaction object of the type `'Henri-Michaelis-Menten'`.

   ```
   kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
   ```

   `reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3. The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) that should be set. To set these variables:

   ```
   setparameter(kineticlawObj,'Vm', 'Va');
   setparameter(kineticlawObj,'Km', 'Ka');
   ```

4. Verify that the parameter variables are correct.

   ```
   get (kineticlawObj, 'ParameterVariableNames')
   ```

   MATLAB returns:

   ```
   ans =

       'Va'    'Ka'
   ```

See Also

`addparameter`, `getspecies`, `setspecies`

setspecies (kineticlaw)

Purpose

Specify species in kinetic law object

Syntax

```
setspecies(kineticlawObj, 'SpeciesVariablesValue',
   'SpeciesVariableNamesValue')
```

Arguments

SpeciesVariablesValue	Specify the species variable in the kinetic law object.
SpeciesVariableNamesValue	Specify the species name with which to configure the species variable in the kinetic law object. Determines the species in the ReactionRate equation.

Description

setspecies configures the kinetic law object SpeciesVariableNames property.

setspecies(kineticlawObj, 'SpeciesVariablesValue', 'SpeciesVariableNamesValue') configures the SpeciesVariableNames property of the kinetic law object, kineticlawObj. SpeciesVariablesValue corresponds to one of the strings in the SpeciesVariables property of kineticlawObj. The corresponding element in kineticlawObj SpeciesVariableNames property is configured to SpeciesVariableNamesValue.

For example, if SpeciesVariables are {'S', 'S1'} and SpeciesVariablesValue is specified as S1, the first element of the SpeciesVariableNames cell array is configured to SpeciesVariableNamesValue.

Examples

Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

`reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has one species variable (`S`) that should be set. To set this variable:

```
setspecies(kineticlawObj,'S', 'a');
```

4 Verify that the species variable is correct.

```
get (kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =

'a'
```

See Also

`addparameter`, `getspecies`, `setparameter`

SimBiology.export.Model.simulate

Purpose

Simulate exported SimBiology model

Syntax

```
[t,x,names] = simulate(model)
[t,x,names] = simulate(model,initialValues)
[t,x,names] = simulate(model,initialValues,doses)
simDataObj = simulate( ___ )
```

Description

`[t,x,names] = simulate(model)` simulates a model, using the default initial values specified by `model.InitialValues` (which are always equal to the `InitialValue` property on the corresponding `ValueInfo` object). `simulate` returns:

- Time samples, `t`.
- Variation in the quantity of states over time, `x`.
- Names for the rows and columns of `x`, `names`.

You can set additional simulation options using the property `SimBiology.export.Model.SimulationOptions`.

`[t,x,names] = simulate(model,initialValues)` simulates a model, using the values specified in `initialValues` as the initial values of the simulation.

`[t,x,names] = simulate(model,initialValues,doses)` simulates the model, using the specified initial values and doses.

`simDataObj = simulate( ___ )` returns a `SimData object` that contains time and state data, as well as metadata, such as the types and names for the reported states. You can access the time, data, and names stores in `simDataObj` using the properties `simDataObj.Time`, `simDataObj.Data`, and `simDataObj.DataNames`, respectively. You can use any of the previous input arguments.

Input Arguments

model

`SimBiology.export.Model` object.

initialValues

Vector of values for `simulate` to use as the initial values of the simulation. `initialValues` must have the same number of elements as `model.InitialValues`.

Default: Values specified in `model.InitialValues`.

doses

Vector of dose objects specifying the doses used for simulation. The input dose objects must be a subset of the doses in the exported model, as returned by `getdose`.

Default: All dose objects in the exported model.

Output Arguments

t

n-by-1 vector of time samples from the simulation, where n is the number of time samples.

x

n-by-m matrix of simulation data, where n is the number of time samples and m is the number of states logged during the simulation. Each column of `x` describes the variation in the quantity of a state over time.

names

m-by-1 cell array of strings with names labeling the rows and columns of `x`, respectively.

simDataObj

`SimData object` containing simulation time and state data, as well as metadata, such as the types and names for the reported states.

Examples

Simulate an Exported SimBiology Model

Load a sample SimBiology model object, and select the species `y1` and `y2` for simulation.

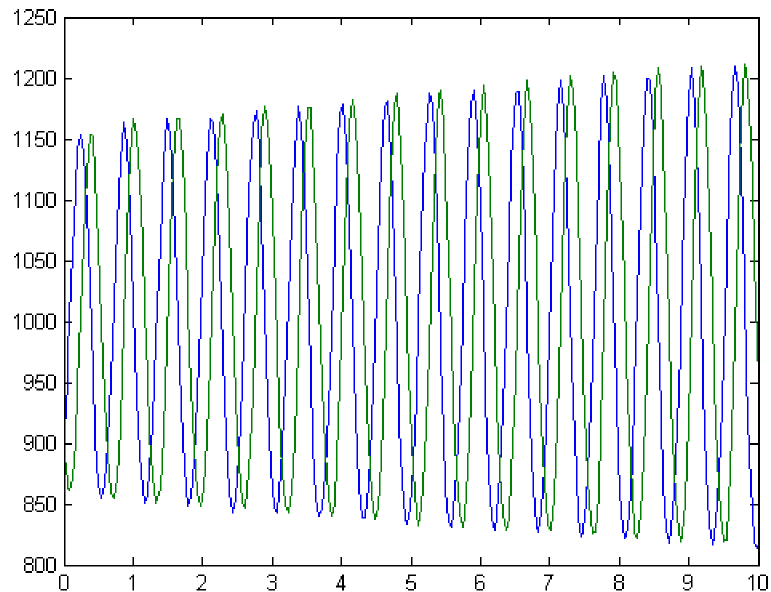
```
modelObj = sbmlimport('lotka');
modelObj.getconfigset.RuntimeOptions.StatesToLog = ...
       sbioselect(modelObj,'Name',{'y1','y2'});
```

Export the model object.

```
em = export(modelObj);
```

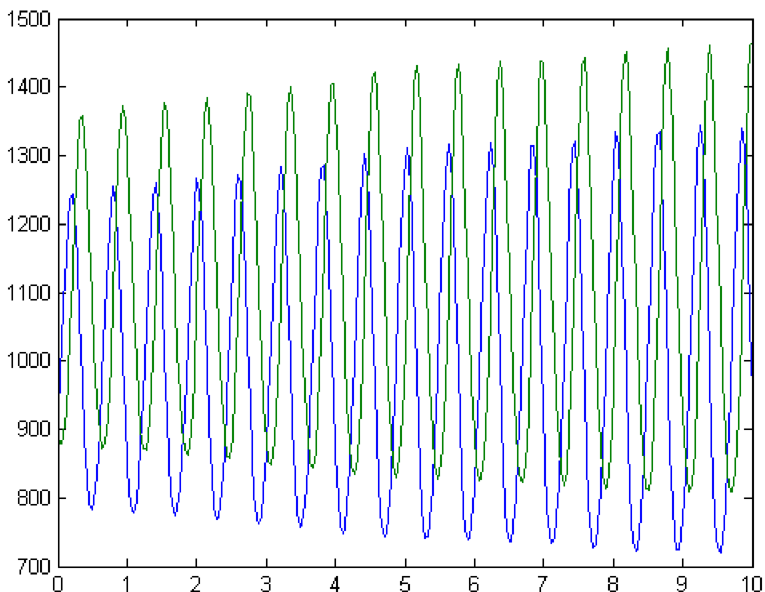
Simulate the exported model.

```
[t,y] = simulate(em);

figure()
plot(t,y)
```



Modify the initial conditions, and simulate again.

```
xIndex = em.getIndex('x');
em.InitialValues(xIndex) = em.InitialValues(xIndex)*1.1;
[t,y] = simulate(em);

figure()
plot(t,y)
```



See Also

SimBiology.export.Model | SimBiology.export.Model.getdose | SimBiology.export.ValueInfo | SimBiology.export.SimulationOptions | SimData object

Related Examples

- "PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics"
- "Deploy a SimBiology Model"

SimData object

Purpose

Simulation data storage

Description

The SimBiology `SimData` object contains simulation data. The output from the `sbiosimulate` function, is stored in the `SimData` object which holds time and state data as well as metadata, such as the types and names for the logged states or the configuration set used during simulation.

You can also store data from multiple simulation runs as an array of SimData objects. Thus, the output of `sbioensemblerun` is an array of SimData objects. You can use any SimData method on an array of SimData objects.

You can access the time, data, and metadata stored in the `SimData` object through the properties in "Property Summary" on page 2-203. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line.

Methods you can use to query the SimData object are listed in "Method Summary" on page 2-202.

Constructor Summary

`sbioensemblerun`	Multiple stochastic ensemble runs of SimBiology model
`sbiosimulate`	Simulate model object

Method Summary

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
getdata (SimData)	Get data from `SimData` object array

getsensmatrix (SimData)	Get 3-D sensitivity matrix from SimData array
resample (SimData)	Resample SimData object array onto new time vector
select (SimData)	Select data from SimData object
selectbyname (SimData)	Select data by name from SimData object array
set (any object)	Set object properties

Property Summary

`Data`	Store simulation data
`DataCount`	Numbers of species, parameters, sensitivities
`DataInfo`	Metadata labels for simulation data
`DataNames`	Show names in SimData object
`ModelName`	Name of model simulated
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`RunInfo`	Information about simulation
`Time`	Simulation time steps or schedule dose times
`TimeUnits`	Show time units for dosing and simulation
`UserData`	Specify data to associate with object

Examples

Return simulation results to a SimData object and plot the results.

```
sbioloadproject('radiodecay', 'm1');
simDataObj = sbiosimulate(m1);
sbioplot(simDataObj)
```

Get simulation data at specific time points using the `resample` method.

```
% Load 'radiodecay' and set the initial amount of species 'x'.
sbioloadproject('radiodecay', 'm1');
x = sbioselect(m1, 'Type', 'species', 'Name', 'x');
x.InitialAmount = 100;
% Change the solver type to a stochastic solver.
cs = m1.getconfigset;
cs.SolverType = 'ssa';
% Simulate the model.
simDataObj = sbiosimulate(m1);
% This result could be misinterpreted as containing fractional molecules.
sbioplot(simDataObj);
title('Simulation Results Before Resampling');
%Resample the data using the zero-order hold method to obtain the correct
%number of molecules at intermediate time steps.
newsimDataObj = resample(simDataObj, linspace(0, 10, 1e4), 'zoh');
sbioplot(newsimDataObj);
title('Simulation Results After Resampling');
```

Initialize a simulation using results from a previous simulation.

```
% Load 'radiodecay'.
sbioloadproject('radiodecay', 'm1');
m1.Species
simDataObj = sbiosimulate(m1);
% Use the Data property to get the states at the final time point.
% Data is an m x n array, where m is the number of time steps in
% the simulation and n is the number of quantities logged.
finaldata = simDataObj.Data(end,:);
% Use the DataInfo property to get names of states.
info = simDataObj.DataInfo;
```

```
% Loop through the states (species) and set their initial amounts.
numSpecies = length(info);
for c = 1:numSpecies
compObj = sbioselect(m1,'type','compartment','Name',info{c}.Compartmen
speciesObj = sbioselect(compObj,'type','species','Name',info{c}.Name);
speciesObj.InitialAmount = finaldata(c);
end
% Verify species initial amounts.
m1.Species
```

See Also

Model object, Parameter object, Reaction object, Root object, Species object

Species object

Purpose

Options for compartment species

Description

The SimBiology species object represents a *species*, which is a chemical or entity that participates in reactions, for example, `DNA`, `ATP`, `Pi`, `creatine`, `G-Protein`, or `Mitogen-Activated Protein Kinase (MAPK)`. Species amounts can vary or remain constant during a simulation.

To add species that participate in reactions, add the reaction to the model. The process of adding the reaction to the model creates a compartment object (*`unnamed`*) and the necessary species objects.

Alternatively, create and add a species object to a compartment object, using the `addspecies` method at the command line.

When you use the SimBiology desktop to create a new model, it adds an empty compartment (*`unnamed`*), to which you can add species.

See "Property Summary" on page 2-207 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

addspecies (model, compartment)	Create species object and add to compartment object within model object

Method Summary

Methods for species objects

copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object

get (any object)	Get object properties
rename (compartment, parameter, species)	Rename object and update expressions
set (any object)	Set object properties

Property Summary

Properties for species objects

`BoundaryCondition`	Indicate species boundary condition
`ConstantAmount`	Specify variable or constant species amount
`InitialAmount`	Species initial amount
`InitialAmountUnits`	Species initial amount units
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`Compartment object`, `Configset object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`

Unit object

Purpose

Hold information about user-defined unit

Description

The SimBiology unit object holds information about user-defined units. To create a unit, create the unit object and add the unit to the library using the `sbioaddtolibrary` function.

Use the unit object property `Composition` to specify the composition of your units. See "Property Summary" on page 2-208 for links to unit object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change unit object properties in the SimBiology desktop, using the Library Explorer. For more information, see "What Are SimBiology Libraries?".

Constructor Summary

`sbiounit`	Create user-defined unit

Method Summary

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

`Composition`	Unit composition
`Multiplier`	Relationship between defined unit and base unit
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object

`Offset`	Unit composition modifier
`Parent`	Indicate parent object
`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`AbstractKineticLaw object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`, `Species object`, `UnitPrefix object`

UnitPrefix object

Purpose

Hold information about user-defined unit prefix

Description

The SimBiology unit prefix object holds information about user-defined unit prefixes. To create a unit prefix, create the unit prefix object and add the unit prefix to the library using the `sbioaddtolibrary` function.

Use the unit prefix object property `Exponent`, to specify the exponent of your unit prefix. See "Property Summary" on page 2-210 for links to unit prefix object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change unit prefix object properties in the SimBiology desktop, using the Library Explorer. For more information, see "What Are SimBiology Libraries?".

Constructor Summary

`sbiounitprefix`	Create user-defined unit prefix

Method Summary

delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
set (any object)	Set object properties

Property Summary

`Exponent`	Exponent value of unit prefix
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object
`Parent`	Indicate parent object

`Tag`	Specify label for SimBiology object
`Type`	Display SimBiology object type
`UserData`	Specify data to associate with object

See Also

`AbstractKineticLaw object`, `KineticLaw object`, `Model object`, `Parameter object`, `Reaction object`, `Root object`, `Rule object`, `Species object`, `Unit object`

Variant object

Purpose

Store alternate component values

Description

The SimBiology variant object stores the names and values of model components and allows you to use the values stored in a variant object as the alternate value to be applied during a simulation. You can store values for species `InitialAmount`, parameter `Value`, and compartment `Capacity` in a variant object. Simulating using a variant does not alter the model component values. The values specified in the variant temporarily apply during simulation.

Using one or more variant objects associated with a model allows you to evaluate model behavior during simulation, with different values for the various model components without having to search and replace these values, or having to create additional models with these values. If you determine that the values in a variant object accurately define your model, you can permanently replace the values in your model with the values stored in the variant object, using the `commit` method.

To use a variant in a simulation you must add the variant object to the model object and set the `Active` property of the variant to true. Set the `Active` property to true if you always want the variant to be applied before simulating the model. You can also enter the variant object as an argument to `sbiosimulate`; this applies the variant only for the current simulation and supersedes any active variant objects on the model.

When there are multiple active variant objects on a model, if there are duplicate specifications for a property's value, the last occurrence for the property value in the array of variants, is used during simulation. You can find out which variant is applied last by looking at the indices of the variant objects stored on the model. Similarly, in the `Content` property, if there are duplicate specifications for a property's value, the last occurrence for the property in the `Content` property, is used during simulation.

Use the `addcontent` method to append contents to a variant object.

See "Property Summary" on page 2-213 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change

their values at the command line. You can graphically change object properties in the graphical user interface.

Constructor Summary

`sbiovariant`	Construct variant object

Method Summary

Methods for variant objects

addcontent (variant)	Append content to variant object
commit (variant)	Commit variant contents to model
copyobj (any object)	Copy SimBiology object and its children
delete (any object)	Delete SimBiology object
display (any object)	Display summary of SimBiology object
get (any object)	Get object properties
rmcontent (variant)	Remove contents from variant object
set (any object)	Set object properties
verify (model, variant)	Validate and verify SimBiology model

Property Summary

Properties for variant objects

`Active`	Indicate object in use during simulation
`Content`	Contents of variant object
`Name`	Specify name of object
`Notes`	HTML text describing SimBiology object

Parent	Indicate parent object
Tag	Specify label for SimBiology object
Type	Display SimBiology object type
UserData	Specify data to associate with object

See Also

Compartment object, Configset object, Model object, Parameter object, Species object

sbiosimulate

Purpose

Validate and verify SimBiology model

Syntax

```
verify(modelObj)
verify(modelObj, configsetObj)
verify(modelObj, variantObj)
verify(modelObj, configsetObj, variantObj)
```

Description

`verify(`*`modelObj`*`)` performs checks on a model object (*`modelObj`*) to verify that you can simulate the model. This method generates stacked errors and warnings if any problems are found. To see the entire list of errors and warnings, use `sbiolasterror` and `sbiolastwarning`. The `verify` method uses the active configuration set for verification.

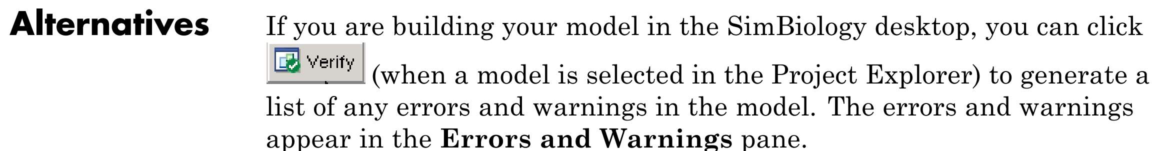
`verify(`*`modelObj`*`, `*`configsetObj`*`)` performs checks on the specified configuration set object (*`configsetObj`*) in conjunction with the model object (*`modelObj`*) to verify that you can simulate the model.

`verify(`*`modelObj`*`, `*`variantObj`*`)` performs checks on the variant object (*`variantObj`*) in conjunction with the model object (*`modelObj`*) to verify that you can simulate the model. The model object is required for the verification of the variant object.

`verify(`*`modelObj`*`, `*`configsetObj`*`, `*`variantObj`*`)` performs checks on the configuration set object *`configsetObj`*, and the variant object *`variantObj`* in conjunction with the model object (*`modelObj`*) to verify that you can simulate the model.

Alternatives

If you are building your model in the SimBiology desktop, you can click



(when a model is selected in the Project Explorer) to generate a list of any errors and warnings in the model. The errors and warnings appear in the **Errors and Warnings** pane.

Examples

```
modelObj = sbmlimport('radiodecay.xml');
verify(modelObj);
```

See Also

`sbiolasterror`, `sbiolastwarning`

verify (covmodel)

Purpose

Check covariate model for errors

Syntax

`verify(CovModelObj)`

Description

`verify(CovModelObj)` verifies that the following are true about the `Expression` property of *`CovModelObj`*, a `CovariateModel` object:

- The expression strings are valid MATLAB code.
- Each expression string is linear with a transformation.
- There is exactly one expression string for each parameter.
- In each expression string, a covariate is used in at most one term.
- In each expression string, there is at most one random effect (`eta`)
- Fixed effect (`theta`) and random effect (`eta`) names are unique within and across expression strings. That is, each covariate has its own fixed effect.

If the previous requirements are true, then `verify` returns nothing.

See Also

`construct` | `CovariateModel` | `Expression` | `PKModelDesign object`

How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- "Specifying a Covariate Model"

Properties — Alphabetical List

AbsoluteTolerance property

Purpose

Absolute error tolerance applied to state value during simulation

Description

AbsoluteTolerance is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

The software uses AbsoluteTolerance to determine the largest allowable absolute error at any step in a simulation. How the software uses AbsoluteTolerance to determine this error depends on whether the AbsoluteToleranceScaling property is enabled.

When AbsoluteToleranceScaling Is Enabled

When the AbsoluteToleranceScaling property is enabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values whose size is of order 1. For all other state values, it scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

When AbsoluteToleranceScaling Is Disabled

When the AbsoluteToleranceScaling property is disabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values, for example, amounts for all species.

Algorithm

At each simulation step, the solver estimates the local error e_i in the ith state vector y. Simulation converges at that time step if e_i satisfies the following equation:

$$|e_i| \ \mathtt{max(RelativeTolerance*}|y_i|\mathtt{,AbsoluteTolerance)}$$

Thus at higher state values, convergence is determined by RelativeTolerance. As the state values approach zero, convergence is controlled by AbsoluteTolerance. The choice of values for RelativeTolerance and AbsoluteTolerance varies depending on the problem. The default values should work for first trials of the simulation. However if you want to optimize the solution, consider that there is a tradeoff between speed and accuracy:

- If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy.
- If the results seem inaccurate, you can decrease the tolerance values, but this will slow down the solver.
- If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`
Data values	Positive scalar. Default is `1e-6`.
Access	Read/write

Examples

This example shows how to change `AbsoluteTolerance`.

1. Retrieve the `configset` object from the `modelObj`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
```

2. Change the `AbsoluteTolerance` to `1e-8`.

