

Stoched: APC 524 Final Project

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b><a href="#">stoched</a></b>	<b>1</b>
<b>2</b>	<b><a href="#">Authors</a></b>	<b>3</b>
<b>3</b>	<b><a href="#">User Guide</a></b>	<b>5</b>
<b>4</b>	<b><a href="#">Hierarchical Index</a></b>	<b>9</b>
4.1	<a href="#">Class Hierarchy</a> . . . . .	9
<b>5</b>	<b><a href="#">Class Index</a></b>	<b>11</b>
5.1	<a href="#">Class List</a> . . . . .	11
<b>6</b>	<b><a href="#">File Index</a></b>	<b>13</b>
6.1	<a href="#">File List</a> . . . . .	13
<b>7</b>	<b><a href="#">Class Documentation</a></b>	<b>15</b>
7.1	<a href="#">EulerLeap Class Reference</a> . . . . .	15
7.1.1	<a href="#">Detailed Description</a> . . . . .	15
7.1.2	<a href="#">Constructor &amp; Destructor Documentation</a> . . . . .	16
7.1.2.1	<a href="#">EulerLeap()</a> . . . . .	16
7.1.2.2	<a href="#">~EulerLeap()</a> . . . . .	16
7.1.3	<a href="#">Member Function Documentation</a> . . . . .	16
7.1.3.1	<a href="#">set_to_initial_state()</a> . . . . .	17
7.1.3.2	<a href="#">step()</a> . . . . .	17
7.2	<a href="#">Event Class Reference</a> . . . . .	17
7.2.1	<a href="#">Detailed Description</a> . . . . .	18

7.2.2	Constructor & Destructor Documentation . . . . .	18
7.2.2.1	Event() . . . . .	18
7.2.2.2	~Event() . . . . .	18
7.2.3	Member Function Documentation . . . . .	19
7.2.3.1	addFunction() . . . . .	19
7.2.3.2	getDeltaVar() . . . . .	19
7.2.3.3	getRate() . . . . .	19
7.2.3.4	getSize() . . . . .	20
7.2.3.5	setDeltaVar() . . . . .	20
7.2.3.6	setRate() . . . . .	20
7.2.3.7	useFunction() . . . . .	21
7.3	FirstReaction Class Reference . . . . .	21
7.3.1	Detailed Description . . . . .	22
7.3.2	Constructor & Destructor Documentation . . . . .	22
7.3.2.1	FirstReaction() . . . . .	22
7.3.2.2	~FirstReaction() . . . . .	23
7.3.3	Member Function Documentation . . . . .	23
7.3.3.1	step() . . . . .	23
7.4	MidpointLeap Class Reference . . . . .	23
7.4.1	Detailed Description . . . . .	24
7.4.2	Constructor & Destructor Documentation . . . . .	24
7.4.2.1	MidpointLeap() . . . . .	24
7.4.2.2	~MidpointLeap() . . . . .	25
7.4.3	Member Function Documentation . . . . .	25
7.4.3.1	set_to_initial_state() . . . . .	25
7.4.3.2	step() . . . . .	25
7.5	Model Class Reference . . . . .	26
7.5.1	Detailed Description . . . . .	27
7.5.2	Constructor & Destructor Documentation . . . . .	27
7.5.2.1	Model() . . . . .	27

7.5.2.2	<a href="#">~Model()</a>	27
7.5.3	<a href="#">Member Function Documentation</a>	27
7.5.3.1	<a href="#">addEvent()</a>	27
7.5.3.2	<a href="#">addEventFct()</a>	28
7.5.3.3	<a href="#">addVars()</a>	28
7.5.3.4	<a href="#">checkTauLeapAvail()</a>	28
7.5.3.5	<a href="#">getContDeriv()</a>	29
7.5.3.6	<a href="#">getEventRate()</a>	29
7.5.3.7	<a href="#">getEventsCount()</a>	29
7.5.3.8	<a href="#">getIthVar()</a>	30
7.5.3.9	<a href="#">getVarsCount()</a>	30
7.5.3.10	<a href="#">getVarsString()</a>	30
7.5.3.11	<a href="#">setDelta()</a>	30
7.5.3.12	<a href="#">setTauLeapFalse()</a>	31
7.5.3.13	<a href="#">updateRates()</a>	31
7.5.3.14	<a href="#">updateState()</a>	31
7.5.3.15	<a href="#">useEventFct()</a>	32
7.6	<a href="#">NextReaction Class Reference</a>	32
7.6.1	<a href="#">Detailed Description</a>	33
7.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	33
7.6.2.1	<a href="#">NextReaction()</a>	33
7.6.2.2	<a href="#">~NextReaction()</a>	34
7.6.3	<a href="#">Member Function Documentation</a>	34
7.6.3.1	<a href="#">set_to_initial_state()</a>	34
7.6.3.2	<a href="#">step()</a>	34
7.7	<a href="#">Paramset Class Reference</a>	35
7.7.1	<a href="#">Detailed Description</a>	35
7.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	35
7.7.2.1	<a href="#">Paramset()</a>	36
7.7.2.2	<a href="#">~Paramset()</a>	36

7.8	Realization Class Reference	36
7.8.1	Detailed Description	37
7.8.2	Constructor & Destructor Documentation	37
7.8.2.1	Realization()	37
7.8.2.2	~Realization()	38
7.8.3	Member Function Documentation	38
7.8.3.1	output_state()	38
7.8.3.2	rates_are_zero()	38
7.8.3.3	set_to_initial_state()	39
7.8.3.4	simulate()	39
7.9	RealizationFactory Class Reference	39
7.9.1	Detailed Description	39
7.9.2	Member Function Documentation	40
7.9.2.1	NewRealization()	40
7.10	rng Class Reference	40
7.10.1	Detailed Description	41
7.10.2	Member Function Documentation	41
7.10.2.1	log_factorial()	41
7.11	xoroshiro128plus Class Reference	42
7.11.1	Detailed Description	42
7.11.2	Constructor & Destructor Documentation	42
7.11.2.1	xoroshiro128plus()	43
7.11.3	Member Function Documentation	43
7.11.3.1	jump()	43

<b>8</b>	<b>File Documentation</b>	<b>45</b>
8.1	/Users/Caleb/APC524/stoched/src/eulerleap.cc File Reference	45
8.1.1	Detailed Description	45
8.2	/Users/Caleb/APC524/stoched/src/eulerleap.h File Reference	45
8.2.1	Detailed Description	46
8.3	/Users/Caleb/APC524/stoched/src/event.cc File Reference	46
8.3.1	Detailed Description	46
8.4	/Users/Caleb/APC524/stoched/src/event.h File Reference	47
8.4.1	Detailed Description	47
8.5	/Users/Caleb/APC524/stoched/src/eventtests.cc File Reference	47
8.5.1	Detailed Description	48
8.6	/Users/Caleb/APC524/stoched/src/firstreaction.cc File Reference	48
8.6.1	Detailed Description	48
8.7	/Users/Caleb/APC524/stoched/src/firstreaction.h File Reference	48
8.7.1	Detailed Description	49
8.8	/Users/Caleb/APC524/stoched/src/midpointleap.cc File Reference	49
8.8.1	Detailed Description	49
8.9	/Users/Caleb/APC524/stoched/src/midpointleap.h File Reference	50
8.9.1	Detailed Description	50
8.10	/Users/Caleb/APC524/stoched/src/model.cc File Reference	50
8.10.1	Detailed Description	51
8.11	/Users/Caleb/APC524/stoched/src/model.h File Reference	51
8.11.1	Detailed Description	51
8.12	/Users/Caleb/APC524/stoched/src/modeltests.cc File Reference	52
8.12.1	Detailed Description	52
8.13	/Users/Caleb/APC524/stoched/src/nextreaction.cc File Reference	52
8.13.1	Detailed Description	52
8.14	/Users/Caleb/APC524/stoched/src/nextreaction.h File Reference	53
8.14.1	Detailed Description	53
8.15	/Users/Caleb/APC524/stoched/src/paramset.cc File Reference	53

8.15.1 Detailed Description . . . . .	53
8.16 /Users/Caleb/APC524/stoched/src/paramset.h File Reference . . . . .	54
8.16.1 Detailed Description . . . . .	54
8.17 /Users/Caleb/APC524/stoched/src/parsertests.cc File Reference . . . . .	54
8.17.1 Detailed Description . . . . .	55
8.18 /Users/Caleb/APC524/stoched/src/realization.cc File Reference . . . . .	55
8.18.1 Detailed Description . . . . .	55
8.19 /Users/Caleb/APC524/stoched/src/realization.h File Reference . . . . .	55
8.19.1 Detailed Description . . . . .	56
8.20 /Users/Caleb/APC524/stoched/src/realization_factory.cc File Reference . . . . .	56
8.20.1 Detailed Description . . . . .	56
8.21 /Users/Caleb/APC524/stoched/src/realization_factory.h File Reference . . . . .	57
8.21.1 Detailed Description . . . . .	57
8.22 /Users/Caleb/APC524/stoched/src/rng.cc File Reference . . . . .	57
8.22.1 Detailed Description . . . . .	58
8.22.2 poisson_ptrs(double mean) implements the PTRS . . . . .	58
8.23 /Users/Caleb/APC524/stoched/src/rng.h File Reference . . . . .	58
8.23.1 Detailed Description . . . . .	59
8.24 /Users/Caleb/APC524/stoched/src/simulate.cc File Reference . . . . .	59
8.24.1 Detailed Description . . . . .	60
8.24.2 Function Documentation . . . . .	60
8.24.2.1 main() . . . . .	60
8.25 /Users/Caleb/APC524/stoched/src/tauleapavail.cc File Reference . . . . .	60
8.25.1 Detailed Description . . . . .	61
8.25.2 Function Documentation . . . . .	61
8.25.2.1 tauLeapAvail() . . . . .	61
8.26 /Users/Caleb/APC524/stoched/src/testevent.cc File Reference . . . . .	61
8.26.1 Detailed Description . . . . .	62
8.26.2 Function Documentation . . . . .	62
8.26.2.1 main() . . . . .	62



8.27 /Users/Caleb/APC524/stoched/src/testmodel.cc File Reference . . . . .	62
8.27.1 Detailed Description . . . . .	63
8.27.2 Function Documentation . . . . .	63
8.27.2.1 main() . . . . .	63
8.28 /Users/Caleb/APC524/stoched/src/testparser.cc File Reference . . . . .	63
8.28.1 Detailed Description . . . . .	64
8.28.2 Function Documentation . . . . .	64
8.28.2.1 main() . . . . .	64
8.29 /Users/Caleb/APC524/stoched/src/testsimulate.cc File Reference . . . . .	64
8.29.1 Detailed Description . . . . .	65
8.30 /Users/Caleb/APC524/stoched/src/xoroshiro128plus.cc File Reference . . . . .	65
8.30.1 Detailed Description . . . . .	65
8.31 /Users/Caleb/APC524/stoched/src/xoroshiro128plus.h File Reference . . . . .	66
8.31.1 Detailed Description . . . . .	66
<b>Index</b>	<b>67</b>



# Chapter 1

## stoched

Welcome to **stoched**, an application developed at Princeton University for modeling stochastic systems modeled by rate equations and simulating reactions from them!

