SimpleSBML Documentation

Release 2.3.0

Caroline Cannistra, Kyle Medley

Aug 18, 2021

Contents

1 Overview **3**

2 Examples **5**
2.1 Examples of Interrogating an Existing Model 7

3 Tests and Examples **9**

4 Classes and Methods **11**

Python Module Index **19**

Index **21**

This page describes the SimpleSBML package and its contents. To install SimpleSBML, go to https://github.com/sys-bio/simplesbml .

To see the documentation for libSBML, go to http://sbml.org/Software/libSBML/docs/python-api/index.html .

To read more about SBML (Systems Biology Markup Language), go to http://sbml.org/Main_Page .

CHAPTER 1

Overview

SimpleSBML is a package that can be used to construct biological models in SBML format using Python without interacting directly with the libSBML package. Using libSBML to build models can be difficult and complicated, even when the model is relatively simple, and it can take time for a user to learn how to use the package properly. This package is intended as an intuitive interface for users who are not already familiar with libSBML. It can be used to construct models with only a few lines of code, print out the resulting models in SBML format, and edit existing models in SBML format.

CHAPTER 2

Examples

Example models are taken from the SBML Level 3 Version 1 documentation.

Here is an example of a simple reaction-based model built with SbmlModel:

```
import simplesbml
model = simplesbml.SbmlModel()
model.addCompartment(1e-14, comp_id='comp')
model.addSpecies('E', 5e-21, comp='comp')
model.addSpecies('S', 1e-20, comp='comp')
model.addSpecies('P', 0.0, comp='comp')
model.addSpecies('ES', 0.0, comp='comp')
model.addReaction(['E', 'S'], ['ES'], 'comp*(kon*E*S-koff*ES)', local_params={'koff':␣
↪0.2, 'kon': 1000000.0}, rxn_id='veq')
model.addReaction(['ES'], ['E', 'P'], 'comp*kcat*ES', local_params={'kcat': 0.1}, rxn_
↪id='vcat')
```

In this example, reaction rate constants are stored locally with the reactions where they are used. It is also possible to define global parameters and use them in reaction expressions. Here is an example of using global parameters which is the most common use case:

```
import simplesbml
model = simplesbml.SbmlModel()
model.addCompartment(1e-14, comp_id='comp')
model.addSpecies('E', 5e-21, comp='comp')
model.addSpecies('S', 1e-20, comp='comp')
model.addSpecies('P', 0.0, comp='comp')
model.addSpecies('ES', 0.0, comp='comp')
model.addParameter('koff', 0.2)
model.addParameter('kon', 1000000.0)
model.addParameter('kcat', 0.1)
model.addReaction(['E', 'S'], ['ES'], 'comp*(kon*E*S-koff*ES)', rxn_id='veq')
model.addReaction(['ES'], ['E', 'P'], 'comp*kcat*ES', rxn_id='vcat')
```

SbmlModel also supports the use of events to change the system state under certain conditions, use of assignment rules and rate rules to explicitly define variable values as a function of the system state. Here is an example of events and

rate rules. In this example, the value of parameter G2 is determined by the relationship between P1 and tau, and the rates of change of P1 and P2 are explicitly defined in equation form instead of with a reaction:

```
import simplesbml
model = simplesbml.SbmlModel()
model.addCompartment(vol=1.0, comp_id='cell')
model.addSpecies('[P1]', 0.0, comp='cell')
model.addSpecies('[P2]', 0.0, comp='cell')
model.addParameter('k1', 1.0)
model.addParameter('k2', 1.0)
model.addParameter('tau', 0.25)
model.addParameter('G1', 1.0)
model.addParameter('G2', 0.0)
model.addEvent(trigger='P1 > tau', assignments={'G2': '1'})
model.addEvent(trigger='P1 <= tau', assignments={'G2': '0'})
model.addRateRule('P1', 'k1 * (G1 - P1)')
model.addRateRule('P2', 'k2 * (G2 - P2)')
```