```
set(configsetObj.SolverOptions, 'AbsoluteTolerance', 1.0e-8);
get(configsetObj.SolverOptions, 'AbsoluteTolerance')

ans =

   1.0000e-008
```

See Also

`AbsoluteToleranceScaling`, `AbsoluteToleranceStepSize`, `RelativeTolerance`

AbsoluteToleranceScaling property

Purpose

Control scaling of absolute error tolerance during simulation

Description

`AbsoluteToleranceScaling` is a property of a `SolverOptions` object, which is a property of a `Configset` object. It is available for the ode solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`).

`AbsoluteToleranceScaling` controls how the software determines the largest allowable absolute error at any step in a simulation.

When AbsoluteToleranceScaling Is Enabled

When the `AbsoluteToleranceScaling` property is enabled, the software uses the `AbsoluteTolerance` value as the absolute error tolerance for all state values whose size is of order 1. For all other state values, it scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of `AbsoluteTolerance`.

When AbsoluteToleranceScaling Is Disabled

When the `AbsoluteToleranceScaling` property is disabled, the software uses the `AbsoluteTolerance` value as the absolute error tolerance for all state values, for example, amounts for all species.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`logical`
Data values	`1`, `0`, `true`, or `false`. Default is `true`.
Access	Read/write

See Also

`AbsoluteTolerance`, `AbsoluteToleranceStepSize`, `RelativeTolerance`

Purpose

Initial guess for time step size for scaling of absolute error tolerance

Description

AbsoluteToleranceStepSize is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

When the AbsoluteToleranceScaling property is enabled, you can set the AbsoluteToleranceStepSize property to specify the initial guess for time step size for scaling. Then, for all state values whose size is of order 1, the software scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

Tip Use AbsoluteToleranceStepSize when a simulation is unsuccessful and generates numerically unstable solutions, and other corrective actions such as checking the model's kinetics do not work. You might encounter unstable solutions if you have very stiff systems in which state values change rapidly at the beginning of a simulation. To solve this, iteratively decrease AbsoluteToleranceStepSize and simulate to find the optimal setting. As a starting point, try setting this property to AbsoluteTolerance * StopTime * 0.1.

Characteristics

Applies to	Object: SolverOptions
Data type	double
Data values	Scalar in units specified by TimeUnits property. Default is [].
Access	Read/write

See Also

AbsoluteTolerance, AbsoluteToleranceScaling, RelativeTolerance

Active property

Purpose

Indicate object in use during simulation

Description

The `Active` property indicates whether a simulation is using a SimBiology object. A SimBiology model is organized into a hierarchical group of objects. Use the `Active` property to include or exclude objects during a simulation.

- **Configuration set** — For the `configset` object, use the method `setactiveconfigset` to set the object `Active` property to `true`.
- **Event, Reaction, or Rule** — When an event, a reaction, or rule object `Active` property is set to `false`, the simulation does not include the event, reaction, or rule. This is a convenient way to test a model with and without a reaction or rule.
- **Variant** — Set the `Active` property to `true` if you always want the variant to be applied before simulating the model. You can also pass the variant object as an argument to `sbiosimulate`; this applies the variant only for the current simulation. For more information on using the `Active` property for variants, see `Variant object`.

Characteristics

Applies to	Objects: configset, event, reaction, RepeatDose, rule, ScheduleDose, variant
Data type	`boolean`
Data values	`true` or `false`. The default value for events, reactions, and rules is `true`. For the `configset` object, default is `true`. For added configset object, the default is `false`. For variants, the default is `false`.
Access	Read/write

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2. Add a reaction object and verify that the `Active` property setting is `'true'` or `1`.

   ```
   reactionObj = addreaction (modelObj, 'a + b -> c + d');
   get (reactionObj, 'Active')
   ```

   MATLAB returns:

   ```
   ans =

   1
   ```

3. Set the `Active` property to `'false'` and verify.

   ```
   set (reactionObj, 'Active', false);
   get (reactionObj, 'Active')
   ```

   MATLAB returns:

   ```
   ans =

   0
   ```

See Also

`addconfigset`, `addreaction`, `addrule`, `Event object`, `Reaction object`,`RepeatDose object`, `Rule object` , `ScheduleDose object`, `Variant object`,

Amount property

Purpose

Amount of dose

Description

`Amount` is a property of a RepeatDose or ScheduleDose object. It defines an increase in the amount of a SimBiology species that receives a dose.

A RepeatDose object defines a series of doses. Each dose is the same amount, as defined by the `Amount` property, and given at equally spaced times, as defined by the `Interval` property. The number of injections in the series, excluding the initial injection, is defined by the `RepeatCount` property, and the `Rate` property defines how fast each dose is given.

A ScheduleDose object defines a series of doses. Each dose can have a different amount, as defined by an amount array in the `Amount` property, and given at specified times, as defined by a time array in the `Time` property. A rate array in the `Rate` property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

Characteristics

Applies to	Object: RepeatDose, ScheduleDose
Data type	double (RepeatDose) or double array (ScheduleDose)
Data values	Nonnegative value. Default is `0` (RepeatDose) or `[]` (ScheduleDose)
Access	Read/write

See Also

`ScheduleDose object` and `RepeatDose object`

Purpose

Dose amount units

Description

`AmountUnits` is a property of a `RepeatDose` or `ScheduleDose` object. This property defines units for the `Amount` property.

If the `TargetName` property defines a species, then `AmountUnits` for a dose must be a chemical amount (for example, milligram, mole, or molecule), not a concentration. To get a list of the defined units in the library, use the `sbioshowunits` function. To add a user-defined unit to the list, see `sbioaddtolibrary`.

Characteristics

Applies to	Objects: `RepeatDose`, `ScheduleDose`
Data type	`string`
Data values	Units from library with dimensions of amount. Default = ” (empty)
Access	Read/write

See Also

`ScheduleDose object` and `RepeatDose object`

Annotation property

Purpose

Store link to URL or file

Note The `Annotation` property will be removed in a future release. Use the `Notes` property instead.

Description

The `Annotation` property stores the URL or file name linking to information about a model.

Characteristics

Applies to	SimBiology objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, or unit
Data type	`char string`, `URL`
Data values	Character string with a directory path and filename or a URL
Access	Read/write

Examples

1. Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2. Set the annotation for a model object.

```
set (modelObj, 'annotation', 'www.reactome.org')
```

3. Verify the assignment.

```
get (modelObj, 'annotation')
```

MATLAB returns:

```
ans =
```

www.reactome.org

See Also

addkineticlaw, addparameter, addreaction, addrule, addspecies, sbiomodel, sbioroot, sbiounit, sbiounitprefix, RepeatDose object, ScheduleDose object

BoundaryCondition property

Purpose

Indicate species boundary condition

Description

The BoundaryCondition property indicates whether a species object has a boundary condition. If BoundaryCondition is true, the species quantity is determined by InitialAmount and/or a rule object, and not by the reaction rate equation. All SimBiology species are state variables regardless of the BoundaryCondition or ConstantAmount property.

By default, BoundaryCondition is false and the reaction rate equations determine the rate of change of a species quantity in the model. Boundary condition is used when a species is modeled as a participant of reactions but the species quantity is not determined by a reaction rate equation.

More Information

Consider the following two use cases of boundary conditions:

- Modeling receptor-ligand interactions that affect the rate of change of the receptor but not the ligand. For example, in response to hormone, steroid receptors such as the glucocorticoid receptor (GR) translocate from the cytoplasm (cyt) to the nucleus (nuc). The hsp90/hsp70 chaperone complex directs this nuclear translocation [Pratt 2004]. The natural ligand for GR is cortisol; the synthetic hormone dexamethasone (dex) is used in place of cortisol in experimental systems. In this system dexamethasone participates in the reaction but the quantity of dexamethasone in the cell is regulated using a rule. To simply model translocation of GR you could use the following reactions:

  Formation of the chaperone-receptor complex,

  ```
  Hsp90_complex + GR_cyt -> Hsp90_complex:GR_cyt
  ```

  In response to the synthetic hormone dexamethasone (dex), GR moves from the cytoplasm to the nucleus.

  ```
  Hsp90_complex:GR_cyt + dex -> Hsp90_complex + GR_nuc + dex
  ```

For dex,

```
BoundaryCondition = true; ConstantAmount = false
```

In this example dex is modeled as a boundary condition with a rule to regulate the rate of change of dex in the system. Here, the quantity of dex is not determined by the rate of the second reaction but by a rate rule such as

```
ddex/dt = 0.001
```

which is specified in the SimBiology software as

```
dex = 0.001
```

- Modeling the role of nucleotides (for example, GTP, ATP, cAMP) and cofactors (for example, Ca^{++}, NAD^{+}, coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.

  ```
  Ras-GDP + GTP  -> Ras-GTP + GDP
  ```

  ```
  For GTP, BoundaryCondition = true; ConstantAmount = true
  ```

  Model GTP and GDP with boundary conditions, thus making them *boundary species*. In addition, you can set the ConstantAmount property of these species to true to indicate that their quantity does not vary during a simulation.

Characteristics

Applies to	Object: species
Data type	boolean
Data values	true or false. The default value is false.
Access	Read/write

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a species object and verify that the boundary condition property setting is `'false'` or `0`.

```
speciesObj = addspecies(modelObj, 'glucose');
get(speciesObj, 'BoundaryCondition')
```

MATLAB returns:

```
ans =

     0
```

3 Set the boundary condition to `'true'` and verify.

```
set(speciesObj, 'BoundaryCondition', true);
get(speciesObj, 'BoundaryCondition')
```

MATLAB returns:

```
ans =

     1
```

References

Pratt, W.B., Galigniana, M.D., Morishima, Y., Murphy, P.J. (2004), Role of molecular chaperones in steroid receptor action, *Essays Biochem*, 40:41-58.

See Also

`addrule`, `addspecies`, `ConstantAmount`, `InitialAmount`

Purpose

Library of built-in components

Description

`BuiltInLibrary` is a SimBiology root object property containing all built-in components namely, unit, unit-prefixes, and kinetic laws that are shipped with the SimBiology product. You cannot add, modify, or delete components in the built-in library. The `BuiltInLibrary` property is an object that contains the following properties:

- `Units` — contains all units that are shipped with the SimBiology product. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in units either by using the command `sbiowhos -builtin -unit`, or by accessing the root object.
- `UnitPrefixes` — contains all unit-prefixes that are shipped with the SimBiology product. You can specify unit—prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in unit-prefixes either by using the command `sbiowhos -builtin -unitprefix`, or by accessing the root object.
- `KineticLaws` — contains all kinetic laws that are shipped with the SimBiology product. Use the command `sbiowhos -builtin -kineticlaw` to see the list of built-in kinetic laws. You can use built-in kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, `kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');`

  See “Kinetic Law Definition” on page 3-65 for a definition and more information.

Characteristics

`BuiltInLibrary`

BuiltInLibrary property

Applies to	Object: root
Data type	object
Data values	Unit, unit-prefix, and abstract kinetic law objects
Access	Read-only

Characteristics for `BuiltInLibrary` properties:

- `Units`

Applies to	`BuiltInLibrary` property
Data type	unit objects
Data values	units
Access	Read-only

- `UnitPrefixes`

Applies to	`BuiltInLibrary` property
Data type	unit prefix objects
Data values	unit prefixes
Access	Read-only

- `KineticLaws`

Applies to	`BuiltInLibrary` property
Data type	Abstract kinetic law object
Data values	kinetic laws
Access	Read-only

Examples

Example 1

This example uses the command `sbiowhos` to show the current list of built-in components.

```
sbiowhos -builtin -kineticlaw
sbiowhos -builtin -unit
sbiowhos -builtin -unitprefix
```

Example 2

This example shows the current list of built-in components by accessing the root object.

```
rootObj = sbioroot;
get(rootObj.BuiltinLibrary, 'KineticLaws')
get(rootObj.BuiltinLibrary, 'Units')
get(rootObj.BuiltinLibrary, 'UnitPrefixes')
```

See Also

Functions — `sbioaddtolibrary`, `sbioremovefromlibrary` `sbioroot`, `sbiounit`, `sbiounitprefix`

Properties — `UserDefinedLibrary`

Capacity property

Purpose

Compartment capacity

Description

The `Capacity` property indicates the size of the SimBiology compartment object. If the size of the compartment does not vary during simulation, set the property `ConstantCapacity` to `true`.

You can vary compartment capacity using rules or events.

Note Remember to set the `ConstantCapacity` property to false for varying capacity.

Events cannot result in the capacity having a negative value. Rules could result in the capacity having a negative value.

Characteristics

Applies to	Object: compartment
Data type	`double`
Data values	Positive real number. The default value is `1`.
Access	Read/write

Examples

Add a compartment to a model and set the compartment capacity.

1. Create a model object named `my_model`.

   ```
   modelObj = sbiomodel ('comp_model');
   ```

2. Add the compartment object named `nucleus` with a capacity of `0.5`.

   ```
   compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);
   ```

See Also

`addcompartment`, `addspecies`, `CapacityUnits`, `ConstantCapacity`

Purpose

Compartment capacity units

Description

The `CapacityUnits` property indicates the unit definition for the `Capacity` property of a compartment object. `CapacityUnits` can be any unit from the units library. To get a list of the defined units in the library, use the `sbioshowunits` function. If `CapacityUnits` changes from one unit definition to another, the `Capacity` does not automatically convert to the new units. The `sbioconvertunits` function does this conversion. To add a user-defined unit to the list, see `sbioaddtolibrary`.

Characteristics

Applies to	Object: compartment
Data type	`char string`
Data values	Units from library with dimensions of length, area, or volume. Default = `''` (empty).
Access	Read/write

Examples

1. Create a model object named `my_model`.

   ```
   modelObj = sbiomodel ('my_model');
   ```

2. Add a compartment object named `cytoplasm` with a capacity of `0.5`.

   ```
   compObj = addcompartment (modelObj, 'cytoplasm', 0.5);
   ```

3. Set the `CapacityUnits` to `femtoliter`, and verify.

   ```
   set (compObj,'CapacityUnits', 'femtoliter');
   get (compObj,'CapacityUnits')
   ```

   MATLAB returns:

   ```
   ans =

   femtoliter
   ```

See Also

`InitialAmount`, `sbioaddtolibrary`, `sbioconvertunits`, `sbioshowunits`

Purpose

Array of compartments in model or compartment

Description

`Compartments` shows you a read-only array of SimBiology compartment objects in the model object and the compartment object. In the model object, the `Compartments` property indicates all the compartments in a `Model` object as a flat list. In the compartment object, the `Compartments` property indicates other compartments that are referenced within the compartment. The two instances of `Compartments` are illustrated in "Examples" on page 3-21.

You can add a compartment object using the method `addcompartment`.

Characteristics

Applies to	Objects: compartment, model
Data type	Array of compartment objects
Data values	Compartment object. Default is `[]` (empty).
Access	Read-only

Examples

1. Create a model object named `modelObj`.

```
modelObj = sbiomodel('cell');
```

2. Add two compartments to the model object.

```
compartmentObj1 = addcompartment(modelObj, 'nucleus');
compartmentObj2 = addcompartment(modelObj, 'mitochondrion');
```

3. Add a compartment to one of the compartment objects.

```
compartmentObj3 = addcompartment(compartmentObj2, 'matrix');
```

4. Display the `Compartments` property in the model object.

```
get(modelObj, 'Compartments')

SimBiology Compartment Array
```

```
Index:    Name:           Capacity:    CapacityUnits:
1         nucleus         1
2         mitochondrion   1
3         matrix          1
```

5 Display the `Compartments` property in the compartment object.

```
get(compartmentObj2, 'Compartments')

SimBiology Compartment - matrix

   Compartment Components:
     Capacity:          1
     CapacityUnits:
     Compartments:      0
     ConstantCapacity:  true
     Owner:             mitochondrion
     Species:           0
```

See Also

`addcompartment`, `addreaction`, `addspecies`, `Compartment object`

Purpose

Dimensional analysis and unit conversion options

Description

The SimBiology `CompileOptions` property is an object that defines the compile options available for simulation; you can specify whether dimensional analysis and unit conversion is necessary for simulation. Compile options are checked during compile time. The compile options object can be accessed through the `CompileOptions` property of the `configset` object. Retrieve `CompileOptions` object properties with the `get` function and configure the properties with the `set` function.

Property Summary

`DefaultSpeciesDimension`	Dimension of species name in expression
`DimensionalAnalysis`	Perform dimensional analysis on model
`Type`	Display SimBiology object type
`UnitConversion`	Perform unit conversion

Characteristics

Applies to	Object: `configset`
Data type	Object
Data values	Compile-time options
Access	Read-only

Examples

1. Retrieve the `configset` object of `modelObj`.

```
modelObj = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2. Retrieve the `CompileOptions` object (`optionsObj`) from the `configsetObj`.

```
optionsObj = get(configsetObj, 'CompileOptions');
```

```
Compile Settings:

     UnitConversion:       false
     DimensionalAnalysis:  true
```

See Also

get, set

Purpose

Unit composition

Description

The `Composition` property holds the composition of a unit object. The `Composition` property shows the combination of base and derived units that defines the unit. For example, molarity is the name of the unit and the composition is mole/liter. Base units are the set of units used to define all unit quantity equations. Derived units are defined using base units or mixtures of base and derived units.

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

Characteristics

Applies to	Object: `Unit`
Data type	`char string`
Data values	Valid combination of units and prefixes from the library. Default is `''` (empty).
Access	Read/write

Examples

This example shows you how to create a user-defined unit, add it to the user-defined library, and query the `Composition` property.

1. Create a unit for the rate constants of a second-order reaction.

```
unitObj = sbiounit('secondconstant', '1/molarity*second', 1);
```

2. Query the `Composition` property.

```
get(unitObj, 'Composition')

ans =

1/molarity*second
```

3. Change the `Composition` property.

```
set(unitObj, 'Composition', 'liter/mole*second'))

ans =

liter/mole*second
```

4 Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

See Also

`get`, `Multiplier`, `Offset`, `sbiounit`, `set`

ConstantAmount property

Purpose

Specify variable or constant species amount

Description

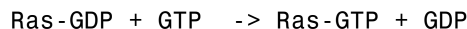
The ConstantAmount property indicates whether the quantity of the species object can vary during the simulation. ConstantAmount can be either true or false. If ConstantAmount is true, the quantity of the species cannot vary during the simulation. By default, ConstantAmount is false and the quantity of the species can vary during the simulation. If ConstantAmount is false, the quantity of the species can be determined by reactions and rules.

The property ConstantAmount is for species objects; the property ConstantValue is for parameter objects.

More Information

The following is an example of modeling species as constant amounts:

Modeling the role of nucleotides (GTP, ATP, cAMP) and cofactors (Ca^{++}, NAD^{+}, coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.



```
Ras-GDP + GTP  -> Ras-GTP + GDP
```

Model GTP and GDP with constant amount set to true. In addition, you can set the BoundaryCondition of these species to true, thus making them *boundary species*.

Characteristics

Applies to	Object: species
Data type	boolean
Data values	true or false. The default value is false.
Access	Read/write

Examples

1 Create a model object named my_model.

```
modelObj = sbiomodel ('my_model');
```

2 Add a species object and verify that the `ConstantAmount` property setting is `'false'` or `0`.

```
speciesObj = addspecies (modelObj, 'glucose');
get (speciesObj, 'ConstantAmount')
```

MATLAB returns:

```
            ans =

     0
```

3 Set the constant amount to `'true'` and verify.

```
set (speciesObj, 'ConstantAmount', true);
get (speciesObj, 'ConstantAmount')
```

MATLAB returns:

```
ans =

     1
```

See Also

`addspecies`, `BoundaryCondition`

Purpose

Specify variable or constant compartment capacity

Description

The `ConstantCapacity` property indicates whether the capacity of the compartment object can vary during the simulation. `ConstantCapacity` can be either `true` (1) or `false` (0). If `ConstantCapacity` is `true`, the quantity of the compartment cannot vary during the simulation. By default, `ConstantCapacity` is `true` and the quantity of the compartment cannot vary during the simulation. If `ConstantCapacity` is `false`, the quantity of the compartment can be determined by rules and events.

Characteristics

Applies to	Object: compartment
Data type	`boolean`
Data values	`true` or `false`. The default value is `true`.
Access	Read/write

Examples

Add a compartment to a model and check the `ConstantCapacity` property of the compartment.

1 Create a model object named `comp_model`.

```
modelObj = sbiomodel ('comp_model');
```

2 Add the compartment object named `nucleus` with a capacity of `0.5`.

```
compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);
```

3 Display the `ConstantCapacity` property.

```
get(compartmentObj, 'ConstantCapacity')

ans =

     1
```

See Also

`addcompartment`, `ConstantAmount`, `ConstantValue`

ConstantValue property

Purpose

Specify variable or constant parameter value

Description

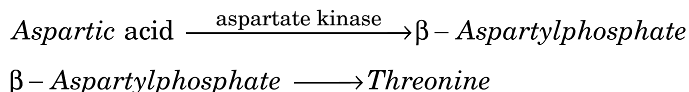
The ConstantValue property indicates whether the value of a parameter can change during a simulation. Enter either true (value is constant) or false (value can change).

You can allow the value of the parameter to change during a simulation by specifying a rule that changes the Value property of the parameter object.

The property ConstantValue is for parameter objects; the property ConstantAmount is for species objects.

More Information

As an example, consider feedback inhibition of an enzyme such as aspartate kinase by threonine. Aspartate kinase has three isozymes that are independently inhibited by the products of downstream reactions (threonine, homoserine, and lysine). Although threonine is made through a series of reactions in the synthesis pathway, for illustration, the reactions are simplified as follows:



$$Aspartic\ \text{acid} \xrightarrow{\text{aspartate kinase}} \beta - Aspartylphosphate$$

$$\beta - Aspartylphosphate \longrightarrow Threonine$$

To model inhibition of aspartate kinase by threonine, you could use a rule like the algebraic rule below to vary the rate of the above reaction and simulate inhibition. In the rule, the rate constant for the above reaction is denoted by k_aspartate_kinase and the quantity of threonine is threonine.

```
k_aspartate_kinase -(1/threonine)
```

Characteristics

Applies to	Object: parameter
Data type	boolean

Data values	`true` or `false`. The default value is `'true'`.
Access	Read/write

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

3 Change the `ConstantValue` property of the parameter object from default (`true`) to `false` and verify.

MATLAB returns `1` for `true` and `0` for `false`.

```
set (parameterObj, 'ConstantValue', false);
get(parameterObj, 'ConstantValue')
```

MATLAB returns:

```
ans =

     0
```

See Also

`addparameter`

Content property

Purpose

Contents of variant object

Description

The `Content` property contains the data for the variant object. `Content` is a cell array with the structure `{'Type', 'Name', 'PropertyName', 'PropertyValue'}`. You can store values for species `InitialAmount`, parameter `Value`, and compartment `Capacity`, in a variant object.

For more information about variants, see `Variant object`.