Getting started information for **stoched** (including build instructions) is located in its interior doc/README.md file.

### Introduction

The platform is a fast, compiled code tool with an extremely simple interface aimed towards scientists with minimal programming experience. While other tools for stochastic modeling and simulation exist, none have non-programmer-friendly interfaces and few are specialized to those systems modeled by rate equations alone. We take user-friendly modeling languages developed for Bayesian inference (BUGS/JAGS and Stan) as guides.

Stoched implements the Gillespie algorithm to perform exact simulations. Also, more scalable approximate algorithms derived from the Gillespie algorithm are useful for large systems. These algorithms have historically been used to solve problems in molecular dynamics; today, they are applied to a wide variety of stochastic modeling problems.

### Platforms

- Linux
- Mac OS X
- Windows

### Requirements

#### Serial Implementation

- None

#### Parallel Implementation (optional)

- [Open MPI](#)

### Requirements for Contributors

- `Flex/Bison`
- `Google Test` (located in interior `lib/` folder; must be built)

### License

Permission to use, copy, modify, and distribute this software and its documentation under the terms of the GNU General Public License is hereby granted. No representations are made about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty. See the GNU General Public License for more details.

Documents produced by Stoched are derivative works derived from the input used in their production; they are not affected by this license.

## Chapter 2

## Authors

### Caleb Peckham

Caleb is a senior undergraduate student at Princeton University, studying Mechanical and Aerospace Engineering. He is currently working on his senior thesis, an application of high-speed object tracking and machine-learning, under Professor Daniel Nosenchuck. He prepared this documentation, wrote the interface between the parser and the realization, and implemented a Google Test suite.

### Dylan Morris

Dylan is a second-year graduate student in the Department of Ecology and Evolutionary Biology at Princeton University. His research interests include the evolution of infectious diseases, statistical inference, and mathematical modeling.

### Kevin Griffin

Kevin is a senior undergraduate student at Princeton University, studying Mechanical and Aerospace Engineering. He is currently working on his senior thesis under Professor Michael Mueller in the area of computational fluid dynamics. He wrote the parser which uses Flex, Bison, and FunctionParser libraries.

### Julienne LaChance

Julie is a first-year graduate student in the MAE Department. She obtained a Master's in Applied Mathematics and bachelors degrees in both Applied Math and Mechanical Engineering from RPI. Julie is advised by Prof. Rowley.



## Chapter 3

# User Guide

### Installation

First go to the [download page](#) to get the latest distribution, if you have not downloaded **stoched** already.

### Download

Download ZIP file at [stoched Github](#)

```
$ unzip stoched-master.zip
$ cd stoched-master
```

### Clone Repository

```
$ git clone https://github.com/APC524/stoched
$ cd stoched
```

### Getting Started

#### Step 1: Creating an input file

The parser uses a custom language that is designed to be accessible to non-programmers. It uses minimal syntax and allows for line comments and end of line comments, and whitespace like new lines between commands. It has been designed to reduce the likelihood of redundant and potentially incorrect information.

A simple input file:

```
SETUP_VARS "a, b"
EVENT RATE "3" "a+1" "b+0"
EVENT RATE "a/3" "a-1" "b+0"

end
```

The first line initializes the variables in the simulation with a comma separated variable list enclosed in quotes. Variable names contain the characters A-Z, a-z, and 0-9. The first character, however must be A-Z or a-z. Note that underscores and spaces are not allowed and that the variable list must not contain spaces.

The next line is an event line. An input file can contain as many event lines as needed. In an abstract sense an event consists of the likelihood of the event occurring and a definition of how it changes the system when it occurs. Practically, an event is a rate function followed by any number of equations that involve the variables in the variable list. The rate function and following event functions can contain nonlinear expressions, support the mathematical symbols  $+$   $-$   $*$   $/$   $($   $)$ , and can contain white space between symbols.

The last line is end. This indicates the end of the file

A more complicated input file with multiple, nonlinear events:

```
SETUP_VARS "a,b,c"
EVENT RATE "a" "a * (1 + c)" "b - 0.4" "0.56"
EVENT RATE "a*b/d" "a" "b-c" "a*b*c"

end
```

The number of variables and the number of events does not have to be the same. But the number of event functions per event must always equal the number of variables in the variable string. In this example there are three variables: a,b,c; therefore, there are always 4 equations in the EVENT. The first one is the rate function and the last three indicate how the three variables are modified. So the first event occurs at a rate equal to a, and when it occurs it sets  $a = a * (1 + c)$ , it sets  $b = b - 0.4$ , and it sets  $c = 0.56$ .

Finer points: Note the syntax "a+1" and "a + 1" are both acceptable. Scientific notation is not yet supported

Lines can be commented by placing a # character. The parser ignores all text on a line after a # character. This means that it can be used to comment out a whole line if it is placed at the beginning of a line, or used to add a note at the end of a line

Comment Example:

```
# This code is now well commented
SETUP_VARS "a,b"
# EVENT RATE "2" "a - 5" "b - 5"
EVENT RATE "3" "a + 1" "b + 1"      # I've added a comment here to explain why the rate is 3
end
```

In the above example the first event will be ignored because the line begins with #. When the second line is parsed, only the text "I've added ..." is removed. This input file is functionally equivalent to the first example input file, but it is more readable by human because it had comments.

## Step 2: Running stoched

### Compiling Serial Code

```
stoched-master$ cd src
stoched-master/src$ make
```

### Sample Execution of Serial Code

```
stoched-master$ cd examples
stoched-master/examples$ ../src/stoched.exe chem.parser.in init_file init_file.txt
```



## Compiling Parallel Code

Assumes installation of OpenMPI

```
stoched-master$ cd src
stoched-master/src$ make parallel
```

## Sample Execution of Parallel Code

For usage on Adroit.

```
stoched-master$ cd examples
stoched-master/examples$ sbatch run_mpi.slurm
```

## Compiling Test Code

Assumes installation of Google Test suite

```
stoched-master$ cd src
stoched-master/src$ make googletests
```

## Sample Execution of Test Code

```
stoched-master$ cd src
stoched-master/src$ ./testmodel.exe
```

## Step 3: Parameters

To specify additional parameters, the user may include additional command line arguments, which are listed below. For example, to run the simulation 5 times, the command would look like this:

```
stoched-master/src$ ./stoched.exe example.parser.in n_realizations 5
```

The command line arguments are as follows:

**init\_file:** Required for specifying the initial states data for most model definitions. The exception is a model definition with two species which each start with zero population. This is the default initial state.

**method:** specifies which algorithm is used to perform computations. Specifying 0 will run the exact Gillespie algorithm, while 1 will run the Euler tau-leap method, and 2 will run the midpoint tau-leap method. Default is 0.

**t\_initial:** allows user to modify the starting time of the simulation. Default is 0 .

**t\_final:** allows user to modify the end time of the simulation. Default is 5000 .

**timestep\_size:** allows user to fix timestep size if desired.

**n\_realizations:** allows user to run the simulation multiple times with the same model and model conditions. Default value is 1.

**max\_iter:** allows user to specify maximum number of iterations. Default value is 100000000.

**seed:** allows user to fix a seed of the random number generator (which allows for verification of consistency between multiple runs, and with external software results). The default value is 502.

**out\_path:** Allows user to specify an alternative output file path name. The default value is stoched\_output, which will write to a file named stoched\_output.txt . The extension is not required when specifying pathname.

**suppress\_print:** Option specified as either a 0 or 1. If 1, the software prints only the final value of the simulation to each output file. If 0 (default value), the software runs as usual, printing the results at each timestep. Specifying 1 results in significant speedup of the code.

## Step 4: Access Documentation

For further information, visit the Doxygen-generated documentation. The associated HTML documentation can be viewed by pointing a HTML browser to the index.html file in the doc/html directory

To see a PDF version of the documentation, open the refman.pdf file located the doc/latex directory



## Chapter 4

# Hierarchical Index

### 4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Event . . . . .	17
Model . . . . .	26
Paramset . . . . .	35
Realization . . . . .	36
EulerLeap . . . . .	15
FirstReaction . . . . .	21
MidpointLeap . . . . .	23
NextReaction . . . . .	32
RealizationFactory . . . . .	39
rng . . . . .	40
xoroshiro128plus . . . . .	42



## Chapter 5

# Class Index

### 5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">EulerLeap</a>	Class <a href="#">EulerLeap</a> implements <a href="#">Realization step()</a> function using the basic tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations . . . . .	15
<a href="#">Event</a>	Class <a href="#">Event</a> holds a user-specified event, namely set of functions and associated rate . . . . .	17
<a href="#">FirstReaction</a>	Class <a href="#">FirstReaction</a> implements <a href="#">Realization step()</a> function using the exact First Reaction algorithm of Gillespie (1971) . . . . .	21
<a href="#">MidpointLeap</a>	Class <a href="#">MidpointLeap</a> implements <a href="#">Realization step()</a> function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations . . . . .	23
<a href="#">Model</a>	Class <a href="#">Model</a> , which holds user-specified models of stochastic systems from which realizations are to be simulated. A model may have variable parameters; each complete set will be stored in an object of class <a href="#">Paramset</a> . . . . .	26
<a href="#">NextReaction</a>	Class <a href="#">NextReaction</a> implements <a href="#">Realization step()</a> function using the exact Next Reaction algorithm of Gibson & Bruck (2000) . . . . .	32
<a href="#">Paramset</a>	Class <a href="#">Paramset</a> holds a particular set of parameters for user requested simulation run(s) . . . . .	35
<a href="#">Realization</a>	Class <a href="#">Realization</a> holds realizations of a <a href="#">Model</a> (state array, propensities, waiting times, etc.) . . . . .	36
<a href="#">RealizationFactory</a>	Class <a href="#">RealizationFactory</a> generates required instance of <a href="#">Realization</a> ( <a href="#">FirstReaction</a> , <a href="#">NextReaction</a> , <a href="#">EulerLeap</a> , <a href="#">MidpointLeap</a> ) based on input . . . . .	39
<a href="#">rng</a>	Class <a href="#">rng</a> implements random number generator, based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ( <a href="mailto:vigna@acm.org">vigna@acm.org</a> ) . . . . .	40
<a href="#">xoroshiro128plus</a>	Class <a href="#">xoroshiro128plus</a> implements a random number generator of Class <a href="#">rng</a> . . . . .	42