Users can edit existing models with the writeCode() method which accepts an SBML document and produces a script of SimpleSBML commands in string format. This method converts the SBML document into a libSBML Model and scans through its elements, adding lines of code for each SimpleSBML-compatible element it finds. The output can be saved to a .py file and edited to create new models based on the original import. This can be very useful for editing existing SBML models. For instance, here is an example of a short script that reproduces the SimpleSBML to reproduce an SbmlModel object:

```
import simplesbml
model = simplesbml.SbmlModel()
model.addCompartment(1e-14, comp_id='comp')
model.addSpecies('E', 5e-21, comp='comp')
model.addSpecies('S', 1e-20, comp='comp')
model.addSpecies('P', 0.0, comp='comp')
model.addSpecies('ES', 0.0, comp='comp')
model.addReaction(['E', 'S'], ['ES'], 'comp*(kon*E*S-koff*ES)', local_params={'koff':␣
↪0.2, 'kon': 1000000.0}, rxn_id='veq')
model.addReaction(['ES'], ['E', 'P'], 'comp*kcat*ES', local_params={'kcat': 0.1}, rxn_
↪id='vcat')

# Load the sbml model and output the equivalent simplesbml scipt
code = simplesbml.writeCodeFromString(model.toSBML())
f = open('example_code.py', 'w')
f.write(code)
f.close()
```

The output saved to 'example_code.py' will look like this:

```
import simplesbml
model = simplesbml.SbmlModel(sub_units='')
model.addCompartment(vol=1e-14, comp_id='comp')
model.addSpecies(species_id='E', amt=5e-21, comp='comp')
model.addSpecies(species_id='S', amt=1e-20, comp='comp')
model.addSpecies(species_id='P', amt=0.0, comp='comp')
model.addSpecies(species_id='ES', amt=0.0, comp='comp')
model.addReaction(reactants=['E', 'S'], products=['ES'], expression='comp * (kon * E␣
↪* S - koff * ES)', local_params={'koff': 0.2, 'kon': 1000000.0}, rxn_id='veq')
model.addReaction(reactants=['ES'], products=['E', 'P'], expression='comp * kcat * ES
↪', local_params={'kcat': 0.1}, rxn_id='vcat')
```

2.1 Examples of Interrogating an Existing Model

Verison 2.0 has a set of new 'get' methods that allows a user to easily interrogate a model for its contents.:

```
import simplesbml
mymodel = loadFromFile ('mymodel.xml')  # Load the model into a string variable
model = simplesbml.loadSBMLStr(mymodel)

# Or:

model = simplesbml.loadSBMLFile('mymodel.xml')

# Or if you're using the Tellurium package:

model = simplesbml.loadSBMLStr(r.getSBML())

print ('Num compartmetns = ', model.getNumCompartmentIds())
print ('Num parameters =', model.getNumParameters())
print ('Num species =', model.getNumSpecies())
print ('Num floating species = ', model.getNumFloatingSpecies())
print ('Num floating species = ', model.getNumBoundarySpecies())
print ('Num reactions = ', model.getNumReactions())
print (model.getListOfCompartments())
print (model.getListOfAllSpecies())
print ('list of floating species = ', model.getListOfFloatingSpecies())
print ('list of boundary species = ', model.getListOfBoundarySpecies())
print ('List of reactions = ', model.getListOfReactionIds())
print ('List of rules = ', model.getListOfRuleIds())
```

Here is an example script that uses simplesbml to create a stoichiometry matrix for a model:

```
import tellurium as te, simplesbml, numpy as np

r = te.loada("""
S0 + S3 -> S2; k0*S0*S3;
S3 + S2 -> S0; k1*S3*S2;
S5 -> S2 + S4; k2*S5;
S0 + S1 -> S3; k3*S0*S1;
S5 -> S0 + S4; k4*S5;
S0 -> S5; k5*S0;
S1 + S1 -> S5; k6*S1*S1;
S3 + S5 -> S1; k7*S3*S5;
S1 -> $S4 + S4; k8*S1;

S0 = 0; S1 = 0; S2 = 0; S3 = 0; S4 = 0; S5 = 0;
k0 = 0; k1 = 0; k2 = 0; k3 = 0; k4 = 0
k5 = 0; k6 = 0; k7 = 0; k8 = 0
""")

model = simplesbml.loadSBMLStr(r.getSBML())

# Allocate space for the stoichiometry matrix
stoich = np.zeros((model.getNumFloatingSpecies(), model.getNumReactions()))
for i in range (model.getNumFloatingSpecies()):
    floatingSpeciesId = model.getNthFloatingSpeciesId (i)

    for j in range (model.getNumReactions()):
```