Characteristics

Applies to	Object: variant
Data type	`cell array`
Data values	Default value is `[]` (empty).
Access	Read/write

Examples

1 Create a model containing three species in one compartment.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
A = addspecies(compObj, 'A');
B = addspecies(compObj, 'B');
C = addspecies(compObj, 'C');
```

2 Add a variant object that varies the species' `InitialAmount` property.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {{'species','A', 'InitialAmount', 5}, ...
{'species', 'B', 'InitialAmount', 10}});
% Display the variant
variantObj

SimBiology Variant - v1 (inactive)

   ContentIndex:     Type:        Name:             Property:           Value:
   1                 species      A                 InitialAmount       5
   2                 species      B                 InitialAmount       10
```

3 Append data to the `Content` property.

```
addcontent(variantObj, {'species', 'C', 'InitialAmount', 15});

SimBiology Variant - v1 (inactive)

   ContentIndex:     Type:        Name:            Property:           Value:
   1                 species      A                InitialAmount        5
   2                 species      B                InitialAmount       10
   3                 species      C                InitialAmount       15
```

4 Remove a species from the `Content` property.

```
rmcontent(variantObj, 3);
```

5 Replace the data in the `Content` property.

```
set(variantObj, 'Content', {'species', 'C', 'InitialAmount', 15});
```

See Also

`addcontent`, `rmcontent`, `sbiovariant`

CovariateLabels property

Purpose

Identify covariate columns in data set

Description

`CovariateLabels` is a property of the `PKData` object. It specifies the column in DataSet that contains the covariate data.

Characteristics

Applies to	Object: `PKData`
Data type	`char string` or cell array of strings
Data values	Column headers from imported data set
Access	Read/write

See Also

`PKData object`

How To

- "Specifying and Classifying the Data to Fit"
- "Specifying a Covariate Model"

CovariateLabels (CovariateModel) property

Purpose

Labels for covariates in `CovariateModel` object

Description

The `CovariateLabels` property is a cell array of strings specifying the labels for the covariates in the `Expression` property of a `CovariateModel` object.

Characteristics

Applies to	Object: `CovariateModel`
Data type	Cell array of strings
Data values	Labels for the covariates in the `Expression` property
Access	Read only

See Also

`CovariateModel` | `Expression`

Data property

Purpose

Store simulation data

Description

The `Data` property contains the simulation data stored in the SimData object.

This property contains all data logged during a simulation, including species amounts, parameter values, and sensitivities. The property is an `m x n` array, where `m` is the number of time steps in the simulation and `n` is the number of quantities logged. The rows of the array are labeled by the time points in the `Time` property, and the columns are labeled by the metadata in the `DataInfo` property.

Characteristics

Applies to	Object: SimData
Data type	`double`
Data values	Default value is `[]` (empty).
Access	Read-only

See Also

`DataInfo`, `ModelName`

Purpose

Numbers of species, parameters, sensitivities

Description

The `DataCount` property shows how many species, parameters, and sensitivities are logged in a SimData object. It is a MATLAB structure with the fields `Species`, `Parameter`, and `Sensitivity`. The information in this property is redundant with the `DataInfo` property and is there to give you a convenient means to access the information.

Characteristics

Applies to	Object: SimData
Data type	`struct`
Data values	Default value for each field is `0`.
Access	Read-only

See Also

`StopTime`, `StopTimeType`

DataInfo property

Purpose

Metadata labels for simulation data

Description

The `DataInfo` property contains the metadata that label the columns of the SimData object array. It is an `n x 1` cell array of structures. The `ith` cell contains metadata labeling the `ith` column of the SimData object array.

The possible types of structures are as follows.

Type	Fields
Species	Type: species Name: Compartment: Units:
Parameter	Type: parameter Name: Reaction: <name of reaction that a parameter is scoped to, or '' if parameter is scoped to model> Units:
Sensitivity	Type: sensitivity Name: <for example: d[x]/d[y]_0> OutputType: <The type of the sensitivity output, either 'species' or 'parameter'> OutputName: <The name of the sensitivity output> OutputQualifier: <The compartment or reaction for the sensitivity output, for species or parameters, respectively> InputType: <The type of the sensitivity input, either 'species' or 'parameter'> InputName: <The name of the sensitivity input> InputQualifier: <The compartment or reaction for

Type	Fields
	the sensitivity input, for species or parameters, respectively> Units:

Characteristics

Applies to	Object: SimData
Data type	`n x 1` cell array of `structs`
Data values	Default value is `0x1` cell array.
Access	Read-only

See Also

`StopTime`, `StopTimeType`

DataNames property

Purpose

Show names in SimData object

Description

The `DataNames` property holds the names that label the columns of the data matrix in the `Data` property. The property contains an `nx1` array of strings. The software provides this information for your convenience.

Characteristics

Applies to	Object: SimData
Data type	`string array`
Data values	Default value is `0x1` cell array.
Access	Read-only

See Also

`StopTime`, `StopTimeType`

Purpose

`Dataset` object containing imported data

Description

`DataSet` is a property of the `PKData` object. It contains the imported data set. The `PKData` object constructor (`PKData`) assigns the specified data set to its `DataSet` property during construction.

Characteristics

Applies to	Object: `PKData`
Data type	`dataset` object
Data values	Variable containing `dataset` object
Access	Read-only

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`

DefaultSpeciesDimension property

Purpose

Dimension of species name in expression

Description

The `DefaultSpeciesDimension` property specifies how SimBiology interprets species names in expressions (namely reaction rate, rule, or event expressions). The valid property values are `substance` or `concentration`. If you specify `InitialAmountUnits`, SimBiology interprets species names appearing in expressions as concentration or substance amount according to the units specified, regardless of the value in `DefaultSpeciesDimension`. Thus, if `DefaultSpeciesDimension` is `concentration` and you specify species units as `molecule`, SimBiology interprets species names in expressions as `substance`. This interpretation applies even when `DimensionalAnalysis` is off.

You can find `DefaultSpeciesDimension` in the `CompileOptions` property.

When you set `DefaultSpeciesDimension` to `substance`, if you do not specify units, SimBiology interprets species names appearing in expressions as substance amounts, and does not scale by compartment capacity. To include a species concentration in an expression, divide by the appropriate compartment capacity in the expression. To specify compartment capacity in an expression enter the compartment name.

When you set `DefaultSpeciesDimension` to `concentration`, SimBiology interprets species names appearing in expressions as concentrations, and scales by compartment capacity in the expressions. To include a species amount in an expression, multiply by the species name by the appropriate compartment name in the expression.

For information on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated”.

Characteristics

Applies to	Object: `CompileOptions` (in `configset` object)
Data type	`char string`

Data values	`concentration` or `substance`. Default value is `concentration`.
Access	Read/write

See Also

`CompileOptions`, `DimensionalAnalysis`, `get`, `getconfigset`, `sbiosimulate`, `set`

DependentVarLabel property

Purpose

Identify dependent variable column in data set

Description

`DependentVarLabel` is a property of a `PKData` object. It specifies the column(s) in DataSet that contain the dependent variable(s), for example, measured response(s). The column must contain numeric values, and cannot contain `Inf` or `Inf`.

Characteristics

Applies to	Object: `PKData`
Data type	`char string` or cell array of strings
Data values	Column header from an imported data set
Access	Read/write

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`

Purpose

Response units in PKData object

Description

`DependentVarUnits` is a property of a `PKData` object. It specifies the units for the column(s) containing the dependent variable(s) (responses) in the imported data set. If unit conversion is on, plot results in the SimBiology desktop show the units specified in `DependentVarUnits`.

To get a list of units, use the `sbioshowunits` function.

Characteristics

Applies to	Object: `PKData`
Data type	`char string` or cell array of strings
Data values	Units from the units library. Default is an empty cell array.

Tip If there are no units associated with the dependent variable(s) in your data set, you can set this property to a cell array of empty strings, or simply an empty cell array.

Access	Read/write

See Also

`DependentVarLabel`, `PKData Object`

DimensionalAnalysis property

Purpose

Perform dimensional analysis on model

Description

The `DimensionalAnalysis` property specifies whether to perform dimensional analysis on the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object. When `DimensionalAnalysis` is set to `true`, the SimBiology software checks whether the physical quantities of the units involved in reactions and rules, match and are applicable.

For example, consider a reaction `a + b  > c`. Using mass action kinetics, the reaction rate is defined as `a*b*k`, where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are 0.01M and 0.005M respectively, then units of `k` are `1/(M*second)`. If you specify `k` with another equivalent unit definition, for example, `1/[(moles/liter)*second]`, `DimensionalAnalysis` checks whether the physical quantities match. If the physical quantities do not match, you see an error and the model is not simulated.

Unit conversion requires dimensional analysis. If `DimensionalAnalysis` is off, and you turn `UnitConversion` on, then `DimensionalAnalysis` is turned on automatically. If `UnitConversion` is on and you turn off `DimensionalAnalysis`, then `UnitConversion` is turned off automatically.

If you have MATLAB function calls in your model, dimensional analysis ignores any expressions containing function calls and generates a warning.

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

Characteristics

Applies to	Object: `CompileOptions` (in `configset` object)
Data type	`boolean`

Data values	`true` or `false`. Default value is `true`.
Access	Read/write

Examples

This example shows how to retrieve and set `DimensionalAnalysis` from the default `true` to `false` in the default configuration set in a model object.

1 Import a model.

```
modelObj = sbmlimport('oscillator')


SimBiology Model - Oscillator

   Model Components:
     Models:            0
     Parameters:        0
     Reactions:         42
     Rules:             0
     Species:           23
```

2 Retrieve the `configset` object of the model object.

```
configsetObj = getconfigset(modelObj)

 Configuration Settings - default (active)
     SolverType:            ode15s
     StopTime:              10.000000

   SolverOptions:
     AbsoluteTolerance:     1.000000e-006
     RelativeTolerance:     1.000000e-003

   RuntimeOptions:
     StatesToLog:           all
```

```
CompileOptions:
  UnitConversion:       true
  DimensionalAnalysis:  true
```

3 Retrieve the `CompileOptions` object.

```
optionsObj = get(configsetObj,'CompileOptions')

Compile Settings:

     UnitConversion:       true
     DimensionalAnalysis:  true
```

4 Assign a value of `false` to `DimensionalAnalysis`.

```
set(optionsObj,'DimensionalAnalysis', false)
```

See Also

`get`, `getconfigset`, `sbiosimulate`, `set`

Purpose

Dosed object name

Description

`Dosed` is a property of the `PKModelMap` object. It specifies the name(s) of species object(s) that receive an input, such as a drug in a compartment or a ligand.

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the `Dosed` property and the `DosingType` property.

Characteristics

Applies to	Object: `PKModelMap`
Data type	`char string` or cell array of strings
Data values	Name of a species object or empty. Default is an empty cell array.
Access	Read/write

See Also

"Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" in the SimBiology User's Guide, `DosingType`, `Estimated`, `Observed`, `PKModelMap object`

DoseLabel property

Purpose

Dose column in data set

Description

`DoseLabel` is a property of the `PKData` object. `DoseLabel` specifies the column that contains that contains the dosing information, in DataSet. The column must contain positive values, and cannot contain `Inf` or `Inf`.

Characteristics

Applies to	Object: `PKData`
Data type	string or array of strings
Data values	Column headers from imported data set
Access	Read/write

See Also

`PKData object`, `sbionmimport`, `sbionmfiledef`, “Specifying and Classifying the Data to Fit” in the SimBiology documentation

DoseUnits property

Purpose

Dose units in PKData object

Description

The `DoseUnits` property indicates the units for dose values in the PKData object. Dose units must have dimensions of amount or mass. The length of `DoseUnits` must be the same as `DoseLabel`. For example, if the `DoseLabel` property defines two columns containing dosing information, `DoseUnits` must also define units for both columns. If unit conversion is on, dose and rate units must be consistent with each other (that is in terms of amount or mass) and must be consistent with the species object that is being dosed.

To get a list of units, use the `sbioshowunits` function.

Characteristics

Applies to	Object: PKData
Data type	string or array of strings
Data values	Units from units library. Default is `''` (empty).
Access	Read/write

See Also

`DoseLabel`, `PKData Object`

DosingType property

Purpose

Drug dosing type in compartment

Description

`DosingType` is a property of the `PKCompartment` and `PKModelMap` objects. It specifies the type of dosing of a drug in a compartment. You can only dose one compartment in the model at any given time. For a description of the types of dosing supported, the model components created for each type of dosing, and the parameters to estimate, see "Dosing Types".

Characteristics

Applies to	Objects: `PKCompartment`, `PKModelMap`
Data type	`char string` or cell array of strings
Data values	`''`, `'Bolus'`, `'Infusion'`, `'ZeroOrder'`, `'FirstOrder'`
Access	Read/write

See Also

`EliminationType`, `PKCompartment object`, `PKModelMap object`

Purpose

Parameter specifying length of time

Description

`DurationParameterName` is a property of a `RepeatDose` or `ScheduleDose` object.

Specify the name of a parameter object that is:

- Scoped to a model
- Constant, that is, its `ConstantValue` property is `true`

This property specifies the length of time it takes to administer a dose.

Note If you set the `DurationParameterName` property of a dose, you must also specify the `Amount` property of the dose, and set the `Rate` property to `0`. This is because the rate is calculated from the amount and duration.

Characteristics

Applies to	Objects: `RepeatDose`, `ScheduleDose`
Data type	`char string`
Data values	Name of a parameter object or empty. Default is an empty string. The parameter object must be: • Scoped to a model • Constant, that is, have a `ConstantValue` property set to `true`
Access	Read/write

See Also

`RepeatDose object`, `ScheduleDose object`

EliminationType property

Purpose

Drug elimination type from compartment

Description

`EliminationType` is a property of the `PKCompartment` object. It specifies the type of elimination of adrug from a compartment. For a description of the types of elimination supported, the model components created for each type of elimination, and the parameters to estimate, see “Elimination Types”.

Characteristics

Applies to	Object: `PKCompartment`
Data type	`char string`
Data values	`'Linear'`, `'Linear-Clearance'`, `'Enzymatic'`, and `''`
Access	Read/write

See Also

`addCompartment`, `DosingType`, `PKCompartment object`

Purpose

Specify explicit or implicit tau error tolerance

Description

The `ErrorTolerance` property specifies the error tolerance for the explicit tau and implicit tau stochastic solvers. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object. The explicit and implicit tau solvers automatically chooses a time interval (`tau`) such that the relative change in the propensity function for each reaction is less than the user-specified error tolerance.

A propensity function describes the probability that the reaction will occur in the next smallest time interval, given the conditions and constraints.

If the error tolerance is too large, there may not be a solution to the problem and that could lead to an error. If the error tolerance is small, the solver will take more steps than when the error tolerance is large leading to longer simulation times. The error tolerance should be adjusted depending upon the problem, but a good value for the error tolerance is between `1 %` to `5 %`.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`
Data values	>0, <1. The default is `3e-2`.
Access	Read/write

Examples

This example shows how to change `ErrorTolerance` settings.

1. Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `expltau`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'expltau')
```

2. Change the `ErrorTolerance` to `1e-8`.

```
set(configsetObj.SolverOptions, 'ErrorTolerance', 5.0e-2);
get(configsetObj.SolverOptions, 'ErrorTolerance')


ans =

  5.000000e-002
```

See Also LogDecimation, RandomState

Purpose

Names of parameters to estimate

Description

`Estimated` is a property of the `PKModelMap` object. It specifies the name(s) of the object(s) to estimate. Specify the name(s) of species, compartment, or parameter object(s) that are scoped to a model.

Note If you specify a species object, you are estimating the `InitialAmount` property of the species object.

Characteristics

Applies to	Object: `PKModelMap`
Data type	`char string` or cell array of strings
Data values	Name of a species, compartment, or parameter object or empty. Default is an empty cell array.
Access	Read/write

See Also

"Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" in the SimBiology User's Guide, `Dosed`, `InitialAmount`, `Observed`, `PKModelMap object`

EventFcns property

Purpose Event expression

Description Property of the event object that defines what occurs when the event is triggered. Specify a cell array of strings.

`EventFcns` can be any MATLAB assignment or expression that defines what is executed when the event is triggered. All `EventFcn` expressions are assignments of the form '*objectname = expression*', where *objectname* is the name of a valid SimBiology object.

For more information about how SimBiology handles events, see "How Events Are Evaluated". For examples of event functions, see "Specifying Event Functions".

Characteristics

Applies to	Object: event
Data type	Cell array of strings
Data values	`EventFcn` strings `''` (empty)
Access	Read/write

Examples

1 Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator');
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

2 Set the `EventFcns` property of the event object.

```
set(eventObj, 'EventFcns', {'pA = OpA','mA = pol'});
```

3 Get the `EventFcns` property.

```
get(eventObj, 'EventFcns')
```

See Also `Event object`, `Trigger`

Purpose

Contain all event objects

Description

Property to indicate events in a model object. Read-only array of `Event` objects.

An event defines an action when a defined condition is met. For example, the quantity of a species may double when the quantity of species `B` is `100`. An event is triggered when the conditions specified in the event are met by the model. For more information, see "Events" and "Event Object".

Add an event to a `Model` object with the method `addevent` method and remove an event with the `delete` method. See `Event object` for more information.

You can view event object properties with the `get` command and modify the properties with the `set` command.

Characteristics

Applies to	Object: model
Data type	Array of event objects
Data values	Event object. The default is `[]` (empty).
Access	Read-only

Examples

1. Create a model object, and then add an event object.

   ```
   modelObj = sbmlimport('oscillator')
   eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
   ```

2. Get a list of properties for an event object.

   ```
   get(modelObj.Events(1));
   ```

   Or

   ```
   get(eventObj)
   ```

MATLAB displays a list of event properties.

```
             Active: 1
         Annotation: ''
          EventFcns: {'OpC = 200'}
               Name: ''
              Notes: ''
             Parent: [1x1 SimBiology.Model]
                Tag: ''
            Trigger: 'time >= 5'
       TriggerDelay: 0
  TriggerDelayUnits: 'second'
               Type: 'event'
           UserData: []
```

See Also

EventFcns, Event object, Model object, Trigger

Purpose

Exponent value of unit prefix

Description

`Exponent` shows the value of `10^Exponent` that defines the numerical value of the unit prefix *`Name`*. You can use the unit prefix in conjunction with any built-in or user-defined units. For example, for the unit `mole`, specify as `picomole` to use the `Exponent`, `-12`.

Characteristics

Applies to	Object: Unit prefix
Data type	`double`
Data values	Real number. Default is `0`.
Access	Read/write

Examples

This example shows you how to create a user-defined unit prefix, add it to the user-defined library, and query the `Exponent` property.

1. Create a unit prefix.

```
unitprefixObj1 = sbiounitprefix('peta', 15);
```

2. Add the unit prefix to the user-defined library.

```
sbioaddtolibrary(unitprefixObj1);
```

3. Query the `Exponent` property.

```
get(unitprefixObj1, 'Exponent')

ans =

    15
```

See Also

`get`, `sbioaddtolibrary`, `sbiounitprefix`, `set`, `UnitPrefix object`

Expression (CovariateModel) property

Purpose

Define relationship between parameters and covariates

Description

The `Expression` property is a string or cell array of strings, where each string represents the relationship between a parameter and one or more covariates. The `Expression` property denotes fixed effects with the prefix `theta`, and random effects with the prefix `eta`.

Each expression string must be in the form:

```
parameterName = relationship
```

This example of an expression string defines the relationship between a parameter (`volume`) and a covariate (`weight`), with fixed effects, but no random effects:

```
CovModelObj.Expression = {'volume = theta1 +
theta2*weight'};
```

This table illustrates expression formats for some common parameter-covariate relationships.

Parameter-Covariate Relationship	Expression Format
Linear with random effect	`Cl = theta1 + theta2*WEIGHT + eta1`
Exponential without random effect	`Cl = exp(theta_Cl + theta_Cl_WT*WEIGHT)`
Exponential, WEIGHT centered by mean, has random effect	`Cl = exp(theta1 + theta2*(WEIGHT - mean(WEIGHT)) + eta1)`

Parameter-Covariate Relationship	Expression Format
Exponential, log(WEIGHT), which is equivalent to power model	`Cl = exp(theta1 + theta2*log(WEIGHT) + eta1)`
Exponential, dependent on WEIGHT and AGE, has random effect	`Cl = exp(theta1 + theta2*WEIGHT + theta3*AGE + eta1)`

Tip To simultaneously fit data from multiple dose levels, use a `CovariateModel` object as an input argument to `sbionlmefit`, and omit the random effect (`eta`) from the `Expression` property in the `CovariateModel` object.

The `Expression` property must meet the following requirements:

- The expression strings are valid MATLAB code.
- Each expression string is linear with a transformation.
- There is exactly one expression string for each parameter.
- In each expression string, a covariate is used in at most one term.
- In each expression string, there is at most one random effect (`eta`)
- Fixed effect (`theta`) and random effect (`eta`) names are unique within and across expression strings. That is, each covariate has its own fixed effect.

Expression (CovariateModel) property

Tip Use the `getCovariateData` method to view the covariate data when writing equations for the `Expression` property of a `CovariateModel` object.

Tip Use the `verify` method to check that the `Expression` property of a `CovariateModel` object meets the conditions described previously.

Characteristics

Applies to	Object: `CovariateModel`
Data type	String or cell array of strings
Data values	`parameterName = relationship`
Access	Read/write

See Also

`CovariateModel` | `getCovariateData` | `verify`

How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- "Specifying a Covariate Model"

Expression (AbstractKineticLaw, KineticLaw) property

Purpose

Expression to determine reaction rate equation

Description

The `Expression` property indicates the mathematical expression that is used to determine the `ReactionRate` property of the reaction object. `Expression` is a reaction rate expression assigned by the kinetic law definition used by the reaction. The kinetic law being used is indicated by the property `KineticLawName`. You can configure `Expression` for user-defined kinetic laws, but not for built-in kinetic laws. `Expression` is read only for kinetic law objects.

Note If you set the `Expression` property to a reaction rate expression that is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" before simulating your model.

Kinetic Law Definition

The *kinetic law definition* provides a mechanism for applying a specific rate law to multiple reactions. It acts as a mapping template for the reaction rate. The kinetic law is defined by a mathematical expression, (defined in the property `Expression`), and includes the species and parameter variables used in the expression. The species variables are defined in the `SpeciesVariables` property, and the parameter variables are defined in the `ParameterVariables` property of the kinetic law object.

If a reaction is using a kinetic law definition, the `ReactionRate` property of the reaction object shows the result of a mapping from the kinetic law definition. To determine `ReactionRate`, the species variables and parameter variables that participate in the reaction rate should be mapped in the kinetic law for the reaction. In this case, SimBiology software determines the `ReactionRate` by using the `Expression` property of the abstract kinetic law object, and by mapping `SpeciesVariableNames` to `SpeciesVariables` and `ParameterVariableNames` to `ParameterVariables`.

For example, the kinetic law definition `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 `Henri-Michaelis-Menten` kinetic law to a reaction `A -> B` with `Va` mapping to `Vm`, `A` mapping to `S`, and `Ka` mapping to `Km`, the rate equation for the reaction becomes Va*A/(Ka+A).

The exact expression of a reaction using `MassAction` kinetic law varies depending upon the number of reactants. Thus, for mass action kinetics the `Expression` property is set to `MassAction` because in general for mass action kinetics the reaction rate is defined as

$$r = k\prod_{i=1}^{n_r}[Si]^{m_i}$$

where `[Si]` is the concentration of the `ith` reactant, m_i is the stoichiometric coefficient of `[Si]`, n_r is the number of reactants, and `k` is the mass action reaction rate constant.

SimBiology software contains some built-in kinetic laws. You can also define your own kinetic laws. To find the list of available kinetic laws, use the `sbiowhos -kineticlaw` command (`sbiowhos`). You can create a kinetic law definition with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

Characteristics

Applies to	Objects: abstract kinetic law, kinetic law
Data type	`char string`
Data values	Defined by kinetic law definition
Access	Read-only in kinetic law object. Read/write in user-defined kinetic law.

Examples

Example 1

Example with Henri-Michaelis-Menten kinetics

1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Verify that the Expression property for the kinetic law object is Henri-Michaelis-Menten.