## Chapter 6

# File Index

### 6.1 File List

Here is a list of all documented files with brief descriptions:

/Users/Caleb/APC524/stoched/src/ <a href="#">eulerleap.cc</a>	
Class <a href="#">EulerLeap</a> implements <a href="#">Realization</a> step() function using the basic tau leap approximate method of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations . . . . .	45
/Users/Caleb/APC524/stoched/src/ <a href="#">eulerleap.h</a>	
Class <a href="#">EulerLeap</a> implements <a href="#">Realization</a> step() function using the basic tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations . . . . .	45
/Users/Caleb/APC524/stoched/src/ <a href="#">event.cc</a>	
Class <a href="#">Event</a> holds a user-specified event, namely set of functions and associated rate . . . . .	46
/Users/Caleb/APC524/stoched/src/ <a href="#">event.h</a>	
Class <a href="#">Event</a> holds a user-specified event, namely set of functions and associated rate . . . . .	47
/Users/Caleb/APC524/stoched/src/ <a href="#">eventtests.cc</a>	
Test <a href="#">Event</a> code . . . . .	47
/Users/Caleb/APC524/stoched/src/ <a href="#">firstreaction.cc</a>	
Class <a href="#">FirstReaction</a> implements <a href="#">Realization</a> step() function using the exact First Reaction algorithm of Gillespie (1971) . . . . .	48
/Users/Caleb/APC524/stoched/src/ <a href="#">firstreaction.h</a>	
Class <a href="#">FirstReaction</a> implements <a href="#">Realization</a> step() function using the exact First Reaction algorithm of Gillespie (1971) . . . . .	48
/Users/Caleb/APC524/stoched/src/ <a href="#">midpointleap.cc</a>	
Class <a href="#">MidpointLeap</a> implements <a href="#">Realization</a> step() function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations . . . . .	49
/Users/Caleb/APC524/stoched/src/ <a href="#">midpointleap.h</a>	
Class <a href="#">MidpointLeap</a> implements <a href="#">Realization</a> step() function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations . . . . .	50
/Users/Caleb/APC524/stoched/src/ <a href="#">model.cc</a>	
Class <a href="#">Model</a> , which holds user-specified models of stochastic systems from which realizations are to be simulated . . . . .	50
/Users/Caleb/APC524/stoched/src/ <a href="#">model.h</a>	
Class <a href="#">Model</a> , which holds user-specified models of stochastic systems from which realizations are to be simulated . . . . .	51
/Users/Caleb/APC524/stoched/src/ <a href="#">modeltests.cc</a>	
Test usage of <a href="#">Model</a> class . . . . .	52

/Users/Caleb/APC524/stoched/src/ <a href="#">nextreaction.cc</a>	
Class <a href="#">NextReaction</a> implements <a href="#">Realization</a> step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000) . . . . .	52
/Users/Caleb/APC524/stoched/src/ <a href="#">nextreaction.h</a>	
Class <a href="#">nextreaction</a> implements <a href="#">Realization</a> step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000) . . . . .	53
/Users/Caleb/APC524/stoched/src/ <a href="#">paramset.cc</a>	
Class <a href="#">Paramset</a> holds a particular set of parameters for user requested simulation run(s) . . . .	53
/Users/Caleb/APC524/stoched/src/ <a href="#">paramset.h</a>	
Class <a href="#">Paramset</a> holds a particular set of parameters for user requested simulation run(s) . . . .	54
/Users/Caleb/APC524/stoched/src/ <a href="#">parser.bison.hh</a> . . . . .	??
/Users/Caleb/APC524/stoched/src/ <a href="#">parsertests.cc</a>	
Example parsing code . . . . .	54
/Users/Caleb/APC524/stoched/src/ <a href="#">realization.cc</a>	
Class <a href="#">Realization</a> holds realizations of a <a href="#">Model</a> (state array, propensities, waiting times, etc.) .	55
/Users/Caleb/APC524/stoched/src/ <a href="#">realization.h</a>	
Class <a href="#">Realization</a> holds realizations of a <a href="#">Model</a> (state array, propensities, waiting times, etc.) .	55
/Users/Caleb/APC524/stoched/src/ <a href="#">realization_factory.cc</a>	
Class <a href="#">RealizationFactory</a> generates required instance of <a href="#">Realization</a> ( <a href="#">FirstReaction</a> , <a href="#">NextReaction</a> , <a href="#">EulerLeap</a> ) based on input . . . . .	56
/Users/Caleb/APC524/stoched/src/ <a href="#">realization_factory.h</a>	
Class <a href="#">RealizationFactory</a> generates required instance of <a href="#">Realization</a> ( <a href="#">FirstReaction</a> , <a href="#">NextReaction</a> , <a href="#">EulerLeap</a> ) based on input . . . . .	57
/Users/Caleb/APC524/stoched/src/ <a href="#">rng.cc</a>	
Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ( <a href="#">vigna@acm.org</a> ) . . . . .	57
/Users/Caleb/APC524/stoched/src/ <a href="#">rng.h</a>	
Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ( <a href="#">vigna@acm.org</a> ) . . . . .	58
/Users/Caleb/APC524/stoched/src/ <a href="#">simulate.cc</a>	
Example definition for ending simulation loop . . . . .	59
/Users/Caleb/APC524/stoched/src/ <a href="#">tauleapavail.cc</a>	
Called by parser.y to determine whether to use tau leap or not . . . . .	60
/Users/Caleb/APC524/stoched/src/ <a href="#">testevent.cc</a>	
Example usage of <a href="#">Event</a> class . . . . .	61
/Users/Caleb/APC524/stoched/src/ <a href="#">testmodel.cc</a>	
Example usage of <a href="#">Model</a> class . . . . .	62
/Users/Caleb/APC524/stoched/src/ <a href="#">testparser.cc</a>	
Example parsing code . . . . .	63
/Users/Caleb/APC524/stoched/src/ <a href="#">testsimulate.cc</a>	
Example simulation code . . . . .	64
/Users/Caleb/APC524/stoched/src/ <a href="#">xoroshiro128plus.cc</a>	
Class <a href="#">xoroshiro128plus</a> implements a random number generator of Class <a href="#">rng</a> . . . . .	65
/Users/Caleb/APC524/stoched/src/ <a href="#">xoroshiro128plus.h</a>	
Class <a href="#">xoroshiro128plus</a> implements a random number generator of Class <a href="#">rng</a> . . . . .	66



## Chapter 7

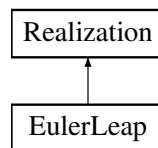
# Class Documentation

### 7.1 EulerLeap Class Reference

Class [EulerLeap](#) implements [Realization step\(\)](#) function using the basic tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

```
#include <eulerleap.h>
```

Inheritance diagram for EulerLeap:



#### Public Member Functions

- [EulerLeap](#) ([Model](#) \*[the\\_model](#), const [Paramset](#) &[the\\_paramset](#), [rng](#) \*[the\\_rng](#), int [n\\_vars](#), int [n\\_events](#))  
*Default constructor for [EulerLeap](#).*
- [~EulerLeap](#) ()  
*Destructor for [MidpointLeap](#).*
- int [step](#) ()  
*Update waiting times.*
- int [set\\_to\\_initial\\_state](#) ()  
*Sets [state\\_array](#) and [state\\_time](#) to their user-specified initial values.*

#### Additional Inherited Members

##### 7.1.1 Detailed Description

Class [EulerLeap](#) implements [Realization step\(\)](#) function using the basic tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

## 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 EulerLeap()

```
EulerLeap::EulerLeap (
    Model * the_model,
    const Paramset & the_paramset,
    rng * the_rng,
    int n_vars,
    int n_events )
```

Default constructor for [EulerLeap](#).

#### Parameters

<i>the_model</i>	is a <a href="#">Model</a> object
<i>the_paramset</i>	is a <a href="#">Paramset</a> object
<i>the_rng</i>	is a random number generator
<i>n_vars</i>	is an int specifying variable count
<i>n_events</i>	is an int specifying event count

#### Returns

nothing

### 7.1.2.2 ~EulerLeap()

```
EulerLeap::~EulerLeap ( )
```

Destructor for [MidpointLeap](#).

#### Returns

nothing

## 7.1.3 Member Function Documentation

7.1.3.1 `set_to_initial_state()`

```
int EulerLeap::set_to_initial_state ( ) [virtual]
```

Sets `state_array` and `state_time` to their user-specified initial values.

**Returns**

int

Reimplemented from [Realization](#).

7.1.3.2 `step()`

```
int EulerLeap::step ( ) [virtual]
```

Update waiting times.

**Returns**

int

Implements [Realization](#).

The documentation for this class was generated from the following files:

- `/Users/Caleb/APC524/stoched/src/eulerleap.h`
- `/Users/Caleb/APC524/stoched/src/eulerleap.cc`

## 7.2 Event Class Reference

Class [Event](#) holds a user-specified event, namely set of functions and associated rate.

```
#include <event.h>
```

**Public Member Functions**

- [Event](#) ()  
*Default constructor for [Event](#).*
- [~Event](#) ()  
*Destructor of [Event](#).*
- void [addFunction](#) (string function, string variables)  
*Add a function parser to the function array.*
- double [useFunction](#) (int iFunction, double \*args)  
*Evaluate function stored at specified spot in the function array.*
- void [setRate](#) (string function, string variables)  
*Set equation for rateFunction.*
- double [getRate](#) (double \*args)  
*Return rate to user based on values of the state array.*
- int [getSize](#) ()  
*Return size of event, namely number of functions, to user.*
- double [getDeltaVar](#) (int i)  
*Return how the ith variable is incremented when the ith equation is called.*
- void [setDeltaVar](#) (int i, double val)  
*set the amount that the ith function increments the ith variable. This is used by midpoint tau leaping*

## Public Attributes

- string **eventName**

## Private Attributes

- FunctionParser \*\* [functionArray\\_](#)  
*Array of function parsers.*
- FunctionParser [rateFunction](#)  
*Rate specified by an equation.*
- int [eq\\_count\\_](#)  
*Number of function parsers.*
- double \* [deltaVar\\_](#)  
*how much each variable in the state changes when its corresponding function is called. Only used by midpoint tau leaping to calculate approximate continuous time derivative.*

### 7.2.1 Detailed Description

Class [Event](#) holds a user-specified event, namely set of functions and associated rate.