(continues on next page)

(continued from previous page)

```
        productStoichiometry = 0; reactantStoichiometry = 0

        numProducts = model.getNumProducts (j)
        for k1 in range (numProducts):
            productId = model.getProduct (j, k1)

            if (floatingSpeciesId == productId):
               productStoichiometry += model.getProductStoichiometry (j, k1)

        numReactants = model.getNumReactants(j)
        for k1 in range (numReactants):
            reactantId = model.getReactant (j, k1)
            if (floatingSpeciesId == reactantId):
               reactantStoichiometry += model.getReactantStoichiometry (j, k1)

        st = int(productStoichiometry - reactantStoichiometry)
        stoich[i,j] = st

print (stoich)
```

CHAPTER 3

Tests and Examples

Two test files can be found in the tests folder. The runTest.py is the more formal testing file. It was decided not to use the Python unittest due to its limitations and pyTest simply made the code unmanagable. A simple test system was therefore created. To run the tests just execute runTests.py or more simply:

```
simplesbml.tests.run()
```

Make sure you have libsbml or Tellurium installed (it can provide libsbml).

CHAPTER 4

Classes and Methods

simplesbml.**loadSBMLStr**(*sbmlStr*)
Load an SBML model in the form of a string and return an instance to SBMLModel

simplesbml.**loadSBMLFile**(*sbmlFile*)
Load an SBML model from a file and return an instance to SBMLModel

class simplesbml.**SbmlModel**(*time_units='second'*, *extent_units='mole'*, *sub_units='mole'*, *level=3*, *version=1*, *sbmlStr=None*, *sbmlFile=None*)
SbmlModel is used to construct simple models using libSBML methods and print out the model in SBML format. A user can add species, parameters, reactions, events, assignment rules, rate rules, and initial assignments to a model. Then, the user can view the model in SBML format by printing the string representation of the class.

SbmlModel contains two attributes: *document*, an SBMLDocument object, an *model*, the Model attribute of *document*.

You can also pass a SBML string and use simlesbml to obtain information about the model using the get methods, e.g

model = simplesbml.SbmlModel (sbmlStr=mySBMLString)

addCompartment(*vol=1*, *comp_id=''*)
Adds a Compartment of volume *vol* litres to the model. The default volume is 1 litre. If the user does not specify *comp_id*, the id is set to 'c<n>' where the new compartment is the nth compartment added to the model. All SbmlModel objects are initialized with a default compartment 'c1'.

addSpecies(*species_id*, *amt*, *comp='c1'*)
Adds a Species to the model. If *species_id* starts with the '$' symbol, the species is set as a boundary condition. If *species_id* is enclosed in brackets, the *amt* is the initial concentration of species within the specified compartment in mol/L. Otherwise, *amt* is the initial amount of species in moles.

addParameter(*param_id*, *val*, *units='per_second'*)
Adds a Parameter to the model. *val* is the value of the parameter, and *param_id* is the parameter id. If units are not specified by the user, the default units are 1/sec.

addReaction(*reactants*, *products*, *expression*, *local_params={}*, *rxn_id=''*)
Adds a Reaction to the model.

reactants and *products* are lists of species ids that the user wishes to define as reactants and products, respectively. If one of these lists contains a string with a number followed by a species id (i.e. '2 G6P') then the number is interpreted as the stoichiometry of the species. Otherwise it is assumed to be 1.

expression is a string that represents the reaction rate expression.

local_params is a dictionary where the keys are local parameter ids and the values are the desired values of the respective parameters.