```
get (kineticlawObj, 'Expression')
```

MATLAB returns:

```
ans =

Vm*S/(Km + S)
```

4 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that you should set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm_d, Km_d, and assign the objects' Parent property value to the kineticlawObj. The species object with Name a is created when reactionObjis created and need not be redefined.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

5 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Vm_d' 'Km_d'});
set(kineticlawObj,'SpeciesVariableNames', {'a'});
```

6. Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Vm_d*a/(Km_d+a)
```

Example 2

Example with `Mass Action` kinetics.

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2. Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
get(kineticlawObj, 'Expression')
```

MATLAB returns:

```
ans =

MassAction
```

3. Assign the rate constant for the reaction.

```
set (kineticlawObj, 'ParameterVariablenames', 'k');

get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
k*a*b
```

See Also

KineticLawName, Parameters, ParameterVariableNames, ParameterVariables, ReactionRate, sbioaddtolibrary, sbiowhos, SpeciesVariables, SpeciesVariableNames

FixedEffectDescription (CovariateModel) property

Purpose

Descriptions of fixed effects in `CovariateModel` object

Description

The `FixedEffectDescription` property is a cell array of strings describing the fixed effects in the `Expression` property of a `CovariateModel` object. Each string describes the role of a fixed effect in the expression equation. For example, in the following expression equation:

```
Cl = exp(theta1 + theta2*WEIGHT + theta3*AGE + eta1)
```

The description for the fixed effect `theta1` is `'Cl'`, which indicates it is the intercept for the parameter `Cl`. Also, the description for the fixed effect `theta2` is `'Cl/WEIGHT'`, which indicates it is the slope of the line that defines the relationship between the parameter `Cl` and the covariate `WEIGHT`.

Characteristics

Applies to	Object: `CovariateModel`
Data type	Cell array of strings
Data values	Description of the roles of the fixed effects in the `Expression` property
Access	Read only

See Also

`CovariateModel` | `Expression` | `FixedEffectNames` | `FixedEffectValues`

Purpose

Names of fixed effects in `CovariateModel` object

Description

The `FixedEffectNames` property is a cell array of strings specifying the names of the fixed effects in the `Expression` property of a `CovariateModel` object. Names of fixed effects are denoted with the prefix `theta`.

Characteristics

Applies to	Object: `CovariateModel`
Data type	Cell array of strings
Data values	Names of the fixed effects in the `Expression` property. These name are denoted with the prefix `theta`.
Access	Read only

See Also

`CovariateModel` | `Expression` | `FixedEffectDescription` | `FixedEffectValues`

FixedEffectValues (CovariateModel) property

Purpose

Values for initial estimates of fixed effects in `CovariateModel` object

Description

The `FixedEffectValues` property is a structure containing one field for each fixed effect in the `Expression` property of a `CovariateModel` object. Each field contains the value of the initial estimate for a fixed effect.

Tip You must set this property before using the `CovariateModel` object as input to `sbionlmefit` or `sbionlmefitsa`. Use the `constructDefautlFixedEffectValues` method to create a structure of fixed-effect initial estimate values, set to a default of zero. Then edit the structure and use it to modify this property.

Characteristics

Applies to	Object: `CovariateModel`
Data type	Structure with one field for each fixed effect
Data values	Each field contains a double specifying the value of the initial estimate for a fixed effect in the `CovariateModel` object
Access	Read/write

See Also

`CovariateModel` | `constructDefautlFixedEffectValues` | `Expression` | `FixedEffectDescription` | `FixedEffectNames`

How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specifying a Covariate Model”

Purpose

Integer identifying each group in data set

Description

`GroupID` is a property of the `PKData` object. It is an array of the same length as the DataSet property containing an integer to identify each group. `PKData` sets this property during construction of the `PKData` object.

Characteristics

Applies to	Object: `PKData`
Data type	`double`
Data values	Index value for each group
Access	Read-only

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`

GroupLabel property

Purpose

Identify group column in data set

Description

`GroupLabel` is a property of the `PKData` object. It specifies the column in DataSet that contains the group identification labels.

Characteristics

Applies to	Object: `PKData`
Data type	`char string`
Data values	Column header string from imported data set
Access	Read/write

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`, `GroupNames`

Purpose

Unique values from `GroupLabel` in data set

Description

`GroupNames` is a property of the `PKData` object. It contains unique values from the data column specified by the `GroupLabel` property. `PKData` sets this property during construction of the `PKData` object.

Characteristics

Applies to	Object: `PKData`
Data type	`char string` or cell array of strings
Data values	Unique values in `GroupLabel`
Access	Read-only

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`, `GroupLabel`

HasLag property

Purpose

Lag associated with dose targeting compartment

Description

HasLag is a property of the PKCompartment object. It is a logical indicating if the dose targeting the compartment has a time lag or not.

Characteristics

Applies to	Object: PKCompartment
Data type	logical
Data values	1 (true) or 0 (false). Default is 0 (false).
Access	Read/write

See Also

addCompartment, DosingType, EliminationType, PKCompartment object

Purpose

Compartment drug concentration reported

Description

`HasResponseVariable` is a property of the `PKCompartment` object. It is a logical indicating if the drug concentration in this compartment is reported.

Note The `HasResponseVariable` property can be `true` for more than one `PKCompartment` object in the model. If you perform a parameter fit on a model, at least one `PKCompartment` object in the model must have a `HasResponseVariable` property set to `true`.

Characteristics

Applies to	Object: `PKCompartment`
Data type	Logical
Data values	`1` (true) or `0` (false). Default is `0` (false).
Access	Read/write

See Also

`addCompartment`, `DosingType`, `EliminationType`, `PKCompartment object`

IndependentVarLabel property

Purpose

Identify independent variable column in data set

Description

`IndependentVarLabel` is a property of the `PKData` object. It specifies the column in DataSet that contains the independent variable (for example, time).

The column must contain positive values, and cannot contain, `NaN`, `Inf` or `Inf`.

Characteristics

Applies to	Object: `PKData`
Data type	`char string`
Data values	Column header from imported data set
Access	Read/write

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`

Purpose

Time units in PKData object

Description

The `IndependentVarUnits` property indicates the units for the column containing the independent variable (time) in the `PKData` object. If unit conversion is on, plot results in the SimBiology desktop show the units specified in `IndependentVarUnits`.

To get a list of units, use the `sbioshowunits` function.

Characteristics

Applies to	Object: PKData
Data type	string
Data values	Time units. Default is `''` (empty).
Access	Read/write

See Also

`DependentVarLabel`, `PKData Object`

InitialAmount property

Purpose

Species initial amount

Description

The InitialAmount property indicates the initial quantity of the SimBiology species object. InitialAmount is the quantity of the species before the simulation starts.

Characteristics

Applies to	Object: species
Data type	double
Data values	Positive real number. Default value is 0.
Access	Read/write

Examples

Add a species to a model and set the initial amount of the species.

1. Create a model object named my_model.

```
modelObj = sbiomodel ('my_model');
```

2. Add the species object named glucose.

```
speciesObj = addspecies (modelObj, 'glucose');
```

3. Set the initial amount to 100 and verify.

```
set (speciesObj, 'InitialAmount',100);
get (speciesObj, 'InitialAmount')
```

MATLAB returns:

```
ans =

   100
```

See Also

addspecies, InitialAmountUnits

Purpose

Species initial amount units

Description

The `InitialAmountUnits` property indicates the unit definition for the `InitialAmount` property of a species object. `InitialAmountUnits` can be one of the built-in units. To get a list of the defined units, use the `sbioshowunits` function. If `InitialAmountUnits` changes from one unit definition to another, `InitialAmount` does not automatically convert to the new units. The `sbioconvertunits` function does this conversion. To add a user-defined unit to the list, use `sbiounit` followed by `sbioaddtolibrary`.

See `DefaultSpeciesDimension` for more information on specifying dimensions for species quantities. `InitialAmountUnits` must have corresponding dimensions to `CapacityUnits`. For example, if the `CapacityUnits` are `meter`^{2}, then species must be `amount/meter`^{2} or `amount`.

Characteristics

Applies to	Object: species
Data type	`char string`
Data values	Units from library with dimensions of amount, amount/length, amount/area, or amount/volume. Default is `''` (empty).
Access	Read/write

Examples

1. Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');
compObj = addcompartment(modelObj, 'cell');
```

2. Add a species object named `glucose`.

```
speciesObj = addspecies (compObj, 'glucose');
```

3. Set the initial amount to `100`, `InitialAmountUnits` to `molecule`, and verify.

```
set (speciesObj,'InitialAmountUnits','molecule');
get (speciesObj,'InitialAmountUnits')
```

MATLAB returns:

```
ans =

molecule
```

See Also

DefaultSpeciesDimension, InitialAmount, sbioaddtolibrary, sbioconvertunits, sbioshowunits, sbiounit,

Purpose

Specify species and parameter input factors for sensitivity analysis

Description

`Inputs` is a property of the `SensitivityAnalysisOptions` object. `SensitivityAnalysisOptions` is a property of the configuration set object.

Use `Inputs` to specify the species and parameters with respect to which you want to compute the sensitivities of the species or parameter states in your model.

The SimBiology software calculates sensitivities with respect to the values of the parameters and the initial amounts of the species specified in the `Inputs` property. When you simulate a model with `SensitivityAnalysis` enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species and parameters specified in the `Outputs` property. For a description of the output, see the `SensitivityAnalysisOptions` property description.

Characteristics

Applies to	Object: `SensitivityAnalysisOptions`
Data type	Species or parameter object or an array of objects **Note** If this object is determined by a repeated assignment rule, then you cannot use it as an `Inputs` property.
Data values	Species or parameter object, or an array of objects. Default is `[]` (empty array).
Access	Read/write

Examples

This example shows how to set `Inputs` for sensitivity analysis.

1. Import the radio decay model from the SimBiology demos.

```
modelObj  = sbmlimport('radiodecay');
```

2. Retrieve the configuration set object from `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

3. Add a parameter to the `Inputs` property and display it. Use the `sbioselect` function to retrieve the parameter object from the model.

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1         c        0.5       1/second
```

See Also

`Outputs`, `sbioselect`, `SensitivityAnalysis`, `SensitivityAnalysisOptions`

Purpose

Time between doses

Description

`Interval` is a property of a `RepeatDose` object. This property defines the equally spaced times between repeated doses.

Note When the `Interval` property is `0`, `RepeatDose` ignores the `RepeatCount` property, that is, it treats it as though it is set to `0`.

Characteristics

Applies to	Object: `RepeatDose`
Data type	`double`
Data values	Nonnegative real number. Default is `0`
Access	Read/Write

See Also

`RepeatDose object`, `ScheduleDose object`

KineticLaw property

Purpose

Show kinetic law used for ReactionRate

Description

The KineticLaw property shows the kinetic law that determines the reaction rate specified in the ReactionRate property of the reaction object. This property shows the kinetic law used to define ReactionRate.

KineticLaw can be configured with the addkineticlaw method. The addkineticlaw function configures the ReactionRate based on the KineticLaw and the species and parameters specified in the kinetic law object properties SpeciesVariableNames and ParameterVariableNames. SpeciesVariableNames are determined automatically for mass action kinetics.

If you update the reaction, the ReactionRate property automatically updates only for mass action kinetics. For all other kinetics, you must set the SpeciesVariableNames property of the kinetic law object.

For information on dimensional analysis for reaction rates, see "How Reaction Rates Are Evaluated".

Characteristics

Applies to	Object: reaction
Data type	Kinetic law object
Data values	Kinetic law object. Default is [] (empty).
Access	Read-only

Examples

Example with Henri-Michaelis-Menten kinetics

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

3 Verify that the `KineticLaw` property for the reaction object is `Henri-Michaelis-Menten`.

```
get (reactionObj, 'KineticLaw')
```

MATLAB returns:

```
SimBiology Kinetic Law Array

 Index:    KineticLawName:
   1       Henri-Michaelis-Menten
```

See Also

`KineticLawName`, `Parameters`, `ParameterVariableNames`, `ReactionRate`, `SpeciesVariableNames`

KineticLawName property

Purpose

Name of kinetic law applied to reaction

Description

The `KineticLawName` property of the kinetic law object indicates the name of the kinetic law definition applied to the reaction. `KineticLawName` can be any valid name from the built-in or user-defined kinetic law library. See "Kinetic Law Definition" on page 3-65 for more information.

You can find the `KineticLawName` list in the kinetic law library by using the command `sbiowhos -kineticlaw` (`sbiowhos`). You can create a kinetic law definition with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

Characteristics

Applies to	Object: kineticlaw
Data type	`char string`
Data values	`char string` specified by kinetic law definition
Access	Read-only

Examples

1. Create a model object, add a reaction object, and define a kinetic law for the reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

2. Verify the `KineticLawName` of `kineticlawObj`.

```
get (kineticlawObj, 'KineticLawName')
```

MATLAB returns:

```
ans =

Henri-Michaelis-Menten
```

See Also

`Expression(AbstractKineticLaw, KineticLaw)`, `Parameters`, `ParameterVariableNames`, `ParameterVariables`, `ReactionRate`, `sbioaddtolibrary`, `sbiowhos`, `SpeciesVariables`, `SpeciesVariableNames`

LagParameter property

Purpose

Parameter specifying time lag for doses

Description

`LagParameter` is a property of the `PKModelMap` object. It specifies the name(s) of parameter object(s) that represent the time lag(s) of doses associated with the `PKModelMap` object.

Specify the name(s) of parameter object(s) that are:

- Scoped to a model
- Constant, that is, their `ConstantValue` property is `true`

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the `LagParameter` property and the `DosingType` property. For a dose that has no lag, use `''` (an empty string). For an example, see "Dosing Multiple Compartments in a Model".

Characteristics

Applies to	Object: `PKModelMap`
Data type	`char string` or cell array of strings
Data values	Name(s) of parameter object(s) or empty. Default is an empty cell array. The parameter objects must be: • Scoped to a model • Constant, that is, have a `ConstantValue` property set to `true`.
Access	Read/write

Tip If you are not using any doses with time lags, you can set this property to a cell array of empty strings, or simply an empty cell array.

See Also

"Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" in the SimBiology User's Guide, `DosingType`, `PKModelMap object`

Purpose

Parameter specifying time lag for dose

Description

`LagParameterName` is a property of a `RepeatDose` or `ScheduleDose` object.

Specify the name of a parameter object that is:

- Scoped to a model
- Constant, that is, its `ConstantValue` property is `true`

The parameter specifies the length of time it takes for the dose to reach its target after being introduced.

Characteristics

Applies to	Objects: `RepeatDose`, `ScheduleDose`
Data type	`char string`
Data values	Name of a parameter object or empty. Default is an empty string. The parameter object must be: • Scoped to a model • Constant, that is, have a `ConstantValue` property set to `true`
Access	Read/write

See Also

`RepeatDose object`, `ScheduleDose object`

LogDecimation property

Purpose

Specify frequency to log stochastic simulation output

Description

`LogDecimation` is a property of the `SolverOptions` property, which is a property of a `configset` object. This property defines how often stochastic simulation data is recorded. `LogDecimation` is available only for stochastic solvers (`ssa`, `expltau`, and `impltau`).

Use `LogDecimation` to specify how frequently you want to record the output of the simulation. For example, if you set `LogDecimation` to `1`, for the command `[t,x] = sbiosimulate(modelObj)`, at each simulation step the time will be logged in `t` and the quantity of each logged species will be logged as a row in `x`. If `LogDecimation` is `10`, then every 10th simulation step will be logged in `t` and `x`.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`int`
Data values	>0. Default is `1`.
Access	Read/write

Examples

This example shows how to change `LogDecimation` settings.

1. Retrieve the `configset` object from the `modelObj`, and change the `SolverType` to `expltau`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'expltau')
```

2. Change the `LogDecimation` to `10`.

```
set(configsetObj.SolverOptions, 'LogDecimation', 10);
get(configsetObj.SolverOptions, 'LogDecimation')

ans =
```

10

See Also `ErrorTolerance`, `RandomState`

MaximumNumberOfLogs property

Purpose

Maximum number of logs criteria to stop simulation

Description

`MaximumNumberOfLogs` is a property of a `Configset` object. This property sets the maximum number of logs criteria to stop a simulation.

A simulation stops when it meets any of the criteria specified by `StopTime`, `MaximumNumberOfLogs`, or `MaximumWallClock`. However, if you specify the `OutputTimes` property of the `SolverOptions` property of the `Configset` object, then `StopTime` and `MaximumNumberOfLogs` are ignored. Instead, the last value in `OutputTimes` is used as the `StopTime` criteria, and the length of `OutputTimes` is used as the `MaximumNumberOfLogs` criteria.

Characteristics

Applies to	Object: `Configset`
Data type	`double`
Data values	Positive value. Default is `Inf`.
Access	Read/write

Examples

Set Maximum Number of Logs Criteria to Stop Simulation

Set the maximum number of logs that triggers a simulation to stop.

Create a `model` object named `cell` and save it in a variable named `modelObj`.

```
modelObj  = sbiomodel('cell');
```

Retrieve the configuration set from `modelObj` and save it in a variable named `configsetObj`.

```
configsetObj = getconfigset(modelObj);
```

Configure the simulation stop criteria by setting the `MaximumNumberOfLogs` property to `50`. Leave the `StopTime` and `MaximumWallClock` properties at their default values of `10` seconds and `Inf`, respectively.

```
set(configsetObj, 'MaximumNumberOfLogs', 50)
```

View the properties of `configsetObj`.

```
get(configsetObj)

                         Active: 1
                 CompileOptions: [1x1 SimBiology.CompileOptions]
                           Name: 'default'
                          Notes: ''
                 RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
     SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOpt
                  SolverOptions: [1x1 SimBiology.ODESolverOptions]
                     SolverType: 'ode15s'
                       StopTime: 10
            MaximumNumberOfLogs: 50
               MaximumWallClock: Inf
                      TimeUnits: 'second'
                           Type: 'configset'
```

When you simulate `modelObj`, the simulation stops when 50 logs are created or when the simulation time reaches 10 seconds, whichever comes first.

See Also

`Configset object`, `MaximumWallClock`, `OutputTimes`, `StopTime`

MaximumWallClock property

Purpose

Maximum elapsed wall clock time to stop simulation

Description

`MaximumWallClock` is a property of a `Configset` object. This property sets the maximum elapsed wall clock time (seconds) criteria to stop a simulation.

A simulation stops when it meets any of the criteria specified by `StopTime`, `MaximumNumberOfLogs`, or `MaximumWallClock`. However, if you specify the `OutputTimes` property of the `SolverOptions` property of the `Configset` object, then `StopTime` and `MaximumNumberOfLogs` are ignored. Instead, the last value in `OutputTimes` is used as the `StopTime` criteria, and the length of `OutputTimes` is used as the `MaximumNumberOfLogs` criteria.

Characteristics

Applies to	Object: `Configset`
Data type	`double`
Data values	Positive scalar. Default is `Inf`.
Access	Read/write

Examples

Set Maximum Wall Clock Criteria to Stop Simulation

Set the maximum wall clock time (in seconds) that triggers a simulation to stop.

Create a `model` object named `cell` and save it in a variable named `modelObj`.

```
modelObj  = sbiomodel('cell');
```

Retrieve the configuration set from `modelObj` and save it in a variable named `configsetObj`.

```
configsetObj = getconfigset(modelObj);
```

Configure the simulation stop criteria by setting the `MaximumWallClock` property to `20` seconds. Leave the `StopTime` and `MaximumNumberOfLogs` properties at their default values of `10` seconds and `Inf`, respectively.

```
set(configsetObj, 'MaximumWallClock', 20)
```

View the properties of `configsetObj`.