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 Event()

```
Event::Event ( )
```

Default constructor for [Event](#).

#### Parameters

<i>eq_count</i>	is the size of the function array
<i>functionArray</i>	contains all user-specified FunctionParsers that govern event

#### Returns

nothing

#### 7.2.2.2 ~Event()

```
Event::~~Event ( )
```

Destructor of [Event](#).

**Returns**

nothing

**7.2.3 Member Function Documentation****7.2.3.1 addFunction()**

```
void Event::addFunction (
    string function,
    string variables )
```

Add a function parser to the function array.

**Parameters**

<i>function</i>	is a string used to generate a FunctionParser object
<i>variables</i>	is a string used to generate a FunctionParser object

**Returns**

void

**7.2.3.2 getDeltaVar()**

```
double Event::getDeltaVar (
    int i )
```

Return how the ith variable is incremented when the ith equation is called.

**Parameters**

<i>i</i>	is an int specifying of which variable to find the delta. 0 is the first variable
----------	---

**Returns**

change in value of i when its corresponding equation is called, as a double

**7.2.3.3 getRate()**

```
double Event::getRate (
    double * stateArray )
```

Return rate to user based on values of the state array.

**Parameters**

<i>stateArray</i>	is a double array specifying variable values of function
-------------------	--

**Returns**

evaluated rateFunction as a double

**7.2.3.4 getSize()**

```
int Event::getSize ( )
```

Return size of event, namely number of functions, to user.

**Returns**

size of event, namely number of functions, as an int

**7.2.3.5 setDeltaVar()**

```
void Event::setDeltaVar (
    int i,
    double val )
```

set the amount that the ith function increments the ith variable. This is used by midpoint tau leaping

**Parameters**

<i>i</i>	is an int specifying of which variable to set. 0 is the first variable
----------	--

**Returns**

void

**7.2.3.6 setRate()**

```
void Event::setRate (
    string function,
    string variables )
```

Set equation for rateFunction.

## Parameters

<i>function</i>	is a string used for parsing rateFunction
<i>variables</i>	is a string used for parsing rateFunction

## Returns

void

## 7.2.3.7 useFunction()

```
double Event::useFunction (
    int iFunction,
    double * stateArray )
```

Evaluate function stored at specified spot in the function array.

## Parameters

<i>iFunction</i>	is an int that indexes function array
<i>stateArray</i>	is a double array specifying variable values of function

## Returns

evaluated functionParser as a double

The documentation for this class was generated from the following files:

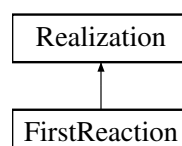
- /Users/Caleb/APC524/stoched/src/[event.h](#)
- /Users/Caleb/APC524/stoched/src/[event.cc](#)

## 7.3 FirstReaction Class Reference

Class [FirstReaction](#) implements [Realization step\(\)](#) function using the exact First Reaction algorithm of Gillespie (1971)

```
#include <firstreaction.h>
```

Inheritance diagram for FirstReaction:



## Public Member Functions

- [FirstReaction](#) ([Model](#) \*[the\\_model](#), const [Paramset](#) &[the\\_paramset](#), [rng](#) \*[the\\_rng](#), int [n\\_vars](#), int [n\\_events](#))  
*Default constructor for [FirstReaction](#).*
- [~FirstReaction](#) ()  
*Destructor for [FirstReaction](#).*
- int [step](#) ()  
*Update waiting times.*

## Private Attributes

- double \* [waiting\\_times](#)  
*pause*

## Additional Inherited Members

### 7.3.1 Detailed Description

Class [FirstReaction](#) implements [Realization step\(\)](#) function using the exact First Reaction algorithm of Gillespie (1971)

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 FirstReaction()

```
FirstReaction::FirstReaction (
    Model * the\_model,
    const Paramset & the\_paramset,
    rng * the\_rng,
    int n\_vars,
    int n\_events )
```

Default constructor for [FirstReaction](#).

#### Parameters

<i><a href="#">the_model</a></i>	is a <a href="#">Model</a> object
<i><a href="#">the_paramset</a></i>	is a <a href="#">Paramset</a> object
<i><a href="#">the_rng</a></i>	is a random number generator
<i><a href="#">n_vars</a></i>	is an int specifying variable count
<i><a href="#">n_events</a></i>	is an int specifying event count



**Returns**

nothing

**7.3.2.2 ~FirstReaction()**

```
FirstReaction::~~FirstReaction ( )
```

Destructor for [FirstReaction](#).

**Returns**

nothing

**7.3.3 Member Function Documentation****7.3.3.1 step()**

```
int FirstReaction::step ( ) [virtual]
```

Update waiting times.

**Returns**

int

Implements [Realization](#).

The documentation for this class was generated from the following files:

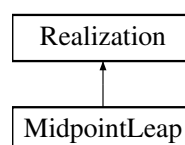
- [/Users/Caleb/APC524/stoched/src/firstreaction.h](#)
- [/Users/Caleb/APC524/stoched/src/firstreaction.cc](#)

**7.4 MidpointLeap Class Reference**

Class [MidpointLeap](#) implements [Realization step\(\)](#) function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

```
#include <midpointleap.h>
```

Inheritance diagram for MidpointLeap:



## Public Member Functions

- [MidpointLeap](#) ([Model](#) \*[the\\_model](#), const [Paramset](#) &[the\\_paramset](#), [rng](#) \*[the\\_rng](#), int [n\\_vars](#), int [n\\_events](#))  
*Default constructor for [MidpointLeap](#).*
- [~MidpointLeap](#) ()  
*Destructor for [MidpointLeap](#).*
- int [step](#) ()  
*Update waiting times.*
- int [set\\_to\\_initial\\_state](#) ()  
*Sets [state\\_array](#) and [state\\_time](#) to their user-specified initial values.*

## Private Attributes

- double \* [midpoint\\_array\\_](#)  
*[midpoint\\_array\\_](#) is a double array holding the current midpoints for the tau-leap step*

## Additional Inherited Members

### 7.4.1 Detailed Description

Class [MidpointLeap](#) implements [Realization step\(\)](#) function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 MidpointLeap()

```
MidpointLeap::MidpointLeap (
    Model * the\_model,
    const Paramset & the\_paramset,
    rng * the\_rng,
    int n\_vars,
    int n\_events )
```

Default constructor for [MidpointLeap](#).

#### Parameters

<i><a href="#">the_model</a></i>	is a <a href="#">Model</a> object
<i><a href="#">the_paramset</a></i>	is a <a href="#">Paramset</a> object
<i><a href="#">the_rng</a></i>	is a random number generator
<i><a href="#">n_vars</a></i>	is an int specifying variable count
<i><a href="#">n_events</a></i>	is an int specifying event count

**Returns**

nothing

initialize midpoint array

**7.4.2.2 ~MidpointLeap()**

```
MidpointLeap::~MidpointLeap ( )
```

Destructor for [MidpointLeap](#).

**Returns**

nothing

**7.4.3 Member Function Documentation****7.4.3.1 set\_to\_initial\_state()**

```
int MidpointLeap::set_to_initial_state ( ) [virtual]
```

Sets state\_array and state\_time to their user-specified initial values.

**Returns**

int

Reimplemented from [Realization](#).

**7.4.3.2 step()**

```
int MidpointLeap::step ( ) [virtual]
```

Update waiting times.

**Returns**

int

Implements [Realization](#).

The documentation for this class was generated from the following files:

- [/Users/Caleb/APC524/stoched/src/midpointleap.h](#)
- [/Users/Caleb/APC524/stoched/src/midpointleap.cc](#)

## 7.5 Model Class Reference

Class [Model](#), which holds user-specified models of stochastic systems from which realizations are to be simulated. A model may have variable parameters; each complete set will be stored in an object of class [Paramset](#).

```
#include <model.h>
```

### Public Member Functions

- [Model](#) ()
- [~Model](#) ()
- void [addVars](#) (string vars)
- void [addEvent](#) (string functionRate)
- void [addEventFct](#) (int iEvent, string function)
  - Add [Event](#) function to specified [Event](#) in [Model](#).*
- void [setTauLeapFalse](#) ()
  - Indicate tau leaping is impermissible by setting the tauLeapAvail\_ flag to false.*
- bool [checkTauLeapAvail](#) ()
  - Check if events are compatible with tau leaping.*
- double [useEventFct](#) (int iEvent, int iFunction, double \*stateArray)
  - Evaluate given function in specified [Event](#).*
- string [getVarsString](#) ()
  - Get a list of the variables that make up the state array.*
- double [getEventRate](#) (int iEvent, double \*stateArray)
  - Evaluate rate function for a specified [Event](#).*
- int [getVarsCount](#) ()
  - Return total number of variables.*
- int [getEventsCount](#) ()
  - Return total number of Events.*
- string [getIthVar](#) (int index)
  - Returns the ith variable in the variable list.*
- double [getContDeriv](#) (int whichVar, double \*stateArray)
  - Returns the continuous time derivative of specified variable.*
- void [setDelta](#) (int whichVar, int whichEvent, double val)
  - Sets the delta of a specified variable in the specified event. This delta is needed to compute the continuous time derivative.*
- void [updateState](#) (int iEvent, double \*stateArray)
  - Update state array by evaluating all functions of a given [Event](#).*
- void [updateRates](#) (double \*stateArray, double \*rateArray)
  - Update rate for all Events in [Model](#)'s [Event](#) list.*

### Private Attributes

- vector< [Event](#) \* > [eventPtrList](#)
  - List of Events in [Model](#).*
- string [vars\\_](#)
  - Variables associated with a model.*
- bool [tauLeapAvail\\_](#)

### 7.5.1 Detailed Description

Class [Model](#), which holds user-specified models of stochastic systems from which realizations are to be simulated. A model may have variable parameters; each complete set will be stored in an object of class [Paramset](#).

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 Model()

```
Model::Model ( )
```

Default constructor for [Model](#)

##### Returns

nothing

#### 7.5.2.2 ~Model()

```
Model::~~Model ( )
```

Destructor of [Model](#)

##### Returns

nothing

### 7.5.3 Member Function Documentation

#### 7.5.3.1 addEvent()

```
void Model::addEvent (
    string functionRate )
```

Add [Event](#) to [Model](#)'s list of Events

##### Parameters

<i>functionRate</i>	is a string that defines an <a href="#">Event</a> 's rate
---------------------	---

**Returns**

void

**7.5.3.2 addEventFct()**

```
void Model::addEventFct (
    int iEvent,
    string function )
```

Add [Event](#) function to specified [Event](#) in [Model](#).