If the user does not define a reaction id, it is set as 'v<n>' where the new reaction is the nth reaction added.

addEvent(*trigger*, *assignments*, *persistent=True*, *initial_value=False*, *priority=0*, *delay=0*, *event_id=''*)
Adds an Event to the model.

trigger is the string representation of a logical expression that defines when an event is 'triggered', meaning when the event is ready to be executed.

delay is a numerical value that defines the amount of time between when the event is triggered and when the event assignment is implemented, in previously defined model-wide time units.

assignments is a dictionary where the keys are variables to be changed and the values are the variables' new values.

persistent is a boolean that defines whether the event will still be executed if the trigger switches from `True` to `False` between the event's trigger and its execution.

initial_value is the value of *trigger* when t < 0.

priority is a numerical value that determines which event is executed if two events are executed at the same time. The event with the larger `priority` is executed.

Note: An event is only triggered when the trigger switches from `False` to `True`. If the trigger's initial value is `True`, the event will not be triggered until the value switches to `False` and then back to `True`.

---

addAssignmentRule(*var*, *math*)
Adds an AssignmentRule to the model. An assignment rule is an equation where one side is equal to the value of a state variable and the other side is equal to some expression. *var* is the id of the state variable and *math* is the string representation of the expression.

addRateRule(*var*, *math*)
Adds a RateRule to the model. A rate rule is similar to an assignment rule, but instead of describing a state variable's value as an expression, it describes the derivative of the state variable's value with respect to time as an expression. *var* is the id of the state variable and *math* is the string representation of the expression.

addInitialAssignment(*symbol*, *math*)
Adds an InitialAssignment to the model. If the initial value of a variable depends on other variables or parameters, this method can be used to define an expression that describes the initial value of the variable in terms of other variables or parameters. *symbol* is the id of the variable and *math* is the string representation of the expression.

getDocument()
Returns the SBMLDocument object of the sbmlModel.

This is not something you need to care about unless you need direct access to libsbml

getModel()
Returns the Model object of the sbmlModel.

This is not something you need to care about unless you need direct access to libsbml

getModelId()
Returns the SBML Id given to the model.

getNumCompartments()
Returns the number of compartments in the current model.

getNumSpecies()
Returns the number of all species in the current model.

getNumParameters()
Returns the number of global parameters in the current model.

getNumReactions()
Returns the number of reactions in the current model.

getNumEvents()
Returns the number of events in the current model.

getNumRules()
Returns the number of rules in the current model.

getNumInitialAssignments()
Returns the number of initial assignments in the current model.

getListOfCompartmentIds()
Returns a list of compartment Ids

getCompartmentId(*index*)
Returns the indexth compartment from the list of compartments

getCompartmentVolume(*Id*)
Parameters

Id: The Id of the compartment in question

Returns

The volume of the specified compartment

getListOfAllSpecies()
Returns a list of **ALL** species Ids in the model

getNthFloatingSpeciesId(*index*)
Returns the Id of the nth floating species

getNthBoundarySpeciesId(*index*)
Returns the Id of the nth boundary species

getCompartmentIdSpeciesIsIn(*speciesId*)
Returns the compartment Id that the species givenin th argument is in.

isSpeciesValueSet(*Id*)
Returns true if the species has been set a value (concentration or amount).

Example: if model.isSpeciesValueSet ('ATP'):

getSpeciesInitialConcentration(*Id*)
Returns the initial values for the concentration of a species with given Id

The species can be an index to the indexth species or the Id of the species. You can get the Ids by calling getListOfSpecies()

Example: value = model.getSpeciesInitialConcentration ('Glucose')

getSpeciesInitialAmount(*Id*)
Returns the initial values for the amount of a species with given Id

The species can be an index to the indexth species or the Id of the species. You can get the Ids by calling getListOfSpecies()

Example: value = model.getSpeciesInitialAmount ('Glucose')

getListOfFloatingSpecies()
Returns a list of all floating species Ids.

getListOfBoundarySpecies()
Returns a list of all boundary species Ids.