```
get(configsetObj)
```

```
                     Active: 1
             CompileOptions: [1x1 SimBiology.CompileOptions]
                       Name: 'default'
                      Notes: ''
             RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
 SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOpt
              SolverOptions: [1x1 SimBiology.ODESolverOptions]
                 SolverType: 'ode15s'
                   StopTime: 10
        MaximumNumberOfLogs: Inf
           MaximumWallClock: 20
                  TimeUnits: 'second'
                       Type: 'configset'
```

When you simulate `modelObj`, the simulation stops when the simulation time reaches `10` seconds or the wall clock time reaches `20` seconds, whichever comes first.

See Also

`Configset object`, `MaximumNumberOfLogs`, `OutputTimes`, `StopTime`

MaxIterations property

Purpose

Specify nonlinear solver maximum iterations in implicit tau

Description

The MaxIterations property specifies the maximum number of iterations for the nonlinear solver in impltau. It is a property of the SolverOptions object. SolverOptions is a property of the configset object.

The implicit tau solver in SimBiology software internally uses a nonlinear solver to solve a set of algebraic nonlinear equations at every simulation step. Starting with an initial guess at the solution, the nonlinear solver iteratively tries to find the solution to the algebraic equations. The closer the initial guess is to the solution, the fewer the iterations the nonlinear solver will take before it finds a solution. MaxIterations specifies the maximum number of iterations the nonlinear solver should take before it issues a “failed to converge” error. If you get this error during simulation, try increasing MaxIterations. The default value of MaxIterations is 15.

Characteristics

Applies to	Object: SolverOptions
Data type	int
Data values	>0. Default is 15.
Access	Read/write

Examples

This example shows how to change MaxIterations settings.

1. Retrieve the configset object from the modelObj, and change the SolverType to impltau.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'impltau');
```

2. Change the MaxIterations to 25.

```
set(configsetObj.SolverOptions, 'MaxIterations', 25);
```

```
get(configsetObj.SolverOptions, 'MaxIterations')

ans =

  25
```

See Also ErrorTolerance, LogDecimation, RandomState

MaxStep property

Purpose

Specify upper bound on ODE solver step size

Description

`MaxStep` is a property of the `SolverOptions` property, which is a property of a `configset` object. This property specifies the bounds on the size of the time steps. `MaxStep` is available only for ODE solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`).

If the differential equation has periodic coefficients or solutions, it might be a good idea to set `MaxStep` to some fraction (such as 1/4) of the period. This guarantees that the solver does not enlarge the time step too much and step over a period of interest. For more information on `MaxStep`, see `odeset` in the MATLAB documentation.

Characteristics

Applies to	Object: `SolverOptions`
Data type	Positive scalar or empty
Data values	Default value is `[]` (empty), which is equivalent to setting `MaxStep` to infinity.
Access	Read/write

See Also

SimBiology property `RelativeTolerance`

MATLAB function `odeset`

ModelName property

Purpose

Name of model simulated

Description

The `ModelName` property shows the name of the model for which the SimData object contains the simulation data.

Characteristics

Applies to	Object: SimData
Data type	`string`
Data values	Default value is `''` (empty).
Access	Read-only

See Also

`Data`, `DataInfo`

Models property

Purpose

Contain all model objects

Description

The `Models` property shows the models in the SimBiology root. It is a read-only array of model objects.

SimBiology has a hierarchical organization. A model object has the SimBiology root as its `Parent`. Parameter objects can have a model object or kinetic law object as `Parent`. You can display all the component objects with `*modelObj*.Models` or `get (*modelObj*, 'Models')`.

Characteristics

Applies to	Objects: root
Data type	Array of model objects
Data values	Model object. Default is `[]` (empty).
Access	Read-only

See Also

`sbiomodel`

Purpose

Relationship between defined unit and base unit

Description

The `Multiplier` is the numerical value that defines the relationship between the unit `Name` and the base unit as a product of the `Multiplier` and the base unit. For example, in `Celsius = (5/9)*( Fahrenheit-32)`; `Multiplier` is `5/9` and `Offset` is `-32`. For `1 mole = 6.0221e23*molecule`, the `Multiplier` is `6.0221e23`.

Characteristics

Applies to	Object: Unit
Data type	`double`
Data values	Nonzero real number. Default value is 1.
Access	Read/write

Examples

This example shows how to create a user-defined unit, add it to the user-defined library, and query the library.

1. Create a user-defined unit called `usermole`, whose composition is `molecule` and `Multiplier` property is `6.0221e23`.

```
unitObj = sbiounit('usermole', 'molecule', 6.0221e23);
```

2. Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

3. Query the `Multiplier` property.

```
get(unitObj, 'Multiplier')

ans =

1/molarity*second
```

See Also

`Composition`, `get`, `Offset`, `sbiounit`, `set`

Name property

Purpose

Specify name of object

Description

The `Name` property identifies a SimBiology object. Compartments, species, parameters, and model objects can be referenced by other objects using the `Name` property, therefore `Name` must be unique for these objects. However, species names need only be unique within each compartment. Parameter names must be unique within a model (if at the model level), or within each kinetic law (if at the kinetic law level). This means that you can have nonunique species names if the species are in different compartments, and nonunique parameter names if the parameters are in different kinetic laws or at different levels. Note that having nonunique parameter names can cause the model to have shadowed parameters and that may not be best modeling practice. For more information on levels of parameters, see "Scope of Parameter Objects".

Use the function `sbioselect` to find an object with the same `Name` property value.

In addition, note the following constraints and reserved characters for the `Name` property in objects:

- Model and parameter names cannot be empty, the word `time`, all whitespace, or contain the characters `[` or `]`.
- Compartment and species names cannot be empty, the word `null`, the word `time` or contain the characters `->`, `<->`, `[` or `]`.
  - However, compartment and species names can contain the words `null` and `time` within the name, such as `nulldrug` or `nullreceptor`.
- Reaction, event, and rule names cannot be the word `time` or contain the characters `[` or `]`.
- If you have a parameter, a species, or compartment name that is not a valid MATLAB variable name, when you write an event function, an event trigger, a reaction, reaction rate equation, or a rule you must enclose that name in brackets. For example, enclose `[DNA polymerase+]` in brackets. In addition, if you have the same species

in multiple compartments you must qualify the species with the compartment name, for example, `nucleus.[DNA polymerase+]`, `[nuclear complex].[DNA polymerase+]`.

For more information on valid MATLAB variable names, see `genvarname` and `isvarname`.

Characteristics

Applies to	Objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, unit, or variant
Data type	`char string`
Data values	Any `char string` except reserved words and characters
Access	Read/write

Examples

1. Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');
```

2. Add a reaction object to the model object. Note the use of brackets because the names are not valid MATLAB variable names.

```
reactionObj = addreaction(modelObj, '[Aspartic acid] -> [beta-Aspartyl-PO4]')
```

MATLAB returns:

```
SimBiology Reaction Array

Index:    Reaction:
  1       [Aspartic acid] -> [beta-Aspartyl-PO4]
```

3. Set the reaction `Name` and verify.

```
set (reactionObj, 'Name', 'Aspartate kinase reaction');
```

```
get (reactionObj, 'Name')
```

MATLAB returns:

```
ans =

  Aspartate kinase reaction
```

See Also

addcompartment, addkineticlaw, addparameter, addreaction, addrule, addspecies, RepeatDose object, sbiomodel, sbiounit, sbiounitprefix, ScheduleDose object

Purpose

Specify normalization type for sensitivity analysis

Description

`Normalization` is a property of the `SensitivityAnalysisOptions` object. `SensitivityAnalysisOptions` is a property of the configuration set object. Use `Normalization` to specify the normalization for the computed sensitivities.

The following values let you specify the type of normalization. The examples show you how sensitivities of a species `x` with respect to a parameter `k` are calculated for each normalization type:

- `'None'` specifies no normalization.

  $$\frac{dx(t)}{dk}$$

- `'Half'` specifies normalization relative to the numerator (species quantity) only.

  $$\left(\frac{1}{x(t)}\right)\left(\frac{dx(t)}{dk}\right)$$

- `'Full'` specifies that the data should be made dimensionless.

  $$\left(\frac{k}{x(t)}\right)\left(\frac{dx(t)}{dk}\right)$$

Characteristics

Applies to	Object: `SensitivityAnalysisOptions`
Data type	`enum`
Data values	`'None'`, `'Half'`, `'Full'`. Default is `'None'`.
Access	Read/write

See Also

`Inputs`, `Outputs`, `SensitivityAnalysis`, `SensitivityAnalysisOptions`

Notes property

Purpose

HTML text describing SimBiology object

Description

Use the `Notes` property of an object to store comments about the object. You can include HTML tagging in the notes to render formatted text in the SimBiology desktop.

Characteristics

Applies to	Objects: compartment, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, unit, or unit prefix
Data type	`char string`
Data values	Any `char string`
Access	Read/write

Examples

1. Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2. Write notes for the model object.

```
set (modelObj, 'notes', '09/01/05 experimental data')
```

3. Verify the assignment.

```
get (modelObj, 'notes')
```

MATLAB returns:

```
ans =

09/01/05 experimental data
```

See Also

`addkineticlaw`, `addparameter`, `addreaction`, `addrule`, `addspecies`, `RepeatDose object`, `sbiomodel`, `sbiounit`, `sbiounitprefix`, `ScheduleDose object`

Purpose

Measured response object name

Description

`Observed` is a property of the `PKModelMap` object. It specifies the name(s) of one or more objects that represent the measured response (the response variable). Specify the name(s) of species or parameter object(s) that are scoped to a model.

Characteristics

Applies to	Object: `PKModelMap`
Data type	`char string` or cell array of strings
Data values	Name of a species or parameter object or empty. Default is an empty cell array.
Access	Read/write

See Also

"Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" in the SimBiology User's Guide, `Dosed`, `Estimated`, `PKModelMap object`

Offset property

Purpose

Unit composition modifier

Description

Note The `Offset` property is currently not supported.

The `Offset` is the numerical value by which the unit composition is modified from the base unit. For example, `Celsius = (5/9)*( Fahrenheit-32); Multiplier` is `5/9` and `Offset` is `-32`.

Characteristics

Applies to	Object: Unit
Data type	`double`
Data values	Real number. Default is `0`.
Access	Read/write

Examples

This example shows how to create a user-defined unit, add it to the user-defined library, and query the library.

1. Create a user-defined unit called `celsius2`, whose composition refers to `fahrenheit`, `Multiplier` property is `9/5`, and `Offset` property is `32`.

   ```
   unitObj = sbiounit('celsius2','fahrenheit',9/5,32);
   ```

2. Add the unit to the user-defined library.

   ```
   sbioaddtolibrary(unitObj);
   ```

3. Query the `Offset` property.

   ```
   get(unitObj, 'Offset')

   ans =
   ```

```
32
```

See Also

`Composition`, `get`, `Multiplier`, `sbioaddtolibrary`, `sbioshowunits`, `sbiounit`, `set`

Outputs property

Purpose

Specify species and parameter outputs for sensitivity analysis

Description

`Outputs` is a property of the `SensitivityAnalysisOptions` object. `SenstivityAnalysisOptions` is a property of the configuration set object.

Use `Outputs` to specify the species and parameters for which you want to compute sensitivities.

The SimBiology software calculates sensitivities with respect to the values of the parameters and the initial amounts of the species specified in the `Inputs` property. When you simulate a model with `SensitivityAnalysis` enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species and parameters specified in `Outputs`. For a description of the output, see the `SensitivityAnalysisOptions` property description.

Characteristics

Applies to	Object: `SensitivityAnalysisOptions`
Data type	Species or parameter object or array of objects **Note** If a species or parameter object is determined by a repeated assignment rule, then you cannot use it as an `Outputs` property.
Data values	Species or parameter object, or an array of objects. Default is `[]` (empty array).
Access	Read/write

Examples

This example shows how to set `Outputs` for sensitivity analysis.

1 Import the radio decay model from the SimBiology demos.

```
modelObj  = sbmlimport('radiodecay');
```

2 Retrieve the configuration set object from `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

3 Add a species to the `Outputs` property and display it. Use the `sbioselect` function to retrieve the species object from the model.

```
SimBiology Species Array

Index:  Compartment:  Name:  InitialAmount: InitialAmountUnits:
  1       unnamed      z        0              molecule
```

See Also

`Inputs`, `sbioselect`, `SensitivityAnalysis`, `SensitivityAnalysisOptions`

OutputTimes property

Purpose

Specify times to log deterministic simulation output

Description

`OutputTimes` is a property of the `SolverOptions` property, which is a property of a `Configset` object. This property specifies the times during a deterministic (ODE) simulation that data is recorded. Time units are specified by the `TimeUnits` property of the `Configset` object. `OutputTimes` is available only for ODE solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`).

If the criteria set in the `MaximumWallClock` property causes a simulation to stop before all time values in `OutputTimes` are reached, then no data is recorded for the latter time values.

The `OutputTimes` property can also control when a simulation stops:

- The last value in `OutputTimes` overrides the `StopTime` property as criteria for stopping a simulation.
- The length of `OutputTimes` overrides the `MaximumNumberOfLogs` property as criteria for stopping a simulation.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`
Data values	Vector of nonnegative, monotonically increasing values, or `[]`, an empty vector. Default is `[]`, which results in data being logged every time the simulation solver takes a step.
Access	Read/write

Examples

Specify Times to Log Deterministic Simulation Output

Specify the times during a deterministic (ODE) simulation that data is recorded.

Create a `model` object named `cell` and save it in a variable named `modelObj`.

```
modelObj  = sbiomodel('cell');
```

Retrieve the configuration set from `modelObj` and save it in a variable named `configsetObj`.

```
configsetObj = getconfigset(modelObj);
```

Specify to log output every second for the first `10` seconds of the simulation. Do this by setting the `OutputTimes` property of the `SolverOptions` property of `ConfigsetObj`.

```
set(configsetObj.SolverOptions, 'OutputTimes', [1:10])
get(configsetObj.SolverOptions, 'OutputTimes')
```

```
ans =

     1     2     3     4     5     6     7     8     9    10
```

When you simulate `modelObj`, output is logged every second for the first `10` seconds of the simulation. Also, the simulation stops after the 10th log.

See Also

`MaximumNumberOfLogs`, `MaximumWallClock`, `SolverOptions`, `StopTime`, `TimeUnits`

Owner property

Purpose

Owning compartment

Description

`Owner` shows you the SimBiology compartment object that owns the compartment object. In the compartment object, the `Owner` property shows you whether the compartment resides within another compartment. The `Compartments` property indicates whether other compartments reside within the compartment. You can add a compartment object using the method `addcompartment`.

Characteristics

Applies to	Object: compartment
Data type	`char string`
Data values	Name of compartment object. Default is `[]`.
Access	Read-only

Examples

1 Create a model object named `modelObj`.

```
modelObj = sbiomodel('cell');
```

2 Add two compartments to the model object.

```
compartmentObj1 = addcompartment(modelObj, 'nucleus');
compartmentObj2 = addcompartment(modelObj, 'mitochondrion');
```

3 Add a compartment to one of the compartment objects.

```
compartmentObj3 = addcompartment(compartmentObj2, 'matrix');
```

4 Display the `Owner` property in the compartment objects.

```
get(compartmentObj3, 'Owner')
```

The result shows you the owning compartment and its components:

```
SimBiology Compartment - mitochondrion
```

```
Compartment Components:
  Capacity:          1
  CapacityUnits:
  Compartments:      1
  ConstantCapacity:  true
  Owner:
  Species:           0
```

5 Change the owning compartment.

```
set(compartmentObj3, 'Owner', compartmentObj1)
```

See Also

`Compartments`, `Parent`

ParameterNames (CovariateModel) property

Purpose

Names of parameters in `CovariateModel` object

Description

The `ParameterNames` property is a cell array of strings specifying the names of the parameters in the `Expression` property of a `CovariateModel` object.

Characteristics

Applies to	Object: `CovariateModel`
Data type	Cell array of strings
Data values	Names of the parameters in the `Expression` property
Access	Read only

See Also

`CovariateModel` | `Expression`

Purpose

Array of parameter objects

Description

The Parameters property indicates the parameters in a Model or KineticLaw object. Read-only array of Parameter objects.

The scope of a parameter object is hierarchical and is defined by the parameter's parent. If a parameter is defined with a kinetic law object as its parent, then only the kinetic law object can use the parameter. If a parameter object is defined with a model object as its parent, then components such as rules, events, and kinetic laws (reaction rate equations) can use the parameter.

You can add a parameter to a model object, or kinetic law object with the method addparameter and delete it with the method delete.

You can view parameter object properties with the get command and configure properties with the set command.

Characteristics

Applies to	Objects: model, kineticlaw
Data type	Array of parameter objects
Data values	Parameter objects. Default value is [] (empty).
Access	Read-only

Examples

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2. Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

3. Add a parameter and assign it to the kinetic law object (kineticlawObj);.

```
parameterObj1 = addparameter (kineticlawObj, 'K1');
```

```
get (kineticlawObj, 'Parameters')

SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1         K1       1
```

4 Add a parameter and assign it to the model object (`modelObj`);.

```
parameterObj1 = addparameter(modelObj, 'K2');
get(modelObj, 'Parameters')
SimBiology Parameter Array

 Index:    Name:    Value:    ValueUnits:
   1         K2       1
```

See Also

`addparameter`, `delete`, `get`, `set`

Purpose

Cell array of reaction rate parameters

Description

The ParameterVariableNames property shows the parameters used by the kinetic law object to determine the ReactionRate equation in the reaction object. Use setparameter to assign ParameterVariableNames. When you assign species to ParameterVariableNames, SimBiology software maps these parameter names to ParameterVariables in the kinetic law object.

If the reaction is using a kinetic law, the ReactionRate property of a reaction object shows the result of a mapping from a "Kinetic Law Definition" on page 3-65. The ReactionRate is determined by the kinetic law object Expression property by mapping ParameterVariableNames to ParameterVariables and SpeciesVariableNames to SpeciesVariables.

Characteristics

Applies to	Object: kineticlaw
Data type	Cell array of strings
Data values	Cell array of parameters
Access	Read/write

Examples

Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj KineticLaw property is configured to kineticlawObj.

3. The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) that should to be set. To set these variables:

```
setparameter(kineticlawObj,'Vm', 'Va');
setparameter(kineticlawObj,'Km', 'Ka');
```

4. Verify that the parameter variables are correct.

```
get (kineticlawObj, 'ParameterVariableNames')
```

MATLAB returns:

```
ans =

    'Va'    'Ka'
```

See Also

`Expression(AbstractKineticLaw, KineticLaw)`, `ParameterVariables`, `ReactionRate`, `setparameter`, `SpeciesVariables`, `SpeciesVariableNames`

Purpose

Parameters in kinetic law definition

Description

The `ParameterVariables` property shows the parameter variables that are used in the `Expression` property of the abstract kinetic law object. Use this property to specify the parameters in the `ReactionRate` equation. Use the method `set` to assign `ParameterVariables` to a kinetic law definition. For more information, see "Kinetic Law Definition" on page 3-65.

Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Specified by kinetic law definition
Access	Read/write in kinetic law definition. Read-only in kinetic law.

Examples

Create a model, add a reaction, and assign the `SpeciesVariableNames` for the reaction rate equation.

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2. Create a kinetic law object for the reaction object of the type `'Henri-Michaelis-Menten'` .

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

   `reactionObj KineticLaw` property is configured to `kineticlawObj`.

3. The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables. To set these variables:

```
get (kineticlawObj, 'ParameterVariables')
```

MATLAB returns:

```
ans =

    'Vm' 'Km'
```

See Also

`Expression(AbstractKineticLaw, KineticLaw)`, `ParameterVariableNames`, `ReactionRate`, `set`, `setparameter`, `SpeciesVariables`, `SpeciesVariableNames`

Purpose

Indicate parent object

Description

The `Parent` property indicates the parent object for a SimBiology object (read-only). The `Parent` property indicates accessibility of the object. The object is accessible to the `Parent` object and other objects within the `Parent` object. The value of `Parent` depends on the type of object and how it was created. All models always have the SimBiology root as the `Parent`.

More Information

The following table shows you the different objects and the possible `Parent` value.

Object	Parent
abstract kinetic law	• `[]` (empty) until added to library • root object upon addition to library
compartment	model object
event	model object or `[]` (empty)
kinetic law	reaction object
model	root object
parameter	model object, kinetic law object, or `[]` (empty)
reaction	model object or `[]` (empty)
RepeatDose	model object or `[]` (empty)
rule	model object or `[]` (empty)
ScheduleDose	model object or `[]` (empty)
species	compartment

Object	Parent
variant	model object or [] (empty)
unit and unit prefixes	• [] (empty) until added to library • root object upon addition to library

Characteristics

Applies to	Objects: abstract kinetic law, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, variant, unit, or unit prefix
Data type	Object
Data values	SimBiology component object or [] (empty)
Access	Read-only

See Also

addkineticlaw, addparameter, addreaction, RepeatDose object, sbiomodel, ScheduleDose object

Purpose

Hold compartments in PK model

Description

PKCompartments is a property of the PKModelDesign object. It is used to specify the compartments in the PKModelDesign object. Each compartment is a PKCompartment object added using the addCompartment method.

Characteristics

Applies to	Objects: PKModelDesign
Data type	object
Data values	PKCompartment object
Access	Read-only

See Also

"Creating Pharmacokinetic Models" in the SimBiology User's Guide, addCompartment, PKCompartment object, PKModelDesign object

Products property

Purpose

Array of reaction products

Description

The `Products` property contains an array of `SimBiology.Species` objects.

`Products` is a `1-by-n` species object array that indicates the species that are changed by the reaction. If the `Reaction` property is modified to use a different species, the `Products` property is updated accordingly.

You can add product species to the reaction with `addproduct` function. You can remove product species from the reaction with `rmproduct`. You can also update reaction products by setting the `Reaction` property with the function `set`.

Characteristics

Applies to	Object: reaction
Data type	Array of objects
Data values	Species objects. Default is `[]` (empty).
Access	Read-only

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add reaction objects.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

3 Verify the assignment.