**Parameters**

<i>iEvent</i>	is an int that indexes <a href="#">Event</a> list
<i>function</i>	is a string that specifies <a href="#">Event</a> function

**Returns**

void

**7.5.3.3 addVars()**

```
void Model::addVars (
    string vars )
```

Add variable list to a [Model](#)

**Parameters**

<i>vars</i>	is a string used to set variables associate with a <a href="#">Model</a>
-------------	--

**Returns**

void

**7.5.3.4 checkTauLeapAvail()**

```
bool Model::checkTauLeapAvail ( )
```

Check if events are compatible with tau leaping.

**Returns**

the status of if tau leaping is available as a boolean

### 7.5.3.5 getContDeriv()

```
double Model::getContDeriv (
    int whichVar,
    double * stateArray )
```

Returns the continuous time derivative of specified variable.

#### Parameters

<i>int</i>	specifying which variable from the state array to find derivative
<i>double</i>	array of the entire state array

#### Returns

ith continuous derivative as a double

### 7.5.3.6 getEventRate()

```
double Model::getEventRate (
    int iEvent,
    double * stateArray )
```

Evaluate rate function for a specified [Event](#).

#### Parameters

<i>iEvent</i>	is an int that indexes <a href="#">Event</a> list
<i>stateArray</i>	is a double array specifying variable values of a function

#### Returns

evaluated rate function as a double

### 7.5.3.7 getEventsCount()

```
int Model::getEventsCount ( )
```

Return total number of Events.

#### Returns

int

#### 7.5.3.8 getIthVar()

```
string Model::getIthVar (
    int index )
```

Returns the ith variable in the variable list.

##### Parameters

<i>index</i>	is an int that indexes variable list
--------------	--------------------------------------

##### Returns

ith variable as string

#### 7.5.3.9 getVarsCount()

```
int Model::getVarsCount ( )
```

Return total number of variables.

##### Returns

int

#### 7.5.3.10 getVarsString()

```
string Model::getVarsString ( )
```

Get a list of the variables that make up the state array.

##### Returns

comma separated list of variable names as a string

#### 7.5.3.11 setDelta()

```
void Model::setDelta (
    int whichVar,
    int whichEvent,
    double val )
```

Sets the delta of a specified variable in the specified event. This delta is needed to compute the continuous time derivative.



## Parameters

<i>int</i>	specifying to which variable from the state array the delta corresponds
<i>int</i>	specifying to which event the delta corresponds
<i>double</i>	specifying the value of delta

## Returns

void

## 7.5.3.12 setTauLeapFalse()

```
void Model::setTauLeapFalse ( )
```

Indicate tau leaping is impermissible by setting the tauLeapAvail\_ flag to false.

## Returns

void

## 7.5.3.13 updateRates()

```
void Model::updateRates (
    double * stateArray,
    double * rateArray )
```

Update rate for all Events in [Model's Event](#) list.

## Parameters

<i>stateArray</i>	is a double array specifying variable values of a function
<i>rateArray</i>	is a double array specifying variable values of a rate function

## Returns

void

## 7.5.3.14 updateState()

```
void Model::updateState (
    int iEvent,
    double * stateArray )
```

Update state array by evaluating all functions of a given [Event](#).

**Parameters**

<i>iEvent</i>	is an int that indexes <a href="#">Event</a> list
<i>stateArray</i>	is a double array specifying variable values of a function

**Returns**

void

**7.5.3.15 useEventFct()**

```
double Model::useEventFct (
    int iEvent,
    int iFunction,
    double * stateArray )
```

Evaluate given function in specified [Event](#).

**Parameters**

<i>iEvent</i>	is an int that indexes <a href="#">Event</a> list
<i>iFunction</i>	is an int that indexes an <a href="#">Event</a> 's Function list
<i>stateArray</i>	is a double array specifying variable values of a function

**Returns**

evaluated function as a double

The documentation for this class was generated from the following files:

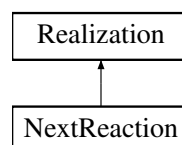
- [/Users/Caleb/APC524/stoched/src/model.h](#)
- [/Users/Caleb/APC524/stoched/src/model.cc](#)

**7.6 NextReaction Class Reference**

Class [NextReaction](#) implements [Realization step\(\)](#) function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

```
#include <nextreaction.h>
```

Inheritance diagram for NextReaction:



## Public Member Functions

- [NextReaction](#) ([Model](#) \*[the\\_model](#), const [Paramset](#) &[the\\_paramset](#), [rng](#) \*[the\\_rng](#), int [n\\_vars](#), int [n\\_events](#))  
*Default constructor for [NextReaction](#).*
- [~NextReaction](#) ()  
*Destructor for [NextReaction](#).*
- int [step](#) ()  
*Update waiting times.*
- int [set\\_to\\_initial\\_state](#) ()  
*Sets [state\\_array](#) and [state\\_time](#) to their user-specified initial values.*

## Private Attributes

- double \* [waiting\\_times](#)  
*pause*

## Additional Inherited Members

### 7.6.1 Detailed Description

Class [NextReaction](#) implements [Realization step\(\)](#) function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 NextReaction()

```
NextReaction::NextReaction (
    Model * the\_model,
    const Paramset & the\_paramset,
    rng * the\_rng,
    int n\_vars,
    int n\_events )
```

Default constructor for [NextReaction](#).

#### Parameters

<i>the_model</i>	is a <a href="#">Model</a> object
<i>the_paramset</i>	is a <a href="#">Paramset</a> object
<i>the_rng</i>	is a random number generator
<i>n_vars</i>	is an int specifying variable count
<i>n_events</i>	is an int specifying event count

**Returns**

nothing

**7.6.2.2 ~NextReaction()**

```
NextReaction::~~NextReaction ( )
```

Destructor for [NextReaction](#).

**Returns**

nothing

**7.6.3 Member Function Documentation****7.6.3.1 set\_to\_initial\_state()**

```
int NextReaction::set_to_initial_state ( ) [virtual]
```

Sets state\_array and state\_time to their user-specified initial values.

**Returns**

int

Reimplemented from [Realization](#).

**7.6.3.2 step()**

```
int NextReaction::step ( ) [virtual]
```

Update waiting times.

**Returns**

int

Implements [Realization](#).

The documentation for this class was generated from the following files:

- /Users/Caleb/APC524/stoched/src/[nextreaction.h](#)
- /Users/Caleb/APC524/stoched/src/[nextreaction.cc](#)

## 7.7 Paramset Class Reference

Class [Paramset](#) holds a particular set of parameters for user requested simulation run(s)

```
#include <paramset.h>
```

### Public Member Functions

- [Paramset](#) (int [method](#), int [n\\_vars](#), double \*[initial\\_values](#), double [t\\_initial](#), double [t\\_final](#), double [timestep\\_size](#), int [n\\_realizations](#), int [max\\_iter](#), int [seed](#), int [suppress\\_output](#))

*Default constructor for [Paramset](#).*

- [~Paramset](#) ()

*Destructor for [Paramset](#).*

### Public Attributes

- const int [method](#)  
*which algorithm to use for simulation*
- const int [n\\_vars](#)  
*number of initial values/variables*
- const double \* [initial\\_values](#)  
*initial values for variables*
- const double [t\\_initial](#)  
*initial time for simulation*
- const double [t\\_final](#)  
*final time for simulation*
- const double [timestep\\_size](#)  
*size of timestep for approximate*
- int [n\\_realizations](#)  
*number of realizations to simulate*
- int [max\\_iter](#)  
*max number of iterations to simulate*
- int [seed](#)  
*seed for the random number generator*
- int [suppress\\_output](#)  
*allows user to only print final state value*

### 7.7.1 Detailed Description

Class [Paramset](#) holds a particular set of parameters for user requested simulation run(s)

### 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 Paramset()

```
Paramset::Paramset (
    int method,
    int n_vars,
    double * initial_values,
    double t_initial,
    double t_final,
    double timestep_size,
    int n_realizations,
    int max_iter,
    int seed,
    int suppress_output )
```

Default constructor for [Paramset](#).

#### Parameters

<i>method</i>	is an int that specifies algorithm to use for simulation
<i>n_vars</i>	is an int that specifies number of variables
<i>initial_values</i>	is a double array that sets initial values for variables
<i>t_initial</i>	is a double that sets initial time for simulation
<i>t_final</i>	is a double that sets initial time for simulation
<i>timestep_size</i>	is a double representing the size of the timestep for approximate methods
<i>n_realizations</i>	is an int representing number of realizations to simulate
<i>max_iter</i>	is an int and is the maximum number of iterations to simulate

### 7.7.2.2 ~Paramset()

```
Paramset::~~Paramset ( )
```

Destructor for [Paramset](#).

#### Returns

nothing

The documentation for this class was generated from the following files:

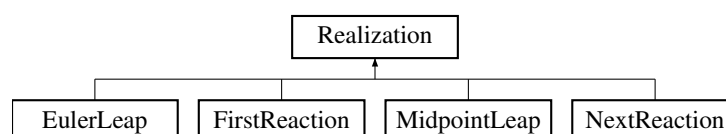
- /Users/Caleb/APC524/stoched/src/[paramset.h](#)
- /Users/Caleb/APC524/stoched/src/[paramset.cc](#)

## 7.8 Realization Class Reference

Class [Realization](#) holds realizations of a [Model](#) (state array, propensities, waiting times, etc.)

```
#include <realization.h>
```

Inheritance diagram for Realization:



## Public Member Functions

- [Realization](#) ([Model](#) \*[the\\_model](#), const [Paramset](#) &[the\\_paramset](#), [rng](#) \*[the\\_rng](#), int [n\\_vars](#), int [n\\_events](#))  
*Default constructor for [Realization](#).*
- virtual [~Realization](#) ()  
*Destructor of [Realization](#).*
- int [simulate](#) (std::ofstream &[myfile](#))  
*Simulates the realization from [t\\_initial](#) to [t\\_final](#).*
- virtual int [step](#) ()=0  
*takes one simulation step according to the chosen algorithm*
- bool [rates\\_are\\_zero](#) ()  
*checks whether all rates are zero*
- int [output\\_state](#) (std::ofstream &[myfile](#))  
*Prints the current state of the simulation.*
- virtual int [set\\_to\\_initial\\_state](#) ()  
*Sets [state\\_array](#) and [state\\_time](#) to their user-specified initial values.*

## Public Attributes

- [Model](#) \* [the\\_model](#)  
*[the\\_model](#) is a [Model](#) instance*
- const [Paramset](#) [the\\_paramset](#)  
*[the\\_paramset](#) is a [Paramset](#) instance*
- [rng](#) \* [the\\_rng](#)  
*[the\\_rng](#) is an random number generator*
- const int [n\\_vars](#)  
*[n\\_vars](#) is an int specifying number of variables*
- const int [n\\_events](#)  
*[n\\_events](#) is an int specifying number of events*
- double \* [state\\_array](#)  
*[state\\_array](#) is a double array specifying variable values of a function*
- double \* [rates](#)  
*[rates](#) is a double array specifying variable values of a rate function*
- double [state\\_time](#)  
*[state\\_time](#) is a double that tracks state progress*

### 7.8.1 Detailed Description

Class [Realization](#) holds realizations of a [Model](#) (state array, propensities, waiting times, etc.)