isFloatingSpecies(*Id*)
Returns true if the Id is a floating species

isBoundarySpecies(*Id*)
Returns true if the Id is a boundary species

isAmount(*Id*)
Returns true if species Id has assigned to it an amount rather then a concentration"

isConcentration(*Id*)
Returns true if species Id has assigned to it a concentration rather then an amount"

getNumFloatingSpecies()
Returns the number of floating species.

getNumBoundarySpecies()
Returns the number of boundary species.

getListOfParameterIds()
Returns a list of all global parameter Ids in the model.

getParameterId(*index*)
Returns the indexth parameter from the list of parameter

isParameterValueSet(*Id*)
Returns true if the parameter has been assigned a value

Example: if model.isParameterValueSet ('k1'):

getParameterValue(*Id*)
Returns the value for a given model parameter.

Parameters

Id: the Id of the parameter in question

Example: value = model.getParameterValue ('k1')

getListOfReactionIds()
Returns a list of all reaction Ids.

getNthReactionId(*index*)
Returns the Id of the nth reaction

getNumReactants(*Id*)

Returns the number of reactants in the reaction given by the Id argument.

Parameters

Id (string): The Id of the reaction in question.

Example: numProducts = model.getNumReactants ('J1')

getNumProducts(*Id*)
Returns the number of products in the reaction given by the Id argument.

Parameters

Id (string): The Id of the reaction in question.

Example: numProducts = model.getNumProducts ('J1')

getRateLaw(*Id*)
Returns the expression for the rate laws of the reaction given by the specified Id

Parameters

Id (string): The Id of the reaction in question.

Example: formulaStr = model.getRateLaw ('J1')

getReactant(*reactionId*, *reactantIndex*)
Returns the Id of the reactantIndexth reactant in the reaction given by the reactionId

Parameters

Id (string): The Id of the reaction in question

reactantIndex (int): The ith reactant in the reaction

Example: astr = model.getReactant ('J1', 0)

getProduct(*reactionId*, *productIndex*)
Returns the Id of the productIndexth product in the reaction given by the reactionId

Parameters

Id (string): The Id of the reaction in question

productIndex (int): The ith product in the reaction

Example: astr = model.getProduct ('J1', 0)

getReactantStoichiometry(*reactionId*, *reactantIndex*)
Returns the stoichiometry for a reactant in a reaction.

Parameters

reactionId (string): The Id of the reaction in question

reactantIndex (int): The ith reactant in the reaction

Returns

The value of the reactant stoichiometry

Example: stInt = model.getReactantStoichiometry ('J1', 0)

getProductStoichiometry(*reactionId*, *productIndex*)
Returns the stoichiometry for a product in a reaction.

Parameters

reactionId (string): The Id of the reaction in question

productIndex (int): The ith product in the reaction

Returns

The value of the product stoichiometry

Example: stInt = model.getProductStoichiometry ('J1', 0)

getNumModifiers (*reactionId*)
Returns the number of modifiers in a given reaction.

getListOfModifiers (*reactionId*)
Returns the list of modifiers in a given reaction.

getListOfRuleIds ()
Returns a list of Ids for the rules

getRuleId (*index*)
Returns the rule Id of the indexth rule

Example: rule = model.getRuleId (2)

getRuleRightSide (*rule*)
Returns the formula on the right-hand side of the rule in question.

The rule can be an index to the indexth rule or the Id of the rule. You can get the Ids by calling getListOfRules()

Example: formula = model.getRuleRightSide (0)

getRuleType (*rule*)
Returns a string indicating the type of rule in question.

The rule can be an index to the indexth rule or the Id of the rule. You can get the Ids by calling getListOfRules()

Example: ruleStr = model.getRuleType (0)

isRuleType_Assignment (*rule*)
Returns true the rule is an assignemnt type

isRuleType_Rate (*rule*)
Returns true the rule is an rate rule (ode) type

isRuleType_Algebraic (*rule*)
Returns true the rule is an algebraic rule type

getEventId (*index*)
Returns the Id for the indexth event

Example: astr = model.getEventId (0)

getEventString (*event*)
Returns the indexth event as a compelte string.