```
productsObj = get(reactionObj, 'Products')
```

MATLAB returns:

```
SimBiology Species Array

Index:  Compartment:  Name:  InitialAmount: InitialAmountUnits:
```

```
1        unnamed        c        0
2        unnamed        d        0
```

See Also

addkineticlaw, addproduct, addspecies, rmproduct

RandomEffectNames (CovariateModel) property

Purpose

Names of random effects in `CovariateModel` object

Description

The `RandomEffectNames` property is a cell array of strings specifying the names of the random effects in the `Expression` property of a `CovariateModel` object. Names of random effects are denoted with the prefix `eta`.

Characteristics

Applies to	Object: `CovariateModel`
Data type	Cell array of strings
Data values	Names of the random effects in the `Expression` property. These name are denoted with the prefix `eta`.
Access	Read only

See Also

`CovariateModel` | `Expression`

Purpose

Set random number generator

Description

The `RandomState` property sets the random number generator for the stochastic solvers. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object.

SimBiology software uses a pseudorandom number generator. The sequence of numbers generated is determined by the state of the generator, which can be specified by the integer `RandomState`. If `RandomState` is set to integer `J`, the random number generator is initialized to its Jth state. The random number generator can generate all the floating-point numbers in the closed interval `[2^(-53), 1-2^(-53)]`. Theoretically, it can generate over `2^1492` values before repeating itself. But for a given state, the sequence of numbers generated will be the same. To change the sequence, change `RandomState`. SimBiology software resets the state at startup. The default value of `RandomState` is `[]`.

Characteristics

Applies to	Objects: `SolverOptions` for `SSA, expltau, impltau`
Data type	`int`
Data values	Default is `[]` (empty).
Access	Read/write

Examples

This example shows how to change `RandomState` settings.

1. Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `expltau`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
set(configsetObj, 'SolverType', 'expltau')
```

2. Change the `Randomstate` to `5`.

```
set(configsetObj.SolverOptions, 'RandomState', 5);
get(configsetObj.SolverOptions, 'RandomState'))


ans =

  5
```

See Also ErrorTolerance, LogDecimation, MaxIterations

Rate property

Purpose

Rate of dose

Description

`Rate` is a property of a `RepeatDose` or `ScheduleDose` object.

This property defines how fast a dose is given.

Note If you set the `Rate` property of a dose, you must also specify the `Amount` property of the dose, and set the `DurationParameterName` property to `''`. This is because the duration is calculated from the amount and rate.

Characteristics

Applies to	Objects: `RepeatDose`, `ScheduleDose`
Data type	`double` (`RepeatDose`) or `double array` (`ScheduleDose`)
Data values	Nonnegative real number. Default is `0` (`RepeatDose`) or `[]` (`ScheduleDose`).
Access	Read/write

See Also

`RepeatDose object`, `ScheduleDose object`

RateUnits property

Purpose

Units for dose rate

Description

`RateUnits` is a property of a `PKData`, `RepeatDose` or `ScheduleDose` object.

- In `RepeatDose` or `ScheduleDose` objects, this property defines units for the `Rate` property.
- In `PKData` object, this property defines units for the `RateLabel` property.

Characteristics

Applies to	Object: `RepeatDose`, `ScheduleDose`, `PKData`
Data type	string
Data values	Units from library with dimensions of amount divided by time. You cannot use units of concentration divided by time. Default = ” (empty).
Access	Read/write

See Also

`PKData object`, `ScheduleDose object`, `RepeatDose object`, `Rate`, `RateLabel`

Purpose

Rate of infusion column in data set

Description

`RateLabel` is a property of the `PKData` object. It specifies the column in DataSet that contains the rate of infusion. This applies only when dosing type is infusion. The data set must contain the rate and not an infusion time. The values must be positive and the column cannot contain `Inf` or `Inf`. `0` specifies an infinite rate (equivalent to a bolus dose), and `NaN` specifies no rate.

Characteristics

Applies to	Objects: `PKData`
Data type	`char string`
Data values	Column header string
Access	Read/write

See Also

"Specifying and Classifying the Data to Fit" in the SimBiology User's Guide, `PKData object`, `DosingType`

Reactants property

Purpose

Array of reaction reactants

Description

The `Reactants` property is a `1-by-n` species object array of reactants in the reaction. If the `Reaction` property is modified to use a different reactant, the `Reactants` property will be updated accordingly.

You can add reactant species to the reaction with the `addreactant` method.

You can remove reactant species from the reaction with the `rmreactant method`. You can also update reactants by setting the `Reaction` property with the function `set`.

Characteristics

Applies to	Object: reaction
Data type	Species object or array of species objects
Data values	Species objects. Default is `[]` (empty).
Access	Read-only

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add reaction objects.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

3 View the reactants for `reactionObj`.

```
get(reactionObj, 'Reactants')
```

MATLAB returns:

```
SimBiology Species Array

Index:  Compartment:  Name:  InitialAmount: InitialAmountUnits:
 1       unnamed        a        0
```

```
2        unnamed        b        0
```

See Also

addreactant, addreaction, addspecies, rmreactant

Reaction property

Purpose

Reaction object reaction

Description

Property to indicate the reaction represented in the reaction object. Indicates the chemical reaction that can change the amount of one or more species, for example, `'A + B  > C'`. This property is different from the model object property called `Reactions`.

See `addreaction` for more information on how the `Reaction` property is set.

Characteristics

Applies to	Object: reaction
Data type	`char string`
Data values	Valid reaction string. Default is `''` (empty).
Access	Read/write

Examples

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Verify that the reaction property records the input.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =

a + b -> c + d
```

See Also

`addreaction`

Purpose

Reaction rate equation in reaction object

Description

The ReactionRate property defines the reaction rate equation. You can define a ReactionRate with or without the KineticLaw property. KineticLaw defines the type of reaction rate. The addkineticlaw function configures the ReactionRate based on the KineticLaw and the species and parameters specified in the kinetic law object properties SpeciesVariableNames and ParameterVariableNames.

The reaction takes place in the reverse direction if the Reversible property is true. This is reflected in ReactionRate. The ReactionRate includes the forward and reverse rate if reversible.

You can specify ReactionRate without KineticLaw. Use the set function to specify the reaction rate equation. SimBiology software adds species variables while creating reactionObj using the addreaction method. You must add the parameter variables (to the modelObj in this case). See the example below.

After you specify the ReactionRate without KineticLaw and you later configure the reactionObj to use KineticLaw, the ReactionRate is unset until you specify SpeciesVariableNames and ParameterVariableNames.

For information on dimensional analysis for reaction rates, see "How Reaction Rates Are Evaluated" .

Note If you set the ReactionRate property to an expression that is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" before simulating your model.

Characteristics

Applies to	Object: reaction
Data type	char string

Data values	Reaction rate string. Default is `''` (empty).
Access	Read/write

Examples

Example 1

Create a model, add a reaction, and assign the expression for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

`reactionObj` KineticLaw property is configured to `kineticlawObj`.

3 The `'Henri-Michaelis-Menten'` kinetic law has two parameter variables (`Vm` and `Km`) and one species variable (`S`) that you should set. To set these variables, first create the parameter variables as parameter objects (`parameterObj1, parameterObj2`) with names `Vm_d` and `Km_d` and assign them to `kineticlawObj`.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

4 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Vm_d' 'Km_d'});
set(kineticlawObj,'SpeciesVariableNames', {'a'});
```

5 Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Vm_d*a/(Km_d + a)
```

Example 2

Create a model, add a reaction, and specify `ReactionRate` without a kinetic law.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a + b -> c + d');
```

2 Specify `ReactionRate` and verify the assignment.

```
set (reactionObj, 'ReactionRate', 'k*a');
get(reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

k*a
```

3 You cannot simulate the model until you add the parameter `k` to the `modelObj`.

```
parameterObj = addparameter(modelObj, 'k');
```

SimBiology adds the parameter to the `modelObj` with default `Value` = `1.0` for the parameter.

See Also

`addparameter`, `addreaction`, `Reversible`

Reactions property

Purpose

Array of reaction objects

Description

Property to indicate the reactions in a `Model` object. Read-only array of reaction objects.

A reaction object defines a chemical reaction that occurs between species. The species for the reaction are defined in the `Model` object property `Species`.

You can add a reaction to a model object with the method `addreaction`, and you can remove a reaction from the model object with the method `delete`.

Characteristics

Applies to	Object: model
Data type	Array of reaction objects
Data values	Reaction object
Access	Read-only

Examples

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Verify that the reactions property records the input.

```
get (modelObj, 'Reactions')
```

MATLAB returns:

```
SimBiology Reaction Array

 Index:    Reaction:
   1       a + b -> c + d
```

See Also

`addreaction`, `delete`

Purpose

Allowable error tolerance relative to state value during a simulation

Description

`RelativeTolerance` is a property of the `SolverOptions` object, which is a property of a `Configset` object. It is available for the ode solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`).

The `RelativeTolerance` property specifies the allowable error tolerance relative to the state vector at each simulation step. The state vector contains values for all the state variables, for example, amounts for all the species.

If you set the `RelativeTolerance` at `1e-2`, you are specifying that an error of 1% relative to each state value is acceptable at each simulation step.

Algorithm

At each simulation step, the solver estimates the local error e_i in the `ith` state vector `y`. Simulation converges at that time step if e_i satisfies the following equation:

$$|e_i| \ \max(\text{RelativeTolerance}*|y_i|,\text{AbsoluteTolerance})$$

Thus at higher state values, convergence is determined by `RelativeTolerance`. As the state values approach zero, convergence is controlled by `AbsoluteTolerance`. The choice of values for `RelativeTolerance` and `AbsoluteTolerance` will vary depending on the problem. The default values should work for first trials of the simulation; however if you want to optimize the solution, consider that there is a trade-off between speed and accuracy. If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy. If the results appear to be inaccurate, you can decrease the tolerance values but this will slow down the solver. If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

Characteristics

Applies to	Object: `SolverOptions`
Data type	`double`

Data values	Positive scalar that is <1. Default is `1e-3`.
Access	Read/write

Examples

This example shows how to change `AbsoluteTolerance`.

1 Retrieve the `configset` object from the `modelObj`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj)
```

2 Change the `AbsoluteTolerance` to `1e-8`.

```
set(configsetObj.SolverOptions, 'RelativeTolerance', 1.0e-6);
get(configsetObj.SolverOptions, 'RelativeTolerance')

ans =

  1.0000e-006
```

See Also

`AbsoluteTolerance`

Purpose

Dose repetitions

Description

`RepeatCount` is a property of a `RepeatDose` object. This property defines the number of doses after the initial dose in a repeat dose series.

Note When the `Interval` property is `0`, `RepeatDose` ignores the `RepeatCount` property, that is, it treats it as though it is set to `0`.

Characteristics

Applies to	Object: `RepeatDose`
Data type	`double`
Data values	Nonnegative integer. Default is `0`
Access	Read/Write

See Also

`ScheduleDose object` and `RepeatDose object`

Reversible property

Purpose

Specify whether reaction is reversible or irreversible

Description

The `Reversible` property defines whether a reaction is reversible or irreversible. The rate of the reaction is defined by the `ReactionRate` property. For a reversible reaction, the reaction rate equation is the sum of the rate of the forward and reverse reactions. The type of reaction rate is defined by the `KineticLaw` property. If a reaction is changed from reversible to irreversible or vice versa after `KineticLaw` is assigned, the new `ReactionRate` is determined only if `Type` is `MassAction`. All other `Types` result in unchanged `ReactionRate`. For `MassAction`, the first parameter specified is assumed to be the rate of the forward reaction.

Characteristics

Applies to	Object: reaction
Data type	`boolean`
Data values	`true`, `false`. Default value is `false`.
Access	Read/write

Examples

Create a model, add a reaction, and assign the expression for the reaction rate equation.

1. Create model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2. Set the `Reversible` property for the `reactionObj` to `true` and verify this setting.

```
set (reactionObj, 'Reversible', true)
get (reactionObj, 'Reversible')
```

MATLAB returns:

```
ans =
```

```
1
```

MATLAB returns 1 for true and 0 for false.

In the next steps the example illustrates how the reaction rate equation is assigned for reversible reactions.

3 Create a kinetic law object for the reaction object of the type 'MassAction'.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

reactionObj KineticLaw property is configured to kineticlawObj.

4 The 'MassAction' kinetic law for reversible reactions has two parameter variables ('Forward Rate Parameter' and 'Reverse Rate Parameter') that you should set. The species variables for MassAction are automatically determined. To set the parameter variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) named Kf and Kr and assign the object to kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Kf');
parameterObj2 = addparameter(kineticlawObj, 'Kr');
```

5 Set the variable names for the kinetic law object.

```
set(kineticlawObj,'ParameterVariableNames', {'Kf' 'Kr'});
```

6 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =

Kf*a*b - Kr*c*d
```

See Also

`addparameter`, `addreactant`, `addreaction`, `ParameterVariableNames`, `ReactionRate`

Purpose

Specify species and parameter interactions

Description

The `Rule` property contains a rule that defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species `A` and species `B` must be some value. `Rule` is a MATLAB expression that defines the change in the species object quantity or a parameter object `Value` when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule` and `RuleType`.

Note If you set the `Rule` property for an algebraic rule, rate rule, or repeated assignment rule, and the rule expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" before simulating your model.

Characteristics

Applies to	Object: rule
Data type	`char string`
Data values	`char string` defined as species or parameter objects. Default is empty.
Access	Read/write

Examples

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Add a rule.

```
ruleObj = addrule(modelObj, '10-a+b')
```

MATLAB returns:

```
SimBiology Rule Array

Index:    RuleType:    Rule:
1         algebraic    10-a+b
```

See Also

addrule, delete

Purpose

Specify type of rule for rule object

Description

The `RuleType` property indicates the type of rule defined by the rule object. A `Rule` object defines how certain species, parameters, and compartments should interact with one another. For example, a rule could state that the total number of species A and species B must be some value. `Rule` is a MATLAB expression that defines the change in the species object quantity or a parameter object `Value` when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule`.

The types of rules in SimBiology are as follows:

- `initialAssignment` — Lets you specify the initial value of a parameter, species, or compartment capacity, as a function of other model component values in the model.
- `repeatedAssignment` — Lets you specify a value that holds at all times during simulation, and is a function of other model component values in the model.
- `algebraic` — Lets you specify mathematical constraints on one or more parameters, species, or compartments that must hold during a simulation.
- `rate` — Lets you specify the time derivative of a parameter value, species amount, or compartment capacity.

Constraints on Varying Species Using a Rate Rule

If the model has a species defined in concentration, being varied by a `rate` rule, and it is in a compartment with varying volume, you can only use `rate` or `initialAssignment` rules to vary the compartment volume.

Conversely, if you are varying a compartment's volume using a `repeatedAssignment` or `algebraic` rules, then you cannot vary a species (defined in concentration) within that compartment, with a `rate` rule.

RuleType property

The reason for these constraints is that, if a species is defined in concentration and it is in a compartment with varying volume, the time derivative of that species is a function of the compartment's rate of change. For compartments varied by rate rules, the solver has that information.

Note that if you specify the species in amounts there are no constraints.

Characteristics

Applies to	Object: rule
Data type	`char string`
Data values	`'initialAssignment'`, `'repeatedAssignment''algebraic'`, `'rate'`. Default value is `'initialAssignment'`.
Access	Read/write

Examples

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a -> b');
```

2 Add a rule that specifies the quantity of a species `c`. In the rule expression, `k` is the rate constant for `a -> b`.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

3 Change the `RuleType` from the default (`'algebraic'`) to `'rate'` and verify it using the `get` command.

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
    Active: 1
Annotation: ''
      Name: ''
```

```
     Notes: ''
    Parent: [1x1 SimBiology.Model]
      Rule: 'c = k*(a+b)'
  RuleType: 'rate'
       Tag: ''
      Type: 'rule'
  UserData: []
```

See Also

"Rule Object" in the *SimBiology User's Guide*, `addrule`, `delete`

Rules property

Purpose

Array of rules in model object

Description

The `Rules` property shows the rules in a `Model` object. Read-only array of `SimBiology.Rule` objects.

A *rule* is a mathematical expression that modifies a species amount or a parameter value. A rule defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species `A` and species `B` must be some value.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule` and `RuleType`.

Characteristics

Applies to	Object: model
Data type	Array of rule objects
Data values	Rule object
Access	Read-only

Examples

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

2 Add a rule.

```
ruleobj = addrule(modelObj, '10-a+b')
```

MATLAB returns:

```
SimBiology Rule Array

Index:    RuleType:    Rule:
 1        algebraic    10-a+b
```

See Also

`addrule`, `delete`

Purpose

Information about simulation

Description

The `RunInfo` property contains information describing the simulation run that yielded the data in the SimData object.

The following information is stored:

- `Configset` — A `struct` form of the configuration set used during simulation. This would typically be the model's active configset.
- `Variant` — A `struct` form of the variant(s) used during simulation.
- `SimulationDate` — The date/time of simulation.
- `SimulationType` — Either `'single run'` or `'ensemble run'`, depending on whether the data object was created using the function `sbiosimulate` or the function `sbioensemblerun`.

Characteristics

Applies to	Object: SimData
Data type	`struct`

Data values	Default values are as follows:
	
	`ConfigSet: []`
	`SimulationDate: ''`
	`SimulationType: ''`
	`Variant: []`
	
	In practice, the `ConfigSet`, `SimulationDate`, and `SimulationType` fields are rarely empty, since they are populated after simulation.

Access	Read-only

See Also

`StopTime`, `StopTimeType`

RuntimeOptions property

Purpose

Options for logged species

Description

The `RuntimeOptions` property holds options for species that will be logged during the simulation run. The run-time options object can be accessed through this property.

The `LogDecimation` property of the configuration set object defines how often data is logged.

Property Summary

`StatesToLog`	Specify species, compartment, or parameter data recorded
`Type`	Display SimBiology object type

Characteristics

Applies to	Object: `configset`
Data type	Object
Data values	Run-time options
Access	Read-only

Examples

1 Create a model object, and retrieve its configuration set.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2 Retrieve the `RuntimeOptions` object from the `configset` object.

```
runtimeObj = get(configsetObj, 'RunTimeOptions')
Runtime Settings:

     StatesToLog:         all
```

See Also

`get`, `set`

Purpose

Enable or disable sensitivity analysis

Description

`SensitivityAnalysis` is a property of the `SolverOptions` property, which is a property of a `configset` object. This property lets you compute the time-dependent sensitivities of all the species states defined by the `StatesToLog` property with respect to the `Inputs` that you specify in the `SensitivityAnalysisOptions` property of the configuration set object.

`SensitivityAnalysis` is available only for the ODE solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`)

Note Models containing the following active components do not support sensitivity analysis:

- Nonconstant compartments
- Algebraic rules
- Events

For more information on setting up sensitivity analysis, see `SensitivityAnalysisOptions` . For a description of sensitivity analysis calculations, see "Sensitivity Calculation".

Characteristics

Applies to	Object: `SolverOptions`
Data type	`logical`
Data values	`1`, `0`, `true`, `false`. Default is `false`.
Access	Read/write

SensitivityAnalysis property

Examples

This example shows how to enable `SensitivityAnalysis`.

1. Retrieve the `configset` object from the `modelObj`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2. Enable `SensitivityAnalysis`.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true);
get(configsetObj.SolverOptions, 'SensitivityAnalysis')

ans =

  on
```

See Also

`SensitivityAnalysisOptions`, `SolverOptions`, `SolverType`, `StatesToLog`

Purpose

Specify sensitivity analysis options

Description

The `SensitivityAnalysisOptions` property is an object that holds the sensitivity analysis options in the configuration set object. Sensitivity analysis is supported only for deterministic (ODE) simulations.

Note The `SensitivityAnalysisOptions` property controls the settings related to sensitivity analysis. To enable or disable sensitivity analysis, use the `SensitivityAnalysis` property.

Properties of `SensitivityAnalysisOptions` are summarized in "Property Summary" on page 3-160.

When sensitivity analysis is enabled, the following command

```
[t,x,names] = sbiosimulate(modelObj)
```

returns `[t,x,names]`, where

- `t` is an `n-by-1` vector, where `n` is the number of steps taken by the ode solver and `t` defines the time steps of the solver.
- `x` is an `n-by-m` matrix, where `n` is the number of steps taken by the ode solver and `m` is:

  ```
  Number of species and parameters specified in StatesToLog +
  (Number of sensitivity outputs * Number of sensitivity input factors)
  ```

  A SimBiology state includes species and nonconstant parameters.
- `names` is the list of states logged and the list of sensitivities of the species specified in `StatesToLog` with respect to the input factors.

For an example of the output, see "Examples" on page 3-160.

You can add a number of configuration set objects with different `SensitivityAnalysisOptions` to the model object with the

`addconfigset` method. Only one configuration set object in the model object can have the `Active` property set to `true` at any given time.

Property Summary

`Inputs`	Specify species and parameter input factors for sensitivity analysis
`Normalization`	Specify normalization type for sensitivity analysis
`Outputs`	Specify species and parameter outputs for sensitivity analysis

Characteristics

Applies to	Object: configuration set
Data type	Object
Data values	`SensitivityAnalysisOptions` properties as summarized in "Property Summary" on page 3-160.
Access	Read-only

Examples

This example shows how to set `SensitivityAnalysisOptions`.

1 Import the radio decay model from SimBiology demos.

```
modelObj  = sbmlimport('radiodecay');
```

2 Retrieve the configuration settings and the sensitivity analysis options from `modelObj`.

```
configsetObj = getconfigset(modelObj);
sensitivityObj = get(configsetObj, 'SensitivityAnalysisOptions');
```

3 Add a species and a parameter to the `Inputs` property. Use the `sbioselect` function to retrieve the species and parameter objects from the model.

4 Add a species to the `Outputs` property and display.

```
SimBiology Species Array

   Index:    Compartment:    Name:    InitialAmount:    InitialAmou
   1         unnamed         z        0                 molecule
```

5 Enable `SensitivityAnalysis`.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true);
get(configsetObj.SolverOptions, 'SensitivityAnalysis')

ans =

  1
```

6 Simulate and return the results to three output variables. See "Description" on page 3-159 for more information.

```
[t,x,names] = sbiosimulate(modelObj);
```

7 Display the `names`.

```
names

names =

    'x'
    'z'
    'd[z]/d[z]_0'
    'd[z]/d[Reaction1.c]'
```

Display state values `x`.