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 Realization()

```
Realization::Realization (
    Model * the\_model,
    const Paramset & the\_paramset,
    rng * the\_rng,
    int n\_vars,
    int n\_events )
```

Default constructor for [Realization](#).

**Parameters**

<i>the_model</i>	is a <a href="#">Model</a> object
<i>the_paramset</i>	is a <a href="#">Paramset</a> object
<i>the_rng</i>	is a random number generator
<i>n_vars</i>	is an int specifying variable count
<i>n_events</i>	is an int specifying event count

**Returns**

nothing

**7.8.2.2 ~Realization()**

```
Realization::~~Realization ( ) [virtual]
```

Destructor of [Realization](#).

**Returns**

nothing

**7.8.3 Member Function Documentation****7.8.3.1 output\_state()**

```
int Realization::output_state (
    std::ofstream & myfile )
```

Prints the current state of the simulation.

**Returns**

int

**7.8.3.2 rates\_are\_zero()**

```
bool Realization::rates_are_zero ( )
```

checks whether all rates are zero

**Returns**

bool



## 7.8.3.3 set\_to\_initial\_state()

```
int Realization::set_to_initial_state ( ) [virtual]
```

Sets state\_array and state\_time to their user-specified initial values.

## Returns

int

Reimplemented in [EulerLeap](#), [MidpointLeap](#), and [NextReaction](#).

## 7.8.3.4 simulate()

```
int Realization::simulate (
    std::ofstream & myfile )
```

Simulates the realization from t\_initial to t\_final.

## Returns

int

The documentation for this class was generated from the following files:

- [/Users/Caleb/APC524/stoched/src/realization.h](#)
- [/Users/Caleb/APC524/stoched/src/realization.cc](#)

## 7.9 RealizationFactory Class Reference

Class [RealizationFactory](#) generates required instance of [Realization](#) ([FirstReaction](#), [NextReaction](#), [EulerLeap](#), [MidpointLeap](#)) based on input.

```
#include <realization_factory.h>
```

### Static Public Member Functions

- static [Realization](#) \* [NewRealization](#) ([Model](#) \*the\_model, const [Paramset](#) &the\_paramset, [rng](#) \*the\_rng, int n\_vars, int n\_events)  
*Create new realization.*

### 7.9.1 Detailed Description

Class [RealizationFactory](#) generates required instance of [Realization](#) ([FirstReaction](#), [NextReaction](#), [EulerLeap](#), [MidpointLeap](#)) based on input.

## 7.9.2 Member Function Documentation

### 7.9.2.1 NewRealization()

```
Realization * RealizationFactory::NewRealization (
    Model * the_model,
    const Paramset & the_paramset,
    rng * the_rng,
    int n_vars,
    int n_events ) [static]
```

Create new realization.

#### Parameters

<i>the_model</i>	is a <a href="#">Model</a> object
<i>the_paramset</i>	is a <a href="#">Paramset</a> object
<i>the_rng</i>	is a random number generator
<i>n_vars</i>	is an int specifying variable count
<i>n_events</i>	is an int specifying event count

#### Returns

nothing

The documentation for this class was generated from the following files:

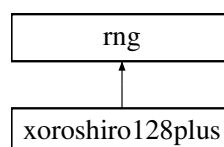
- [/Users/Caleb/APC524/stoched/src/realization\\_factory.h](#)
- [/Users/Caleb/APC524/stoched/src/realization\\_factory.cc](#)

## 7.10 rng Class Reference

Class rng implements random number generator, based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ([vigna@acm.org](mailto:vigna@acm.org))

```
#include <rng.h>
```

Inheritance diagram for rng:



## Public Member Functions

- virtual `~rng()`  
*Destructor of rng object.*
- virtual `uint64_t next()`=0  
*get a new random int64*
- virtual `double runif()`=0  
*get a new random uniform(0, 1) RV*
- `double rexp(double lambda)`  
*get a new random exponential(lambda) RV*
- `long rpois(double mean)`  
*get a new random poisson(mean) RV*
- virtual `void jump()`=0  
*quick 2^64 calls to next (for parallelism)*
- `double log_factorial(int k)`

## Private Member Functions

- `long poisson_knuth(double mean)`  
*random poisson*
- `long poisson_ptrs(double mean)`  
*random poisson*

### 7.10.1 Detailed Description

Class `rng` implements random number generator, based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ([vigna@acm.org](mailto:vigna@acm.org))

### 7.10.2 Member Function Documentation

#### 7.10.2.1 `log_factorial()`

```
double rng::log_factorial (  
    int k )
```

`log factorial` function modified from public domain C# implementation by John D. Cook ([http://www.johndcook.com/blog/csharp\\_log\\_factorial/](http://www.johndcook.com/blog/csharp_log_factorial/)) and PTRS algorithm by Wolfgang Hoermann (1993)

The documentation for this class was generated from the following files:

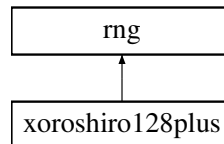
- `/Users/Caleb/APC524/stoched/src/rng.h`
- `/Users/Caleb/APC524/stoched/src/rng.cc`

## 7.11 xoroshiro128plus Class Reference

Class [xoroshiro128plus](#) implements a random number generator of Class [rng](#).

```
#include <xoroshiro128plus.h>
```

Inheritance diagram for xoroshiro128plus:



### Public Member Functions

- [xoroshiro128plus](#) (int seed)  
*Default constructor.*
- [~xoroshiro128plus](#) ()  
*Default Destructor for Xoroshiro128plus.*
- uint64\_t [next](#) ()  
*get random int and update state*
- double [runif](#) ()  
*get random uniform(0, 1) double and update state*
- void [jump](#) ()

### Private Member Functions

- uint64\_t [rotr](#) (const uint64\_t x, int k)  
*simulated rotate*
- uint64\_t [splitmix64](#) ()  
*splitmix64 next function, for initializing generator*

### Private Attributes

- uint64\_t [s](#) [2]
- uint64\_t [splitmixstate](#)

#### 7.11.1 Detailed Description

Class [xoroshiro128plus](#) implements a random number generator of Class [rng](#).

#### 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 xoroshiro128plus()

```
xoroshiro128plus::xoroshiro128plus (
    int seed )
```

Default constructor.

initialize state with splitmix64 random ints from seed int (prevents similar seeds from generating correlated states)

## 7.11.3 Member Function Documentation

### 7.11.3.1 jump()

```
void xoroshiro128plus::jump ( ) [virtual]
```

This is the jump function for the generator. It is equivalent to  $2^{64}$  calls to [next\(\)](#); it can be used to generate  $2^{64}$  non-overlapping subsequences for parallel computations.

Implements [rng](#).

The documentation for this class was generated from the following files:

- [/Users/Caleb/APC524/stoched/src/xoroshiro128plus.h](#)
- [/Users/Caleb/APC524/stoched/src/xoroshiro128plus.cc](#)



## Chapter 8

# File Documentation

### 8.1 /Users/Caleb/APC524/stoched/src/eulerleap.cc File Reference

Class [EulerLeap](#) implements [Realization](#) step() function using the basic tau leap approximate method of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

```
#include "eulerleap.h"
```

#### 8.1.1 Detailed Description

Class [EulerLeap](#) implements [Realization](#) step() function using the basic tau leap approximate method of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

##### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

##### Date

12/6/16

##### Version

1.0

### 8.2 /Users/Caleb/APC524/stoched/src/eulerleap.h File Reference

Class [EulerLeap](#) implements [Realization](#) step() function using the basic tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

```
#include <stdexcept>
#include "realization.h"
```

## Classes

- class [EulerLeap](#)

Class [EulerLeap](#) implements [Realization step\(\)](#) function using the basic tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

### 8.2.1 Detailed Description

Class [EulerLeap](#) implements [Realization](#) step() function using the basic tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.3 /Users/Caleb/APC524/stoched/src/event.cc File Reference

Class [Event](#) holds a user-specified event, namely set of functions and associated rate.

```
#include "event.h"
#include "fparser/fparser.hh"
#include <assert.h>
#include <string>
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
```

### 8.3.1 Detailed Description

Class [Event](#) holds a user-specified event, namely set of functions and associated rate.

#### Author

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

#### Date

12/6/16

#### Version

1.0



## 8.4 /Users/Caleb/APC524/stoched/src/event.h File Reference

Class [Event](#) holds a user-specified event, namely set of functions and associated rate.

```
#include "fparser/fparser.hh"
```

### Classes

- class [Event](#)

*Class [Event](#) holds a user-specified event, namely set of functions and associated rate.*

#### 8.4.1 Detailed Description

Class [Event](#) holds a user-specified event, namely set of functions and associated rate.

#### Author

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.5 /Users/Caleb/APC524/stoched/src/eventtests.cc File Reference

Test [Event](#) code.

```
#include <string>
#include <stdio.h>
#include <iostream>
#include "event.h"
#include "gtest/gtest.h"
```

### Functions

- **TEST\_F** (EventTests, UseFunction)
- **TEST\_F** (EventTests, RateFunction)
- int **main** (int argc, char \*\*argv)

### 8.5.1 Detailed Description

Test [Event](#) code.

