SimpleSBML Documentation

Release 1.2.2

Caroline Cannistra, Kyle Medley

Sep 20, 2017

Contents

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 (Synthetic 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 simplify existing models in SBML format by finding the SimpleSBML methods that can be used to build a libSBML version of the model.

CHAPTER 2

Classes and Methods

class simplesbml.**sbmlModel**(*self*, *time_units='second'*, *extent_units='mole'*, *sub_units='mole'*, *level=3*, *version=1*)

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, and *model*, the Model attribute of *document*.

addCompartment(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*)

Returns the SBMLDocument object of the sbmlModel.

getModel(*self*)

Returns the Model object of the sbmlModel.

toSBML(*self*)

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

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.

CHAPTER 3

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 this usage:

```
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, and the 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 instantaneously 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. For instance, here is an example of a short script that reproduces the SimpleSBML code 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')

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')
```

Python Module Index

S

Index

S

W