```
x
```

The display follows the column order shown in `names` for the values in `x`. The rows correspond to `t`.

See Also `addconfigset`, `getconfigset`, `SensitivityAnalysis`

Purpose

Specify model solver options

Description

The `SolverOptions` property is an object that holds the model solver options in the `configset` object. Changing the property `SolverType` changes the options specified in the `SolverOptions` object.

Properties of `SolverOptions` are summarized in "Property Summary" on page 3-163.

Property Summary

`AbsoluteTolerance`	Absolute error tolerance applied to state value during simulation
`AbsoluteToleranceScaling`	Control scaling of absolute error tolerance during simulation
`AbsoluteToleranceStepSize`	Initial guess for time step size for scaling of absolute error tolerance
`ErrorTolerance`	Specify explicit or implicit tau error tolerance
`LogDecimation`	Specify frequency to log stochastic simulation output
`MaxIterations`	Specify nonlinear solver maximum iterations in implicit tau
`MaxStep`	Specify upper bound on ODE solver step size
`OutputTimes`	Specify times to log deterministic simulation output
`RandomState`	Set random number generator
`RelativeTolerance`	Allowable error tolerance relative to state value during a simulation

SensitivityAnalysis	Enable or disable sensitivity analysis
Type	Display SimBiology object type

Characteristics

Applies to	Object: configset
Data type	Object
Data values	Solver options depending on SolverType. Default is SolverOptions for default SolverType (ode15s).
Access	Read-only

Examples

This example shows the changes in SolverOptions for various SolverType settings.

1. Retrieve the configset object from the modelObj.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2. Configure the SolverType to ode45.

```
set(configsetObj, 'SolverType', 'ode45');
get(configsetObj, 'SolverOptions')


Solver Settings: (ode)

     AbsoluteTolerance:    1.000000e-006
     RelativeTolerance:    1.000000e-003
```

3. Configure the SolverType to ssa.

```
set(configsetObj, 'SolverType', 'ssa');
get(configsetObj, 'SolverOptions')
```

```
Solver Settings: (ssa)

     LogDecimation:        1
     RandomState:          []
```

4 Configure the `SolverType` to `impltau`.

```
set(configsetObj, 'SolverType', 'impltau');
get(configsetObj, 'SolverOptions')


Solver Settings: (impltau)

     ErrorTolerance:       3.000000e-002
     LogDecimation:        1
     AbsoluteTolerance:    1.000000e-002
     RelativeTolerance:    1.000000e-002
     MaxIterations:        15
     RandomState:          []
```

5 Configure the `SolverType` to `expltau`.

```
set(configsetObj, 'SolverType', 'expltau');
get(configsetObj, 'SolverOptions')


Solver Settings: (expltau)

     ErrorTolerance:       3.000000e-002
     LogDecimation:        1
     RandomState:          []
```

See Also

`addconfigset`, `getconfigset`

SolverType property

Purpose

Select solver type for simulation

Description

The `SolverType` property lets you specify the solver to use for a simulation. For a discussion about solver types, see "Choosing a Simulation Solver".

Changing the solver type changes the options (properties) specified in the `SolverOptions` property of the `configset` object. If you change any `SolverOptions`, these changes are persistent when you switch `SolverType`. For example, if you set the `ErrorTolerance` for the `expltau` solver and then change to `impltau` when you switch back to `expltau`, the `ErrorTolerance` will have the value you assigned.

Characteristics

Applies to	Object: `Configset`
Data type	`enum`
Data values	`'ode15s'`, `'ode23t'`, `'ode45'`, `'sundials'`, `'ssa'`, `'expltau'`, `'impltau'`. Default is `'ode15s'`.

Note If your model contains events, you cannot specify `'expltau'` or `'impltau'` for the `SolverType` property.

Note If your model contains doses, you cannot specify `'ssa'`, `'expltau'`, or `'impltau'` for the `SolverType` property.

Access	Read/write

Examples

1. Retrieve the `configset` object from the `modelObj`.

```
modelObj  = sbiomodel('cell');
```

```
configsetObj = getconfigset(modelObj)


Configuration Settings - default (active)
     SolverType:                    ode15s
     StopTime:                      10.000000

   SolverOptions:
     AbsoluteTolerance:             1.000000e-006
     RelativeTolerance:             1.000000e-003
     SensitivityAnalysis:           false

   RuntimeOptions:
     StatesToLog:                   all

   CompileOptions:
     UnitConversion:                false
     DimensionalAnalysis:           true

   SensitivityAnalysisOptions:
     Inputs:                        0
     Outputs:                       0
```

2 Configure the `SolverType` to `ode45`.

```
set(configsetObj, 'SolverType', 'ode45')
configsetObj

   Configuration Settings - default (active)
     SolverType:                    ode45
     StopTime:                      10.000000

   SolverOptions:
     AbsoluteTolerance:             1.000000e-006
     RelativeTolerance:             1.000000e-003
     SensitivityAnalysis:           false
```

```
RuntimeOptions:
  StatesToLog:                 all

CompileOptions:
  UnitConversion:              false
  DimensionalAnalysis:         true

SensitivityAnalysisOptions:
  Inputs:                      0
  Outputs:                     0
```

See Also getconfigset, set

Purpose

Array of species in compartment object

Description

The `Species` property is a property of the compartment object and indicates all the species in a compartment object. `Species` is a read-only array of SimBiology species objects.

In the model object, `Species` contains a flat list of all the species that exist within all the compartments in the model. You should always access a species through its compartment rather than the model object. Use the format *`compartmentName.speciesName`*, for example, `nucleus.DNA`. Another example of the syntax is `modelObj.Compartments(2).Species(1)`. The `Species` property in the model object might not be available in a future version of the software.

Species are entities that take part in reactions. A species object is added to the `Species` property when a reaction is added to the model object with the method `addreaction`. A species object can also be added to the `Species` property with the method `addspecies`.

If you remove a reaction with the method `delete`, and a species is no longer being used by any of the remaining reactions, the species object is *not* removed from the `Species` property. You have to use the `delete` method to remove species.

There are reserved characters that cannot be used in species object names. See `Name` for more information.

Characteristics

Applies to	Object: compartment
Data type	Array of species objects
Data values	Species object. Default is `[]` (empty).
Access	Read-only

See Also

`addcompartment`, `addreaction`, `addspecies`, `delete`

SpeciesVariableNames property

Purpose

Cell array of species in reaction rate equation

Description

The SpeciesVariableNames property shows the species used by the kinetic law object to determine the ReactionRate equation in the reaction object. Use setspecies to assign SpeciesVariableNames. When you assign species to SpeciesVariableNames, SimBiology software maps these species names to SpeciesVariables in the kinetic law object.

The ReactionRate property of a reaction object shows the result of a mapping from kinetic law definition. The ReactionRate is determined by the kinetic law object Expression property by mapping ParameterVariableNames to ParameterVariables and SpeciesVariableNames to SpeciesVariables.

Characteristics

Applies to	Object: kinetic law
Data type	Cell array of strings
Data values	Cell array of species names
Access	Read/write

Examples

Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

The reactionObj KineticLaw property is configured to kineticlawObj.

3. The `'Henri-Michaelis-Menten'` kinetic law has one species variable (`S`) that you should set. To set this variable:

```
setspecies(kineticlawObj,'S', 'a');
```

4. Verify that the species variable is correct.

```
get (kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =

'a'
```

See Also

`Expression(AbstractKineticLaw, KineticLaw)`, `ParameterVariables`, `ParameterVariableNames`, `ReactionRate`, `setparameter`, `SpeciesVariables`

SpeciesVariables property

Purpose

Species in abstract kinetic law

Description

This property shows species variables that are used in the `Expression` property of the kinetic law object to determine the `ReactionRate` equation in the reaction object. Use the MATLAB function `set` to assign `SpeciesVariables` to an abstract kinetic law. For more information, see abstract kinetic law.

Characteristics

Applies to	Objects: abstract kinetic law, kineticlaw
Data type	Cell array of strings
Data values	Defined by abstract kinetic law
Access	Read/write in abstract kinetic law. Read-only in kinetic law.

Examples

Create a model, add a reaction, and assign the `SpeciesVariableNames` for the reaction rate equation.

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
reactionObj = addreaction(modelObj, 'a -> c + d');
```

2. Create a kinetic law object for the reaction object, of the type `'Henri-Michaelis-Menten'`.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

   `reactionObj` `KineticLaw` property is configured to `kineticlawObj`.

3. View the species variable for`'Henri-Michaelis-Menten'` kinetic law.

```
get (kineticlawObj, 'SpeciesVariables')
```

   MATLAB returns:

```
ans =

    'S'
```

See Also

`Expression(AbstractKineticLaw, KineticLaw)`, `ParameterVariables`, `ParameterVariableNames`, `ReactionRate`, `set`, `setparameter`, `SpeciesVariableNames`

StartTime property

Purpose

Start time for initial dose time

Description

`StartTime` is a property of a `RepeatDose` object. For a series of repeated doses, the `StartTime` property defines the amount of time that elapses before the first (initial) dose is given.

Characteristics

Applies to	Objects: `RepeatDose`
Data type	`double`
Data values	Nonnegative real number. Default value is `0`
Access	Read-write

See Also

`RepeatDose object`

Purpose

Specify species, compartment, or parameter data recorded

Description

The `StatesToLog` property specifies the species, compartment, or parameter data to log during a simulation. This is the data returned in `x` during execution of `[t,x] = sbiosimulate(modelObj)`. By default, all species, nonconstant compartments, and nonconstant parameters are logged.

Characteristics

Applies to	Object: `RuntimeOptions`
Data type	Object or vector of objects
Data values	Species objects, compartment objects, or parameter objects. Default is `all`, which means all species objects, all compartment objects whose `ConstantCapacity = false`, and all parameter objects whose `ConstantValue = false`.
Access	Read/write

Examples

This example shows how to assign species to `StatesToLog`.

1 Create a model object by importing the file `oscillator.xml`.

```
modelObj  = sbmlimport('oscillator');
```

2 Retrieve the first and second species in `modelObj`.

```
speciesObj1 = modelObj.Species(1);
speciesObj2 = modelObj.Species(2);
```

3 Retrieve the `configsetObj` of `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

4 Set the `StatesToLog` to record three species: two using the retrieved species objects and one using indexing and view the species in `StatesToLog`.

```
set (configsetObj.RuntimeOptions, 'StatesToLog', ...
    [speciesObj1, speciesObj2, modelObj.Species(3)]);
get(configsetObj.RuntimeOptions, 'StatesToLog')
```

5 Set the `StatesToLog` property back to the default setting of `all`.

```
set (configsetObj.RuntimeOptions, 'StatesToLog', 'all');
```

Purpose

Species coefficients in reaction

Description

The `Stoichiometry` property specifies the species coefficients in a reaction. Enter an array of `doubles` indicating the stoichiometry of reactants (negative value) and products (positive value). Example: [-1 -1 2].

The `double` specified cannot be `0`. The reactants of the reaction are defined with a negative number. The products of the reaction are defined with a positive number. For example, the reaction 3 H + A-> 2 C + F has the `Stoichiometry` value of [-3 -1 2 1].

When this property is configured, the `Reaction` property updates accordingly. In the above example, if the `Stoichiometry` value was set to [-2 -1 2 3], the reaction is updated to 2H + A -> 2C + 3F.

The length of the `Stoichiometry` array is the sum of the `Reactants` array and the `Products` array. To remove a product or reactant from a reaction, use the `rmproduct` or `rmreactant` function. Add a product or reactant and set stoichiometry with methods `addproduct` and `addreactant`.

ODE solvers support `double` stoichiometry values such as `0.5`. Stochastic solvers and dimensional analysis currently support only integers in `Stoichiometry`, therefore you must balance the reaction equation and specify integer values for these two cases.

`A -> null` has a stoichiometry value of `[-1]`. `null -> B` has a stoichiometry value of `[1]`.

Characteristics

Applies to	Object: reaction
Data type	Double array
Data values	1-by-n double, where n is length (products) + length (reactants). Default is [ ] (empty).
Access	Read/write

Examples

1 Create a reaction object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, '2 a + 3 b -> d + 2 c');
```

2 Verify the `Reaction` and `Stoichiometry` properties for `reactionObj`.

```
get(reactionObj,'Stoichiometry')
```

MATLAB returns:

```
ans =

-2    -3     1     2
```

3 Set stoichiometry to `[-1 -2 2 2]`.

```
set (reactionObj, 'Stoichiometry', [-1 -2 2 2]);
get (reactionObj, 'Stoichiometry')
```

MATLAB returns:

```
ans =

  -1    -2     2     2
```

4 Note with `get` that the `Reaction` property updates automatically.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =

a + 2 b -> 2 d + 2 c
```

See Also

`addproduct`, `addreactant`, `addreaction`, `Reaction`, `rmproduct`, `rmreactant`

Purpose

Simulation time criteria to stop simulation

Description

`StopTime` is a property of a `Configset` object. This property sets the maximum simulation time criteria to stop a simulation. Time units are specified by the `TimeUnits` property of the `Configset` object.

A simulation stops when it meets any of the criteria specified by `StopTime`, `MaximumNumberOfLogs`, or `MaximumWallClock`. However, if you specify the `OutputTimes` property of the `SolverOptions` property of the `Configset` object, then `StopTime` and `MaximumNumberOfLogs` are ignored. Instead, the last value in `OutputTimes` is used as the `StopTime` criteria, and the length of `OutputTimes` is used as the `MaximumNumberOfLogs` criteria.

Characteristics

Applies to	Object: `Configset`
Data type	`double`
Data values	Nonnegative scalar. Default is `10`.
Access	Read/write

Examples

Set Simulation Time Criteria to Stop Simulation

1. Create a `model` object named `cell` and save it in a variable named `modelObj`. Retrieve the configuration set from `modelObj` and save it in a variable named `configsetObj`.

```
modelObj  = sbiomodel('cell');
configsetObj = getconfigset(modelObj);
```

2. Configure the simulation stop criteria by setting the `StopTime` property to `20` seconds. Leave the `MaximumNumberOfLogs`and `MaximumWallClock` properties at their default values of `Inf`.

```
set(configsetObj, 'StopTime', 20)
get(configsetObj)
```

```
                        Active: 1
                CompileOptions: [1x1 SimBiology.CompileOptions]
                          Name: 'default'
                         Notes: ''
                RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
    SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOpti
                 SolverOptions: [1x1 SimBiology.ODESolverOptions]
                    SolverType: 'ode15s'
                      StopTime: 20
          MaximumNumberOfLogs: Inf
             MaximumWallClock: Inf
                     TimeUnits: 'second'
                          Type: 'configset'
```

When you simulate `modelObj`, the simulation stops when the simulation time reaches `20` seconds.

See Also

`Configset object`, `MaximumNumberOfLogs`, `MaximumWallClock`, `OutputTimes`, `TimeUnits`

Purpose

Specify type of stop time for simulation

Note `StopTimeType` will be removed in a future release. Use `StopTime`,`MaximumNumberOfLogs`, `MaximumWallClock` and `OutputTimes` instead.

Description

`StopTimeType` is a property of a `Configset` object. The `StopTimeType` property sets the type of stop time for a simulation. The stop time is specified in the `StopTime` property of the `configset` object. Valid types are `approxWallTime`, `numberOfLogs`, and `simulationTime`. The default is `simulationTime`.

- `simulationTime` — Specify the stop time for the simulation. The solver determines and sets the time steps and the simulation stops when it reaches the specified `StopTime`.
- `approxWallTime` — Specify the approximate stop time according to the clock. For example,`10s` of `approxWallTime` is approximately `10s` of real time.
- `numberOfLogs` — Specify the total number of simulation steps to be recorded during the simulation. For example if you want to log three simulation steps, the `numberOfLogs` is `3`. The simulation will stop after the specified `numberOfLogs`.

You can change the `StopTimeType` setting with the `set` function.

Characteristics

Applies to	Object: `Configset`
Data type	`enum`
Data values	`approxWallTime`, `numberOfLogs`, and `simulationTime`
Access	Read/write

See Also `set`, `StatesToLog`, `StopTime`, `TimeUnits`

Tag property

Purpose

Specify label for SimBiology object

Description

The `Tag` property specifies a label associated with a SimBiology object. Use this property to group objects and then use `sbioselect` to retrieve. For example, use the `Tag` property in reaction objects to group synthesis or degradation reactions. You can then retrieve all synthesis reactions using `sbioselect`. Similarly, for species objects you can enter and store classification information, for example, membrane protein, transcription factor, enzyme classifications, or whether a species is an independent variable. You can also enter the full form of the name of the species.

Characteristics

Applies to	Objects: abstract kinetic law, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species
Data type	`char string`
Data values	Any `char string`
Access	Read/write

Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

2 Add a reaction object and set the `Tag` property to `'Synthesis Reaction'`.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
set (reactionObj, 'Tag', 'Synthesis Reaction')
```

3 Verify the `Tag` assignment.

```
get (reactionObj, 'Tag');
```

MATLAB returns:

```
ans =
```

```
'Synthesis Reaction'
```

See Also

addkineticlaw, addparameter, addreaction, addrule, addspecies, RepeatDose object, sbioabstractkineticlaw, sbiomodel, sbioroot, ScheduleDose object

Purpose

Species receiving dose

Description

`TargetName` is a property of a `RepeatDose` or `ScheduleDose` object. This property defines the SimBiology species receiving the dose. The dose amount increases the species amount at each time interval defined by a repeat dose or at each time point defined by a schedule dose.

The value of `TargetName` is the name of a species. If the model has more than one species with the same name, `TargetName` is defined as *compartmentName.speciesName*, where *compartmentName* is the name of the compartment containing the species.

Characteristics

Applies to	Objects: `RepeatDose`, `ScheduleDose`
Data type	`string`
Data values	Species name. Default value is ” (empty).
Access	Read/Write

See Also

`ScheduleDose object` and `RepeatDose object`

Trigger property

Purpose

Event trigger

Description

`Trigger` is a property of an `Event` object

A `Trigger` is a condition that must become true for an event to execute. You can use a combination of relational and logical operators to build a trigger expression. `Trigger` can be a string, an expression, or a function handle that when evaluated returns a value of `true` or `false`. A `Trigger` can access species, parameters, and compartments.

A trigger can contain the keyword `time` and relational operators to trigger an event that occurs at a specific time during the simulation. For example, `time >= x`. In this example trigger, note that:

- The units associated with the keyword `time` are the units for the `TimeUnits` property for the `Configset` object associated with the simulation.
- If `x` is an expression containing compartments, species, or parameters, then any units associated with the expression must have the same dimensions as the keyword `time`.
- If `x` is a raw number, then its dimensions (and units, if unit conversion is on) are assumed to be the same as the keyword `time`.

For more information about how the SimBiology software handles events, see “How Events Are Evaluated”. For examples of event functions, see “Specifying Event Triggers”.

Characteristics

Applies to	Object: `event`
SimBiology type	String, function handle
SimBiology values	Specify a MATLAB expression as a string. Default is `''` (empty string).
Access	Read/write

Examples

1. Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator');
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

2. Set the `Trigger` property of the event object.

```
set(eventObj, 'Trigger', '(time >=5) && (speciesA<1000)');
```

3. Get the `Trigger` property.

```
get(eventObj, 'Trigger')
```

See Also

`Event object`, `EventFcns`

Time property

Purpose

Simulation time steps or schedule dose times

Description

`Time` is a property of a `SimData` or `ScheduleDose` object.

SimData Object

For a simulation, the `Time` property records the time steps.

ScheduleDose Object

For a series of scheduled doses, the `Time` property defines the times to give a dose.

A ScheduleDose object defines a series of doses. Each dose can have a different amount, as defined by an amount array in the `Amount` property, and given at specified times, as defined by a time array in the `Time` property. A rate array in the `Rate` property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

Characteristics

Applies to	Objects: `SimData`, `ScheduleDose`
Data type	`double` (SimData), `double array` (ScheduleDose)
Data values	Vector of doubles (SimData)Array of nonnegative real numbers. Default value is `[]` (ScheduleDose)
Access	Read-only

See Also

`ScheduleDose object`, `SimData object`, `StopTime`, `StopTimeType`

Purpose

Show time units for dosing and simulation

Description

The TimeUnits property specifies time units for these properties:

- StopTime property of a Configset object
- OutputTimes and AbsoluteToleranceStepSize properties of the SolverOptions property of a Configset object
- StartTime and Interval properties of a RepeatDose object
- Time property of a ScheduleDose object
- Time property of a SimData object

Note If you change the value of the TimeUnits property, make sure:

- You update the values of the Time, StartTime, Interval, StopTime, and OutputTimes properties accordingly.
- You update raw numbers used in any event triggers that use the keyword time accordingly. For more information, see Trigger.
- The units, if any, associated with expressions used in any event triggers that use the keyword time, are consistent with the updated TimeUnits property. For more information, see Trigger.

Characteristics

Applies to	Objects: Configset, RepeatDose, ScheduleDose, SimData
Data type	string

Data values	Empty string or a string specifying any unit defined in the Units Library. Default value is: • `second` — properties of a `Configset` object or `SimData` object for a `model` object created using `sbiomodel` • `hour` — properties of a `Configset` object or `SimData` object for a `model` object created from a `PKModelDesign` object • `''` (empty string) — properties of `RepeatDose` and `ScheduleDose` objects
Access	Read/write for properties of `Configset`, `RepeatDose`, and `ScheduleDose` objects Read only for properties of `SimData` objects

See Also

`Configset object`, `RepeatDose object`, `ScheduleDose object`, `SimData object`, `Interval`, `OutputTimes`, `StartTime`, `StopTime`, `Time`

Purpose

Display SimBiology object type

Description

The `Type` property indicates a SimBiology object type. When you create a SimBiology object, the value of `Type` is automatically defined.

For example, when a `Species` object is created, the value of the `Type` property is automatically defined as `'species'`.

Characteristics

Applies to	Objects: abstract kinetic law, compartment, configuration set, `CompileOptions`, event, kinetic law, model, parameter, reaction, RepeatDose, root, rule, ScheduleDose, species, `RuntimeOptions`, `SolverOptions`, unit, unitprefix, and variant
Data type	`char string`
Data values	`abstract_kinetic_law`, `compartment`, `configset`, `compileoptions`, `event`, `kineticlaw`, `parameter`, `reaction`, `repeatdose`, `root`, `rule`, `runtimeoptions`, `sbiomodel`, `scheduledose`, `species`, `solveroptions`, `unit`, `unitprefix`, and `variant`
Access	Read-only

See Also

`RepeatDose object`, `sbiomodel`, `sbioroot`, `ScheduleDose object`, `setactiveconfigset`

UnitConversion property

Purpose

Perform unit conversion

Description

The `UnitConversion` property specifies whether to perform unit conversion for the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object.

When `UnitConversion` is set to `true`, the SimBiology software converts the matching physical quantities to one consistent unit system in order to resolve them. This conversion is in preparation for correct simulation, but species amounts are returned in the user-specified units.

For example, consider a reaction `a + b  > c`. Using mass action kinetics the reaction rate is defined as `a*b*k` where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are 0.01M and 0.005M respectively, then units of `k` are `1/(M*second)`. If you specify `k` with another equivalent unit definition, for example, `1/((molecules/liter)*second)`, `UnitConversion` occurs after `DimensionalAnalysis`.

Unit conversion requires dimensional analysis. If `DimensionalAnalysis` is off, and you turn `UnitConversion` on, then `DimensionalAnalysis` is turned on automatically. If `UnitConversion` is on and you turn off `DimensionalAnalysis`, then `UnitConversion` is turned off automatically.

If `UnitConversion` fails, then you see an error when you simulate (`sbiosimulate`).

If `UnitConversion` is set to `false`, the simulation uses the given object values.

Unit conversion involving temperature supports `Celsius` as the temperature unit. Avoid using mixed temperature units as you might get an error.

Characteristics

Applies to	Object: `CompileOptions` (in `configset` object)
Data type	`boolean`
Data values	`true` or `false`. Default value is `false`.
Access	Read/write

Examples

This example shows how to retrieve and set `unitconversion` from the default `true` to `false` in the default configuration set in a model object.

1. Import a model.