#### Author

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

#### Date

1/12/17

#### Version

1.0

## 8.6 /Users/Caleb/APC524/stoched/src/firstreaction.cc File Reference

Class [FirstReaction](#) implements [Realization](#) `step()` function using the exact First Reaction algorithm of Gillespie (1971)

```
#include "firstreaction.h"
```

### 8.6.1 Detailed Description

Class [FirstReaction](#) implements [Realization](#) `step()` function using the exact First Reaction algorithm of Gillespie (1971)

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.7 /Users/Caleb/APC524/stoched/src/firstreaction.h File Reference

Class [FirstReaction](#) implements [Realization](#) `step()` function using the exact First Reaction algorithm of Gillespie (1971)

```
#include "realization.h"
```

## Classes

- class [FirstReaction](#)

Class [FirstReaction](#) implements [Realization step\(\)](#) function using the exact First Reaction algorithm of Gillespie (1971)

### 8.7.1 Detailed Description

Class [FirstReaction](#) implements [Realization](#) step() function using the exact First Reaction algorithm of Gillespie (1971)

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.8 /Users/Caleb/APC524/stoched/src/midpointleap.cc File Reference

Class [MidpointLeap](#) implements [Realization](#) step() function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

```
#include "midpointleap.h"
```

### 8.8.1 Detailed Description

Class [MidpointLeap](#) implements [Realization](#) step() function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

2016-01-16

#### Version

1.0

## 8.9 /Users/Caleb/APC524/stoched/src/midpointleap.h File Reference

Class [MidpointLeap](#) implements [Realization](#) `step()` function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

```
#include <stdexcept>
#include "realization.h"
```

### Classes

- class [MidpointLeap](#)

*Class [MidpointLeap](#) implements [Realization](#) `step()` function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.*

### 8.9.1 Detailed Description

Class [MidpointLeap](#) implements [Realization](#) `step()` function using the midpoint tau leap approximate algorithm of Gillespie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

2016-01-16

#### Version

1.0

## 8.10 /Users/Caleb/APC524/stoched/src/model.cc File Reference

Class [Model](#), which holds user-specified models of stochastic systems from which realizations are to be simulated.

```
#include "event.h"
#include "model.h"
#include "fparser/fparser.hh"
#include <assert.h>
#include <string>
#include <stdio.h>
#include <iostream>
#include <vector>
#include <sstream>
```

### 8.10.1 Detailed Description

Class [Model](#), which holds user-specified models of stochastic systems from which realizations are to be simulated.

**Author**

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

**Date**

12/6/16

**Version**

1.0

## 8.11 /Users/Caleb/APC524/stoched/src/model.h File Reference

Class [Model](#), which holds user-specified models of stochastic systems from which realizations are to be simulated.

```
#include "event.h"  
#include <vector>  
#include <sstream>  
#include <string>
```

### Classes

- class [Model](#)

*Class [Model](#), which holds user-specified models of stochastic systems from which realizations are to be simulated. A model may have variable parameters; each complete set will be stored in an object of class [Paramset](#).*

### 8.11.1 Detailed Description

Class [Model](#), which holds user-specified models of stochastic systems from which realizations are to be simulated.

**Author**

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

**Date**

12/6/16

**Version**

1.0

## 8.12 /Users/Caleb/APC524/stoched/src/modeltests.cc File Reference

Test usage of [Model](#) class.

```
#include <string>
#include <stdio.h>
#include <iostream>
#include "model.h"
#include "gtest/gtest.h"
```

### Functions

- **TEST\_F** (ModelTests, UseEventFunction)
- **TEST\_F** (ModelTests, EventRateFunction)
- **TEST\_F** (ModelTests, UpdateRateFunction)
- **TEST\_F** (ModelTests, UpdateStateArray)
- **int main** (int argc, char \*\*argv)

### 8.12.1 Detailed Description

Test usage of [Model](#) class.

#### Author

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

#### Date

12/11/16

#### Version

1.0

## 8.13 /Users/Caleb/APC524/stoched/src/nextreaction.cc File Reference

Class [NextReaction](#) implements [Realization](#) step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

```
#include "nextreaction.h"
```

### 8.13.1 Detailed Description

Class [NextReaction](#) implements [Realization](#) step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

## 8.14 /Users/Caleb/APC524/stoched/src/nextreaction.h File Reference

Class nextreaction implements [Realization](#) step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

```
#include "realization.h"
#include <float.h>
```

### Classes

- class [NextReaction](#)

*Class [NextReaction](#) implements [Realization step\(\)](#) function using the exact Next Reaction algorithm of Gibson & Bruck (2000)*

### 8.14.1 Detailed Description

Class nextreaction implements [Realization](#) step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.15 /Users/Caleb/APC524/stoched/src/paramset.cc File Reference

Class [Paramset](#) holds a particular set of parameters for user requested simulation run(s)

```
#include "paramset.h"
```

### 8.15.1 Detailed Description

Class [Paramset](#) holds a particular set of parameters for user requested simulation run(s)

#### Author

Dillon Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.16 /Users/Caleb/APC524/stoched/src/paramset.h File Reference

Class [Paramset](#) holds a particular set of parameters for user requested simulation run(s)

### Classes

- class [Paramset](#)

*Class [Paramset](#) holds a particular set of parameters for user requested simulation run(s)*

### 8.16.1 Detailed Description

Class [Paramset](#) holds a particular set of parameters for user requested simulation run(s)

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.17 /Users/Caleb/APC524/stoched/src/parsertests.cc File Reference

Example parsing code.

```
#include <string>
#include <stdio.h>
#include <iostream>
#include "model.h"
#include "event.h"
#include "gtest/gtest.h"
```

### Functions

- int **parseFile** ([Model](#) &model, string inputfilename)
- **TEST\_F** (ParserTests, ParserReturn)
- int **main** (int argc, char \*\*argv)



### 8.17.1 Detailed Description

Example parsing code.

**Author**

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

**Date**

1/12/17

**Version**

1.0

## 8.18 /Users/Caleb/APC524/stoched/src/realization.cc File Reference

Class [Realization](#) holds realizations of a [Model](#) (state array, propensities, waiting times, etc.)

```
#include "realization.h"
```

### 8.18.1 Detailed Description

Class [Realization](#) holds realizations of a [Model](#) (state array, propensities, waiting times, etc.)

**Author**

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

**Date**

12/6/16

**Version**

1.0

## 8.19 /Users/Caleb/APC524/stoched/src/realization.h File Reference

Class [Realization](#) holds realizations of a [Model](#) (state array, propensities, waiting times, etc.)

```
#include <math.h>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <float.h>
#include <stdexcept>
#include <iostream>
#include "model.h"
#include "paramset.h"
#include "rng.h"
```

## Classes

- class [Realization](#)

Class [Realization](#) holds realizations of a [Model](#) (state array, propensities, waiting times, etc.)

### 8.19.1 Detailed Description

Class [Realization](#) holds realizations of a [Model](#) (state array, propensities, waiting times, etc.)

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.20 /Users/Caleb/APC524/stoched/src/realization\_factory.cc File Reference

Class [RealizationFactory](#) generates required instance of [Realization](#) ([FirstReaction](#), [NextReaction](#), [EulerLeap](#)) based on input.

```
#include "realization_factory.h"
```

### 8.20.1 Detailed Description

Class [RealizationFactory](#) generates required instance of [Realization](#) ([FirstReaction](#), [NextReaction](#), [EulerLeap](#)) based on input.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.21 /Users/Caleb/APC524/stoched/src/realization\_factory.h File Reference

Class [RealizationFactory](#) generates required instance of [Realization](#) ([FirstReaction](#), [NextReaction](#), [EulerLeap](#)) based on input.

```
#include "realization.h"
#include "nextreaction.h"
#include "firstreaction.h"
#include "eulerleap.h"
#include "midpointleap.h"
```

### Classes

- class [RealizationFactory](#)

*Class [RealizationFactory](#) generates required instance of [Realization](#) ([FirstReaction](#), [NextReaction](#), [EulerLeap](#), [MidpointLeap](#)) based on input.*

### 8.21.1 Detailed Description

Class [RealizationFactory](#) generates required instance of [Realization](#) ([FirstReaction](#), [NextReaction](#), [EulerLeap](#)) based on input.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.22 /Users/Caleb/APC524/stoched/src/rng.cc File Reference

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ([vigna@acm.org](mailto:vigna@acm.org))

```
#include "rng.h"
#include <math.h>
```

### 8.22.1 Detailed Description

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ([vigna@acm.org](mailto:vigna@acm.org))

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

### 8.22.2 `poisson_ptrs(double mean)` implements the PTRS

algorithm of Wolfgang Hoermann (1993) and is based upon the function `rk_poisson_ptrs` from the `mtrand` module of `numpy`, available here: [github.com/numpy/numpy/blob/master/numpy/random/mtrand/distributions.c](https://github.com/numpy/numpy/blob/master/numpy/random/mtrand/distributions.c)

That function appears with the following copyright and license:

Copyright 2005 Robert Kern ([robert.kern@gmail.com](mailto:robert.kern@gmail.com))

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 8.23 `/Users/Caleb/APC524/stoched/src/rng.h` File Reference

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ([vigna@acm.org](mailto:vigna@acm.org))

```
#include <stdint.h>
#include <math.h>
#include <stdexcept>
```

## Classes

- class [rng](#)

*Class [rng](#) implements random number generator, based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ([vigna@acm.org](mailto:vigna@acm.org))*

## Functions

- double [to\\_double](#) (uint64\_t x)  
*convert uint64\_t to double in (0, 1)*

### 8.23.1 Detailed Description

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna ([vigna@acm.org](mailto:vigna@acm.org))

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.24 /Users/Caleb/APC524/stoched/src/simulate.cc File Reference

Example definition for ending simulation loop.

```
#include <stdio.h>
#include "model.h"
#include "paramset.h"
#include "realization.h"
```

## Functions

- int [main](#) (int argc, char \*argv[])  
*Example definition for ending simulation loop.*

### 8.24.1 Detailed Description

Example definition for ending simulation loop.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

### 8.24.2 Function Documentation

#### 8.24.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Example definition for ending simulation loop.

#### Returns

int

## 8.25 /Users/Caleb/APC524/stoched/src/tauleapavail.cc File Reference

Called by parser.y to determine whether to use tau leap or not.

```
#include <cstdio>
#include <iostream>
#include "model.h"
```

### Functions

- bool [tauLeapAvail](#) ([Model](#) &cModel, string varListStr, string functionStr, int eqnCnt, int eventCnt)  
*Is tau leap available?*

### 8.25.1 Detailed Description

Called by parser.y to determine whether to use tau leap or not.

#### Author

Kevin Griffin ([kpgriffin@princeton.edu](mailto:kpgriffin@princeton.edu))

### 8.25.2 Function Documentation

#### 8.25.2.1 tauLeapAvail()

```
bool tauLeapAvail (
    Model & cModel,
    string varListStr,
    string functionStr,
    int eqnCnt,
    int eventCnt )
```

Is tau leap available?