The event argument can be an index to the indexth event or the Id of the event

Example: print (model.getEventString (0))

getEventTrigger (*event*)
Returns the formula as a string for the event trigger of the event event.

The event argument can be an index to the indexth event or the Id of the event

Example : astr = model.getEventTrigger (0)

getNumEventAssignments (*index*)
Returns the number of assignments in the indexth rule.

getEventVariable (*event*, *assignmentIndex*)
Returns the event variables (i.e the left-hand side) for the assignmentIndexth assignment in the event, given by event,

The event argument can be an index to the indexth event or the Id of the event

Example: astr = model.getEventVariable (0, 0)

getEventAssignment (*event*, *assignmentIndex*)
Retuns the assignmentIndexth assignment in the event event.

The event argument can be an index to the indexth event or the Id of the event

Example: mathStr = model.getEventAssignment (1, 0)

getListOfFunctionIds ()
Returns a list of function definition Ids

getNumFunctionDefinitions ()
Returns the number of user defined functions in the model

getFunctionId (*index*)
Retuns the Id of the indexth user function definition

Example: mathStr = model.getFunctionId (0)

getFunctionBody (*func*)

getNumArgumentsInUserFunction (*func*)
Returns the number of arguments in the given function. The func argument can either be the name of a user function or its index.

Example:

nargs = model.getNumArgumentsInUserFunction ('Hill')

nargs = model.getNumArgumentsInUserFunction (0)

getListOfArgumentsInUserFunction (*func*)
Returns a list of arguments in the specified user function. The func argument can either be the name of a user function or its index in the list of user fucntions.

Example:

alist = model.getListOfArgumentsInUserFunction ('Hill')

alist = model.getListOfArgumentsInUserFunction (0)

toSBML ()
Returns the model in SBML format as a string. Also checks model consistency and prints all errors and warnings.

Example: print (model.toSBML())

simplesbml.**writeCode** (*doc*)
Returns a string containing calls to SimpleSBML functions that reproduce the model contained in the SBMLDocument *doc* in an sbmlModel object.

simplesbml.**writeCodeFromFile** (*filename*)
Reads the file saved under *filename* as an SBML format model and returns a string containing calls to SimpleSBML functions that reproduce the model in an sbmlModel object.

simplesbml.**writeCodeFromString** (*sbmlstring*)
Reads *sbmlstring* as an SBML format model and returns a string containing calls to SimpleSBML functions that reproduce the model in an sbmlModel object.

Python Module Index

S

simplesbml, 11

Index

A

addAssignmentRule() (*simplesbml.SbmlModel method*), 12
addCompartment() (*simplesbml.SbmlModel method*), 11
addEvent() (*simplesbml.SbmlModel method*), 12
addInitialAssignment() (*simplesbml.SbmlModel method*), 12
addParameter() (*simplesbml.SbmlModel method*), 11
addRateRule() (*simplesbml.SbmlModel method*), 12
addReaction() (*simplesbml.SbmlModel method*), 11
addSpecies() (*simplesbml.SbmlModel method*), 11