```
modelObj = sbmlimport('oscillator')


SimBiology Model - Oscillator

   Model Components:
     Models:            0
     Parameters:        0
     Reactions:         42
     Rules:             0
     Species:           23
```

2. Retrieve the `configset` object of the model object.

```
configsetObj = getconfigset(modelObj)

 Configuration Settings - default (active)
     SolverType:           ode15s
     StopTime:             10.000000

   SolverOptions:
     AbsoluteTolerance:    1.000000e-006
     RelativeTolerance:    1.000000e-003
```

```
   RuntimeOptions:
     StatesToLog:          all

   CompileOptions:
     UnitConversion:       false
     DimensionalAnalysis:  true
```

3 Retrieve the `CompileOptions` object.

```
optionsObj = get(configsetObj,'CompileOptions')

Compile Settings:

     UnitConversion:       false
     DimensionalAnalysis:  true
```

4 Assign a value of false to `UnitConversion`.

```
 set(optionsObj,'UnitConversion', true)
```

See Also

`get`, `getconfigset`, `sbiosimulate`, `set`

Purpose

Specify data to associate with object

Description

Property to specify data that you want to associate with a SimBiology object. The object does not use this data directly, but you can access it using the function `get` or dot notation.

Characteristics

Applies to	Objects: abstract kinetic law, configuration set, compartment, data, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, or unit
Data type	Any
Data values	Any. Default is empty.
Access	Read/write

See Also

`RepeatDose object`, `sbioabstractkineticlaw`, `sbiomodel`, `sbioroot`, `sbiounit`, `sbiounitprefix`, `ScheduleDose object`

UserDefinedLibrary property

Purpose

Library of user-defined components

Description

`UserDefinedLibrary` is a SimBiology root object property containing all user-defined components of unit, unit prefixes, and kinetic laws that you define. You can add, modify, or delete components in the user-defined library. The `UserDefinedLibrary` property is an object that contains the following properties:

- `Units` — Contains any user-defined units. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined units either by using the command `sbiowhos -userdefined -unit`, or by accessing the root object.
- `UnitPrefixes` — Contains any user-defined unit prefixes. You can specify unit prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined unit prefixes either by using the command `sbiowhos -userdefined -unitprefix`, or by accessing the root object.
- `KineticLaws` — Contains any user-defined kinetic laws. Use the command `sbiowhos -userdefined -kineticlaw` to see the list of user-defined kinetic laws. You can use user-defined kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, `kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');`.

  See "Kinetic Law Definition" on page 3-65 for a definition and more information.

Characteristics

Applies to	Object: root
Data type	object

Data values	Unit, unit prefix, and abstract kinetic law objects
Access	Read-only

Characteristics for `UserDefinedLibrary` properties:

- `Units`

Applies to	`UserDefinedLibrary` property
Data type	Unit objects
Data values	Units
Access	Read/write

- `UnitPrefixes`

Applies to	`UserDefinedLibrary` property
Data type	Unit prefix objects
Data values	Unit prefixes
Access	Read/write

- `KineticLaws`

Applies to	`UserDefinedLibrary` property
Data type	Abstract kinetic law object
Data values	Kinetic laws
Access	Read/write

Examples

Example 1

This example uses the command `sbiowhos` to show the current list of user-defined components.

```
sbiowhos -userdefined -kineticlaw
sbiowhos -userdefined -unit
sbiowhos -userdefined -unitprefix
```

Example 2

This example shows the current list of user-defined components by accessing the root object.

```
rootObj = sbioroot;
get(rootObj.UserDefinedLibrary, 'KineticLaws')
get(rootObj.UserDefinedLibrary, 'Units')
get(rootObj.UserDefinedLibrary, 'UnitPrefixes')
```

See Also

`BuiltInLibrary`, `sbioaddtolibrary`, `sbioremovefromlibrary`, `sbioroot`, `sbiounit`, `sbiounitprefix`

Value property

Purpose

Assign value to parameter object

Description

The `Value` property is the value of the parameter object. The parameter object defines an assignment that can be used by the model object and/or the kinetic law object. Create parameters and assign `Value` using the method `addparameter`.

Characteristics

Applies to	Object: parameter
Data type	`double`
Data values	Any `double`. Default value is `1.0`.
Access	Read/write

Examples

Assign a parameter with a value to the model object.

1. Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
```

2. Add a parameter to the model object (`modelObj`) with `Value` `0.5`.

```
parameterObj1 = addparameter (modelObj, 'K1', 0.5)
```

MATLAB returns:

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1         K1       0.5
```

See Also

`addparameter`

ValueUnits property

Purpose

Parameter value units

Description

The `ValueUnits` property indicates the unit definition of the parameter object `Value` property. `ValueUnits` can be one of the built-in units. To get a list of the built-in units, use the `sbioshowunits` function. If `ValueUnits` changes from one unit definition to another, the `Value` does not automatically convert to the new units. The `sbioconvertunits` function does this conversion.

You can add a parameter object to a model object or a kinetic law object.

Characteristics

Applies to	Object: parameter
Data type	`char string`
Data values	Unit from units library. Default is `''` (empty string). Note that the default value of an empty string means unspecified. Unspecified units are permitted during dimensional analysis, but not during unit conversion. (Use the string `'dimensionless'` to specify dimensionless units.)
Access	Read/write

Examples

Assign a parameter with a value to the model object.

1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
```

2 Add a parameter with `Value 0.5`, and assign it to the model object (`modelObj`).

```
parameterObj1 = addparameter(modelObj, 'K1', 0.5, 'ValueUnits', '1/second')
```

MATLAB returns:

```
SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
 1         K1       0.5       1/second
```

See Also

addparameter, sbioconvertunits, sbioshowunits

ZeroOrderDurationParameter property

Purpose

Zero-order dose absorption duration

Description

`ZeroOrderDurationParameter` is a property of the `PKModelMap` object. It specifies the name(s) of parameter object(s) that represent the duration of absorption when the `DosingType` property is `ZeroOrder`.

Specify the name(s) of parameter object(s) that are:

- Scoped to a model
- Constant, that is, their `ConstantValue` property is `true`

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the `ZeroOrderDurationParameter` property and the `DosingType` property. For a dose that is not dosed with zero-order kinetics, use `''` (an empty string). For an example, see "Dosing Multiple Compartments in a Model".

Characteristics

Applies to	Object: `PKModelMap`
Data type	`char string` or cell array of strings

Tip If you are not using any zero-order doses, you can set this property to a cell array of empty strings, or simply an empty cell array.

Data values	Name of a parameter object or empty. Default is an empty cell array. The parameter object(s) must be: • Scoped to a model • Constant, that is, have a `ConstantValue` property set to `true`
Access	Read/write

See Also

"Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" in the SimBiology User's Guide, `DosingType`, `PKModelMap object`

Index

A

AbsoluteTolerance property
  reference 3-2
AbsoluteToleranceScaling property
  reference 3-4
AbsoluteToleranceStepSize property
  reference 3-5
abstract kinetic law object
  reference 2-4
Active property
  reference 3-6
addcompartment method
  reference 2-6
addCompartment method
  reference 2-11
addconfigset method
  reference 2-15
addcontent method
  reference 2-19
adddose method
  reference 2-21
addevent method
  reference 2-23
addkineticlaw method
  reference 2-26
addparameter method
  reference 2-34
addproduct method
  reference 2-39
addreactant method
  reference 2-42
addreaction method
  reference 2-45
addrule method
  reference 2-52
addspecies method
  reference 2-57
addvariant method
  reference 2-63
Amount property
  reference 3-8
AmountUnits property
  reference 3-9

B

BoundaryCondition property
  reference 3-12
BuiltInLibrary property
  reference 3-15

C

Capacity property
  reference 3-18
CapacityUnits property
  reference 3-19
commit method
  reference 2-65
compartment object
  reference 2-67
Compartments property
  reference 3-21
CompileOptions property
  reference 3-23
Composition property 3-25
configset object
  reference 2-71
Conserved Moieties
  function for 1-24
ConstantAmount property
  reference 3-27
ConstantCapacity property
  reference 3-29
ConstantValue property
  reference 3-30
construct method
  reference 2-74
constructDefaultFixedEffectValues method
  reference 2-76

Content property
reference 3-32
copyobj method
reference 2-77
CovariateLabels property
reference 3-34
CovariateModel object
reference 2-79

D

Data property
reference 3-36
DataCount property
reference 3-37
DataInfo property
reference 3-38
DataNames property
reference 3-40
DataSet property
reference 3-41
DefaultSpeciesDimension property
reference 3-42
delete method
reference 2-83
DependentVarLabel property
reference 3-44
DependentVarUnits property
reference 3-45
DimensionalAnalysis property
reference 3-46
display method
reference 2-85
Dosed property
reference 3-49
DoseLabel property
reference 3-50
DoseUnits property
reference 3-51
DosingType property
reference 3-52
DurationParameterName property
reference 3-53

E

EliminationType property
reference 3-54
Ensemble Runs
function for 1-37 1-39 1-43
ErrorTolerance property
reference 3-55
Estimated property
reference 3-57
event object
reference 2-86
Exponent property
reference 3-61
Expression property
reference 3-65

F

functions
sbioabstractkineticlaw 1-2
sbioaddtolibrary 1-21
sbioconsmoiety 1-24
sbioconvertunits 1-29
sbiocopylibrary 1-31
sbiodesktop 1-33
sbiodose 1-35
sbioensembleplot 1-37
sbioensemblerun 1-39
sbioensemblestats 1-43
sbiogetmodel 1-50
sbiolasterror 1-52
sbiolastwarning 1-56
sbioloadproject 1-57
sbiomodel 1-58
sbionlinfit 1-62

sbionlmefit 1-69
sbionlmefitsa 1-77
sbioparamestim 1-92
sbioplot 1-116
sbioremovefromlibrary 1-118
sbioreset 1-120
sbioroot 1-123
sbiosaveproject 1-125
sbioselect 1-127
sbioshowunitprefixes 1-139
sbioshowunits 1-141
sbiosimulate 1-143
sbiosubplot 1-148
sbiotrellis 1-150
sbiounit 1-152
sbiounitcalculator 1-156
sbiounitprefix 1-157
sbiovariant 1-160
sbiowhos 1-163
sbmlexport 1-165
setactiveconfigset 2-192
setparameter 2-194
setspecies 2-196
simbiology 1-169

G

get method
reference 2-91
getadjacencymatrix method
reference 2-93
getconfigset method
reference 2-95
getdata method
reference 2-99
getdose method
reference 2-106
getparameters method
reference 2-115
getsensmatrix method
reference 2-117
getspecies method
reference 2-120
getstoichmatrix method
reference 2-122
getvariant method
reference 2-124
GroupID property
reference 3-73
GroupLabel property
reference 3-74
GroupNames property
reference 3-75

H

HasLag property
reference 3-76
HasResponseVariable property
reference 3-77

I

IndependentVarLabel property
reference 3-78
IndependentVarUnits property
reference 3-79
InitialAmount property
reference 3-80
InitialAmountUnits property
reference 3-81
Inputs property
reference 3-83
Interval property
reference 3-85

K

kinetic law definition 2-4
kinetic law object
reference 2-128

KineticLaw property
reference 3-86
KineticLawName property
reference 3-88

L

LagParameter property
reference 3-90
LagParameterName property
reference 3-91
LogDecimation property
reference 3-92

M

MaximumNumberOfLogs property
reference 3-94
MaximumWallClock property
reference 3-96
MaxIterations property
reference 3-98
MaxStep property
reference 3-100
methods
addcompartment 2-6
addCompartment 2-11
addconfigset 2-15
addcontent 2-19
adddose 2-21
addevent 2-23
addkineticlaw 2-26
addparameter 2-34
addproduct 2-39
addreactant 2-42
addreaction 2-45
addrule 2-52
addspecies 2-57
addvariant 2-63
commit 2-65
construct 2-74
constructDefaultFixedEffectValues 2-76
copyobj 2-77
delete 2-83
get 2-91
getadjacencymatrix 2-93
getconfigset 2-95
getdata 2-99
getdose 2-106
getparameters 2-115
getsensmatrix 2-117
getspecies 2-120
getstoichmatrix 2-122
getvariant 2-124
removeconfigset 2-152
removedose 2-154
removevariant 2-156
rename 2-158
reorder 2-160
resample 2-165
reset 2-168
rmcontent 2-170
rmproduct 2-173
rmreactant 2-175
select 2-183
selectbyname 2-187
set 2-190
verify 2-215
Methods
display 2-85
model object
reference 2-136 2-162 2-181 2-206 2-212
ModelName property
reference 3-101
Models property
reference 3-102
Moiety Conservation
function for 1-24
Multiplier property
reference 3-103

N

Name property
    reference 3-104
Normalization property
    reference 3-107
Notes property
    reference 3-108

O

object
    abstract kinetic law 2-4
    compartment 2-67
    configset 2-71
    CovariateModel 2-79
    event 2-86
    kinetic law 2-128
    model 2-136 2-162 2-181 2-206 2-212
    parameter 2-140
    PKCompartment 2-142
    PKData 2-144
    PKModelDesign 2-146
    PKModelMap 2-148
    reaction 2-149
    root 2-177
    rule 2-179
    SimData 2-202
    unit 2-208 2-210
Observed property
    reference 3-109
Offset property
    reference 3-110
Outputs property
    reference 3-112
OutputTimes property
    reference 3-114
Owner property
    reference 3-116

P

Parameter Estimation
    function for 1-92
parameter object
    reference 2-140
Parameters property
    reference 3-119
ParameterVariableNames property
    reference 3-121
ParameterVariables property
    reference 3-123
Parent property
    reference 3-125
PKCompartment object
    reference 2-142
PKCompartments property
    reference 3-127
PKData object
    reference 2-144
PKModelDesign object
    reference 2-146
PKModelMap object
    reference 2-148
Products property
    reference 3-128
properties
    AbsoluteTolerance 3-2
    AbsoluteToleranceScaling 3-4
    AbsoluteToleranceStepSize 3-5
    Active 3-6
    Amount 3-8 to 3-9
    BoundaryCondition 3-12
    BuiltInLibrary 3-15
    Capacity 3-18
    CapacityUnits 3-19
    Compartments 3-21
    CompileOptions 3-23
    Composition 3-25
    ConstantAmount 3-27
    ConstantCapacity 3-29

ConstantValue 3-30
Content 3-32
CovariateLabels 3-34
Data 3-36
DataCount 3-37
DataInfo 3-38
DataNames 3-40
DataSet 3-41
DefaultSpeciesDimension 3-42
DependentVarLabel 3-44
DependentVarUnits 3-45
DimensionalAnalysis 3-46
Dosed 3-49
DoseLabel 3-50
DoseUnits 3-51
DosingType 3-52
DurationParameterName 3-53
EliminationType 3-54
ErrorTolerance 3-55
Estimated 3-57
Exponent 3-61
Expression 3-65
GroupID 3-73
GroupLabel 3-74
GroupNames 3-75
HasLag 3-76
HasResponseVariable 3-77
IndependentVarLabel 3-78
IndependentVarUnits 3-79
InitialAmount 3-80
InitialAmountUnits 3-81
Inputs 3-83
Interval 3-85
KineticLaw 3-86
KineticLawName 3-88
LagParameter 3-90
LagParameterName 3-91
LogDecimation 3-92
MaximumNumberOfLogs 3-94
MaximumWallClock 3-96
MaxIterations 3-98
MaxStep 3-100
ModelName 3-101
Models 3-102
Multiplier 3-103
Name 3-104
Normalization 3-107
Notes 3-108
Observed 3-109
Offset 3-110
Outputs 3-112
OutputTimes 3-114
Owner 3-116
Parameters 3-119
ParameterVariableNames 3-121
ParameterVariables 3-123
Parent 3-125
PKCompartments 3-127
Products 3-128
RandomState 3-131
Rate 3-133
RateLabel 3-135
RateUnits 3-134
Reaction 3-138
ReactionRate 3-139
Reactions 3-142
RelativeTolerance 3-143
RepeatCount 3-145
Reversible 3-146
Rule 3-149
Rules 3-154
RuleType 3-151
RunInfo 3-155
RuntimeOptions 3-156
SensitivityAnalysis 3-157
SensitivityAnalysisOptions 3-159
SolverOptions 3-163
SolverType 3-166
Species 3-169
SpeciesVariableNames 3-170

SpeciesVariables 3-172
StartTime 3-174
StatesToLog 3-175
Stoichiometry 3-177
StopTime 3-179
Tag 3-183
TargetName 3-185
Time 3-188
TimeUnits 3-189
Type 3-191
UnitConversion 3-192
UserData 3-195
UserDefinedLibrary 3-196
Value 3-199
ValueUnits 3-200
ZeroOrderDurationParameter 3-202
Properties
Reactants 3-136

R

RandomState property
reference 3-131
Rate property
reference 3-133
RateLabel property
reference 3-135
RateUnits property
reference 3-134
Reactants property
reference 3-136
reaction object
reference 2-149
Reaction property
reference 3-138
ReactionRate property
reference 3-139
Reactions property
reference 3-142
RelativeTolerance property
reference 3-143
removeconfigset method
reference 2-152
removedose method
reference 2-154
removevariant method
reference 2-156
rename method
reference 2-158
reorder method
reference 2-160
RepeatCount property
reference 3-145
resample method
reference 2-165
reset method
reference 2-168
Reversible property
reference 3-146
rmcontent method
reference 2-170
rmproduct method
reference 2-173
rmreactant method
reference 2-175
root object
reference 2-177
rule object
reference 2-179
Rule property
reference 3-149
Rules property
reference 3-154
RuleType property
reference 3-151
RunInfo property
reference 3-155
RuntimeOptions property
reference 3-156

S

sbioabstractkineticlaw function
reference 1-2
sbioaddtolibrary function
reference 1-21
sbioconsmoiety function
reference 1-24
sbioconvertunits function
reference 1-29
sbiocopylibrary function
reference 1-31
sbiodesktop function
reference 1-33
sbiodose function
reference 1-35
sbioensembleplot function
reference 1-37
sbioensemblerun function
reference 1-39
sbioensemblestats function
reference 1-43
sbiogetmodel function
reference 1-50
sbiolasterror function
reference 1-52
sbiolastwarning function
reference 1-56
sbioloadproject function
reference 1-57
sbiomodel function
reference 1-58
sbionlmefit function
reference 1-69
sbionlmefitsa function
reference 1-77
sbioparamestim function
reference 1-92
sbioplot function
reference 1-116
sbioremovefromlibrary function
reference 1-118
sbioreset function
reference 1-120
sbioroot function
reference 1-123
sbiosaveproject function
reference 1-125
sbioselect function
reference 1-127
sbioshowunitprefixes function
reference 1-139
sbioshowunits function
reference 1-141
sbiosimulate function
reference 1-143
sbiosubplot function
reference 1-148
sbiotrellis function
reference 1-150
sbiounit function
reference 1-152
sbiounitcalculator function
reference 1-156
sbiounitprefix function
reference 1-157
sbiovariant function
reference 1-160
sbiowhos function
reference 1-163
sbmlexport function
reference 1-165
select method
reference 2-183
selectbyname method
reference 2-187
Sensitivity Analysis
properties for 3-83 3-107 3-112 3-157 3-159
SensitivityAnalysis property
reference 3-157
SensitivityAnalysisOptions property

reference 3-159
set method
reference 2-190
setactiveconfigset function
reference 2-192
setparameter function
reference 2-194
setspecies function
reference 2-196
simbiology function
reference 1-169
SimData object
reference 2-202
SolverOptions property
reference 3-163
SolverType property
reference 3-166
Species property
reference 3-169
SpeciesVariableNames property
reference 3-170
SpeciesVariables property
reference 3-172
StartTime property
reference 3-174
StatesToLog property
reference 3-175
Stoichiometry property
reference 3-177
StopTime property
reference 3-179

T

Tag property
reference 3-183
TargetName property
reference 3-185
Time property
reference 3-188
TimeUnits property
reference 3-189
Type property
reference 3-191

U

unit object
reference 2-208 2-210
UnitConversion property
reference 3-192
UserData property
reference 3-195
UserDefinedLibrary property
reference 3-196

V

Value property
reference 3-199
ValueUnits property
reference 3-200
verify method
reference 2-215

Z

ZeroOrderDurationParameter property
reference 3-202