#### Parameters

<i>cmodel</i>	is a <a href="#">Model</a> object
<i>varList</i>	str is a string of variables
<i>functionStr</i>	is a string used to generate a Function Parser
<i>eqnCnt</i>	is an int specifying number of equations
<i>eventCnt</i>	is an int specifying number of events

#### Returns

nothing

## 8.26 /Users/Caleb/APC524/stoched/src/testevent.cc File Reference

Example usage of [Event](#) class.

```
#include <iostream>
#include <string>
#include <stdio.h>
#include "event.h"
```

### Functions

- int [main](#) ()

Example usage of [Event](#) class.

### 8.26.1 Detailed Description

Example usage of [Event](#) class.

#### Author

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

#### Date

12/6/16

#### Version

1.0

### 8.26.2 Function Documentation

#### 8.26.2.1 main()

```
int main ( )
```

Example usage of [Event](#) class.

#### Returns

int

## 8.27 /Users/Caleb/APC524/stoched/src/testmodel.cc File Reference

Example usage of [Model](#) class.

```
#include <iostream>
#include <string>
#include <stdio.h>
#include "event.h"
#include "model.h"
```

### Functions

- int [main](#) ()

*Example usage of [Model](#) class.*



### 8.27.1 Detailed Description

Example usage of [Model](#) class.

#### Author

Caleb Peckham ([peckham@princeton.edu](mailto:peckham@princeton.edu))

#### Date

12/6/16

#### Version

1.0

### 8.27.2 Function Documentation

#### 8.27.2.1 `main()`

```
int main ( )
```

Example usage of [Model](#) class.

#### Returns

int

## 8.28 /Users/Caleb/APC524/stoched/src/testparser.cc File Reference

Example parsing code.

```
#include <stdio.h>
#include <iostream>
#include "model.h"
#include "event.h"
```

### Functions

- int [parseFile](#) ([Model](#) &model, string inputfilename)  
*Declare the parser method written by flex and bison.*
- int [main](#) (int argc, char \*argv[])  
*Example parsing code.*

### 8.28.1 Detailed Description

Example parsing code.

#### Author

Kevin Griffin ([kevinpg@princeton.edu](mailto:kevinpg@princeton.edu))

#### Date

12/6/16

#### Version

1.0

### 8.28.2 Function Documentation

#### 8.28.2.1 main()

```
int main (
    int argc,
    char * argv[ ] )
```

Example parsing code.

#### Returns

int

## 8.29 /Users/Caleb/APC524/stoched/src/testsimulate.cc File Reference

Example simulation code.

```
#include <iostream>
#include <string>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <chrono>
#include "event.h"
#include "model.h"
#include "paramset.h"
#include "realization.h"
#include "xoroshiro128plus.h"
#include "nextreaction.h"
#include "firstreaction.h"
```

## Functions

- int **main** ()

### 8.29.1 Detailed Description

Example simulation code.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.30 /Users/Caleb/APC524/stoched/src/xoroshiro128plus.cc File Reference

Class xoroshorio128plus implements a random number generator of Class rng.

```
#include "xoroshiro128plus.h"
#include "rng.h"
#include "math.h"
#include "string.h"
```

## Macros

- #define **INT64\_C**(c) (int64\_t) c
- #define **UINT64\_C**(c) (uint64\_t) c

## Functions

- double [to\\_double](#) (uint64\_t x)  
*convert uint64\_t to double in (0, 1)*

### 8.30.1 Detailed Description

Class xoroshorio128plus implements a random number generator of Class rng.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

## 8.31 /Users/Caleb/APC524/stoched/src/xoroshiro128plus.h File Reference

Class xoroshorio128plus implements a random number generator of Class rng.

```
#include <stdint.h>
#include "rng.h"
```

### Classes

- class [xoroshiro128plus](#)

*Class [xoroshiro128plus](#) implements a random number generator of Class rng.*

### 8.31.1 Detailed Description

Class xoroshorio128plus implements a random number generator of Class rng.

#### Author

Dylan Morris ([dhmorris@princeton.edu](mailto:dhmorris@princeton.edu))

#### Date

12/6/16

#### Version

1.0

# Index

/Users/Caleb/APC524/stoched/src/eulerleap.cc, [45](#)  
/Users/Caleb/APC524/stoched/src/eulerleap.h, [45](#)  
/Users/Caleb/APC524/stoched/src/event.cc, [46](#)  
/Users/Caleb/APC524/stoched/src/event.h, [47](#)  
/Users/Caleb/APC524/stoched/src/eventtests.cc, [47](#)  
/Users/Caleb/APC524/stoched/src/firstreaction.cc, [48](#)  
/Users/Caleb/APC524/stoched/src/firstreaction.h, [48](#)  
/Users/Caleb/APC524/stoched/src/midpointleap.cc, [49](#)  
/Users/Caleb/APC524/stoched/src/midpointleap.h, [50](#)  
/Users/Caleb/APC524/stoched/src/model.cc, [50](#)  
/Users/Caleb/APC524/stoched/src/model.h, [51](#)  
/Users/Caleb/APC524/stoched/src/modeltests.cc, [52](#)  
/Users/Caleb/APC524/stoched/src/nextreaction.cc, [52](#)  
/Users/Caleb/APC524/stoched/src/nextreaction.h, [53](#)  
/Users/Caleb/APC524/stoched/src/paramset.cc, [53](#)  
/Users/Caleb/APC524/stoched/src/paramset.h, [54](#)  
/Users/Caleb/APC524/stoched/src/parsertests.cc, [54](#)  
/Users/Caleb/APC524/stoched/src/realization.cc, [55](#)  
/Users/Caleb/APC524/stoched/src/realization.h, [55](#)  
/Users/Caleb/APC524/stoched/src/realization\_↔  
factory.cc, [56](#)  
/Users/Caleb/APC524/stoched/src/realization\_↔  
factory.h, [57](#)  
/Users/Caleb/APC524/stoched/src/rng.cc, [57](#)  
/Users/Caleb/APC524/stoched/src/rng.h, [58](#)  
/Users/Caleb/APC524/stoched/src/simulate.cc, [59](#)  
/Users/Caleb/APC524/stoched/src/tauleapavail.cc, [60](#)  
/Users/Caleb/APC524/stoched/src/testevent.cc, [61](#)  
/Users/Caleb/APC524/stoched/src/testmodel.cc, [62](#)  
/Users/Caleb/APC524/stoched/src/testparser.cc, [63](#)  
/Users/Caleb/APC524/stoched/src/testsimulate.cc, [64](#)  
/Users/Caleb/APC524/stoched/src/xoroshiro128plus.cc,  
[65](#)  
/Users/Caleb/APC524/stoched/src/xoroshiro128plus.h,  
[66](#)  
~EulerLeap  
EulerLeap, [16](#)  
~Event  
Event, [18](#)  
~FirstReaction  
FirstReaction, [23](#)  
~MidpointLeap  
MidpointLeap, [25](#)  
~Model  
Model, [27](#)  
~NextReaction  
NextReaction, [34](#)  
~Paramset  
Paramset, [36](#)  
~Realization  
Realization, [38](#)  
addEvent  
Model, [27](#)  
addEventFct  
Model, [28](#)  
addFunction  
Event, [19](#)  
addVars  
Model, [28](#)  
checkTauLeapAvail  
Model, [28](#)  
EulerLeap, [15](#)  
~EulerLeap, [16](#)  
EulerLeap, [16](#)  
set\_to\_initial\_state, [16](#)  
step, [17](#)  
Event, [17](#)  
~Event, [18](#)  
addFunction, [19](#)  
Event, [18](#)  
getDeltaVar, [19](#)  
getRate, [19](#)  
getSize, [20](#)  
setDeltaVar, [20](#)  
setRate, [20](#)  
useFunction, [21](#)  
FirstReaction, [21](#)  
~FirstReaction, [23](#)  
FirstReaction, [22](#)  
step, [23](#)  
getContDeriv  
Model, [28](#)  
getDeltaVar  
Event, [19](#)  
getEventRate  
Model, [29](#)  
getEventsCount  
Model, [29](#)  
getIthVar  
Model, [29](#)  
getRate  
Event, [19](#)  
getSize  
Event, [20](#)  
getVarsCount

- Model, 30
- getVarsString
  - Model, 30
- jump
  - xoroshiro128plus, 43
- log\_factorial
  - rng, 41
- main
  - simulate.cc, 60
  - testevent.cc, 62
  - testmodel.cc, 63
  - testparser.cc, 64
- MidpointLeap, 23
  - ~MidpointLeap, 25
  - MidpointLeap, 24
  - set\_to\_initial\_state, 25
  - step, 25
- Model, 26
  - ~Model, 27
  - addEvent, 27
  - addEventFct, 28
  - addVars, 28
  - checkTauLeapAvail, 28
  - getContDeriv, 28
  - getEventRate, 29
  - getEventsCount, 29
  - getIthVar, 29
  - getVarsCount, 30
  - getVarsString, 30
  - Model, 27
  - setDelta, 30
  - setTauLeapFalse, 31
  - updateRates, 31
  - updateState, 31
  - useEventFct, 32
- NewRealization
  - RealizationFactory, 40
- NextReaction, 32
  - ~NextReaction, 34
  - NextReaction, 33
  - set\_to\_initial\_state, 34
  - step, 34
- output\_state
  - Realization, 38
- Paramset, 35
  - ~Paramset, 36
  - Paramset, 35
- rates\_are\_zero
  - Realization, 38
- Realization, 36
  - ~Realization, 38
  - output\_state, 38
  - rates\_are\_zero, 38
  - Realization, 37
  - set\_to\_initial\_state, 38
  - simulate, 39
- RealizationFactory, 39
  - NewRealization, 40
- rng, 40
  - log\_factorial, 41
- set\_to\_initial\_state
  - EulerLeap, 16
  - MidpointLeap, 25
  - NextReaction, 34
  - Realization, 38
- setDelta
  - Model, 30
- setDeltaVar
  - Event, 20
- setRate
  - Event, 20
- setTauLeapFalse
  - Model, 31
- simulate
  - Realization, 39
- simulate.cc
  - main, 60
- step
  - EulerLeap, 17
  - FirstReaction, 23
  - MidpointLeap, 25
  - NextReaction, 34
- tauLeapAvail
  - tauleapavail.cc, 61
- tauleapavail.cc
  - tauLeapAvail, 61
- testevent.cc
  - main, 62
- testmodel.cc
  - main, 63
- testparser.cc
  - main, 64
- updateRates
  - Model, 31
- updateState
  - Model, 31
- useEventFct
  - Model, 32
- useFunction
  - Event, 21
- xoroshiro128plus, 42
  - jump, 43
  - xoroshiro128plus, 42