G

getCompartmentId() (*simplesbml.SbmlModel method*), 13
getCompartmentIdSpeciesIsIn() (*simplesbml.SbmlModel method*), 13
getCompartmentVolume() (*simplesbml.SbmlModel method*), 13
getDocument() (*simplesbml.SbmlModel method*), 12
getEventAssignment() (*simplesbml.SbmlModel method*), 17
getEventId() (*simplesbml.SbmlModel method*), 16
getEventString() (*simplesbml.SbmlModel method*), 16
getEventTrigger() (*simplesbml.SbmlModel method*), 16
getEventVariable() (*simplesbml.SbmlModel method*), 16
getFunctionBody() (*simplesbml.SbmlModel method*), 17
getFunctionId() (*simplesbml.SbmlModel method*), 17
getListOfAllSpecies() (*simplesbml.SbmlModel method*), 13
getListOfArgumentsInUserFunction() (*simplesbml.SbmlModel method*), 17
getListOfBoundarySpecies() (*simplesbml.SbmlModel method*), 14
getListOfCompartmentIds() (*simplesbml.SbmlModel method*), 13
getListOfFloatingSpecies() (*simplesbml.SbmlModel method*), 14
getListOfFunctionIds() (*simplesbml.SbmlModel method*), 17
getListOfModifiers() (*simplesbml.SbmlModel method*), 16
getListOfParameterIds() (*simplesbml.SbmlModel method*), 14
getListOfReactionIds() (*simplesbml.SbmlModel method*), 14
getListOfRuleIds() (*simplesbml.SbmlModel method*), 16
getModel() (*simplesbml.SbmlModel method*), 12
getModelId() (*simplesbml.SbmlModel method*), 12
getNthBoundarySpeciesId() (*simplesbml.SbmlModel method*), 13
getNthFloatingSpeciesId() (*simplesbml.SbmlModel method*), 13
getNthReactionId() (*simplesbml.SbmlModel method*), 14
getNumArgumentsInUserFunction() (*simplesbml.SbmlModel method*), 17
getNumBoundarySpecies() (*simplesbml.SbmlModel method*), 14
getNumCompartments() (*simplesbml.SbmlModel method*), 13
getNumEventAssignments() (*simplesbml.SbmlModel method*), 16
getNumEvents() (*simplesbml.SbmlModel method*), 13
getNumFloatingSpecies() (*simplesbml.SbmlModel method*), 14
getNumFunctionDefinitions() (*simplesbml.SbmlModel method*), 17
getNumInitialAssignments() (*simplesbml.SbmlModel method*), 13

getNumModifiers() (*simplesbml.SbmlModel method*), 16
getNumParameters() (*simplesbml.SbmlModel method*), 13
getNumProducts() (*simplesbml.SbmlModel method*), 15
getNumReactants() (*simplesbml.SbmlModel method*), 14
getNumReactions() (*simplesbml.SbmlModel method*), 13
getNumRules() (*simplesbml.SbmlModel method*), 13
getNumSpecies() (*simplesbml.SbmlModel method*), 13
getParameterId() (*simplesbml.SbmlModel method*), 14
getParameterValue() (*simplesbml.SbmlModel method*), 14
getProduct() (*simplesbml.SbmlModel method*), 15
getProductStoichiometry() (*simplesbml.SbmlModel method*), 15
getRateLaw() (*simplesbml.SbmlModel method*), 15
getReactant() (*simplesbml.SbmlModel method*), 15
getReactantStoichiometry() (*simplesbml.SbmlModel method*), 15
getRuleId() (*simplesbml.SbmlModel method*), 16
getRuleRightSide() (*simplesbml.SbmlModel method*), 16
getRuleType() (*simplesbml.SbmlModel method*), 16
getSpeciesInitialAmount() (*simplesbml.SbmlModel method*), 13
getSpeciesInitialConcentration() (*simplesbml.SbmlModel method*), 13

I

isAmount() (*simplesbml.SbmlModel method*), 14
isBoundarySpecies() (*simplesbml.SbmlModel method*), 14
isConcentration() (*simplesbml.SbmlModel method*), 14
isFloatingSpecies() (*simplesbml.SbmlModel method*), 14
isParameterValueSet() (*simplesbml.SbmlModel method*), 14
isRuleType_Algebraic() (*simplesbml.SbmlModel method*), 16
isRuleType_Assignment() (*simplesbml.SbmlModel method*), 16
isRuleType_Rate() (*simplesbml.SbmlModel method*), 16
isSpeciesValueSet() (*simplesbml.SbmlModel method*), 13

L

loadSBMLFile() (*in module simplesbml*), 11
loadSBMLStr() (*in module simplesbml*), 11

S

SbmlModel (*class in simplesbml*), 11
simplesbml (*module*), 1, 11

T

toSBML() (*simplesbml.SbmlModel method*), 17

W

writeCode() (*in module simplesbml*), 17
writeCodeFromFile() (*in module simplesbml*), 17
writeCodeFromString() (*in module simplesbml*), 17