Exact Stochastic Simulation Software

Generated by Doxygen 1.8.7

Tue Sep 30 2014 18:06:06

Contents

Chapter 1

Documentation

Chapter 2

Namespace Documentation

2.1 nw Namespace Reference

Classes

- class _Event

  Abstract base class for system state transitions (events).
- class _Input

  Abstract base class for input procedures.
- class _Random

  abstract base class to define an interface for Random Generators
- class _Species

  Abstract base class for the molecular species of a system.
- class _System

  Abstract base class for the implementation of simulation algorithms.
- class _Voxel

  Abstract base class for all different types of voxel.
- class Border_Vxl

  The border Voxel defines the border condition of a whole voxel-system.
- class Channel_Spc

  Channel Species. This class implements a channel species. This class realizes channel proteins build of a certain number of subunits which open if a defined number of those subunits is in an open state.
- class ChFlux_Rct_Evt

  Channel flux reaction event. This event is derived by the reaction event and introduces the flux of a molecular species through a channel. It differs from an ordinary reaction event due to the fact that the educt (the channel) is not modified by the event itself. Thus the educt vector has to be generated manually in the init() function.
- class Diffusion_Evt

  The diffusion event realizes the diffusion between two voxel.
- class Gillespie_Sys

  The Gillespie_Sys coordinates Gillespie's SSA and the whole output procedure. The whole logic of the Simulation Software is combined at this point. All Events and all Voxel are represented and important functions like build_dependency_graph are located here.
  Important fact is, that Gillespie_Sys just uses the Interfaces _Voxel and _Event what makes it really easy to extend the Simulation.
- class Reaction_Evt

  The Reaction_Event realizes the Reaction of one ore two molecules.
- class Standard_Ipt

  Input class that parses a .xml files and generates a Gillespie_Sys.
- class Standard_Spc

A Standard_Spc is an object that holds the properties of a special kind of molecules.

- class Standard_Vxl

  Standard voxel implementation.
- class SubUnitSwitch_Rct_Evt
- class Uni_Rnd

  "Minimal" random number generator of Park and Miller

Typedefs

- typedef vector< _Voxel * > VoxelVector

  typedefs of Vector related structures
- typedef VoxelVector::iterator VoxelIterator
- typedef vector< _Event * > EventVector

  typedefs of Event related structures
- typedef EventVector::iterator EventIterator
- typedef vector< _Species * > SpeciesVector

  typedefs of Species related structures
- typedef SpeciesVector::iterator SpeciesIterator

Variables

- static const double N_AVO = 6.022e23
- static const string NEGATIVETAUMSG = "NEGATIVE TAU ERROR LAST EVENT "

2.1.1 Typedef Documentation

2.1.1.1 typedef EventVector::iterator nw::EventIterator

2.1.1.2 typedef vector<_Event*> nw::EventVector

typedefs of Event related structures

2.1.1.3 typedef SpeciesVector::iterator nw::SpeciesIterator

2.1.1.4 typedef vector<_Species*> nw::SpeciesVector

typedefs of Species related structures

2.1.1.5 typedef VoxelVector::iterator nw::VoxelIterator

2.1.1.6 typedef vector< _Voxel * > nw::VoxelVector

typedefs of Vector related structures

2.1.2 Variable Documentation

2.1.2.1 const double nw::N_AVO = 6.022e23 [static]

Definition of Avogadro's constant

2.1.2.2 const string nw::NEGATIVETAUMSG = "NEGATIVE TAU ERROR LAST EVENT " [static]

Chapter 3

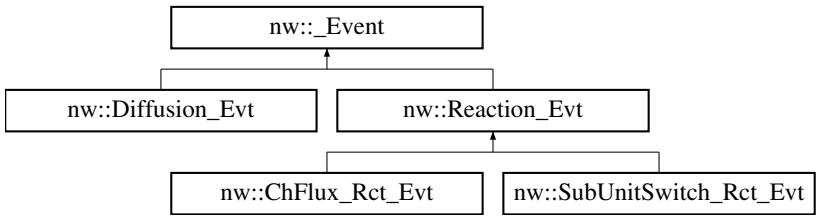
Class Documentation

3.1 nw::_Event Class Reference

Abstract base class for system state transitions (events).

`#include <_Event.h>`

Inheritance diagram for nw::_Event:



Classes

- struct tv_struct

  tau-voxel-structure (tv_struct) is a structure that associates a tau value with a _Voxel.

Public Member Functions

- _Event (long id, string name, double k, VoxelVector vvc, Uni_Rnd *rg)

  Constructor.
- virtual ~_Event ()

  Destructor.
- virtual double update (double last_tau)=0

  Updates the event.
- virtual void execute ()=0

  Event execution function.
- virtual void init ()=0

  Event initialization.
- virtual double get_a ()=0

  returns current _Event propensity a.
- void add_dep_list (_Event *e)

  Adds an _Event pointer to the dependency list.

- void set_flag (bool b)

  set the dirty flag.
- string get_name ()
- long get_id ()
- double get_tau ()
- vector< long > get_sc_vec ()
- vector< _Event * > get_dep_list ()

Protected Attributes

- long id

  ID.
- string name

  Name.
- double k

  Rate constant.
- vector< long > sc_vec

  State change vector. Defines an event in terms of a vector, representing the molecular change for each existing _Species.
- vector< tv_struct > tv_vec

  Tau voxel vector. Container for tv_structs, each representing a _Voxel where this _Event can occur.
- Uni_Rnd * rg

  Uniform random number generator.
- long nextVoxel

  Index of next _Voxel to be executed.
- double c

  Adapted _Event rate constant. Based on the type of _Event `c` *is depends on _Voxel volume (bimoleuclar reactions) or _Voxel box length (diffusion)*
- bool dirty_flag

  indicates wheather or not the tau value has to be recalculated (`TRUE`*) or adapted to the absolute time scale (*`FALSE`*)*
- vector< _Event * > dep_list

  Dependency vector that stores references to _Events that depend on this _Event.

3.1.1 Detailed Description

Abstract base class for system state transitions (events).

A system state transition is called event and is defined by a propensity function, a state change vector and a dependency list. Propensity functions are derived from rate constants that result either from experimental observations and/or fundamental physical laws. A state change vector represent the stoichiometry of an event and represents the numerical changes in molecular populations caused by the respective event. The dependency list is an implementation of a dependency graph that defines how events influence each other. This data structure is crucial to implement Gibbson and Bruck's optimized update procedure.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 nw::_Event::_Event (long *id*, string *name*, double *k*, VoxelVector *vvc*, Uni_Rnd * *rg*) [inline]

Constructor.

Parameters

id	Event ID
name	Event name
k	Rate constant
vvc	Voxel vector containing all _Voxels where this _Event can occur.
rg	Random generator that produces uniform distributed random numbers

```
35                                                                          :
36          id(id),
37          name(name),
38          k(k),
39          rg(rg),
40          dirty_flag(true){
41
42  //      add self-reference to dependency list.
43          this->dep_list.push_back(this);
44  //      check if all voxel in the assigned voxel vector have the same size
45          for (size_t i = 0; i < tv_vec.size(); i++){
46              if(vvc[i]->get_volume() != vvc[0]->get_volume()){
47                  cout << "ERROR: Voxel sizes in Event: " << this->name << " differ in size. "
48                          "You either have to create another Event or resize the voxel to one size"
49                          << std::endl;
50                  break;
51              }
52          }
53
54  //      create the tau voxel structure container that assigns a tau value to each _Voxel
55  //      in the _Voxel vector
56          tv_vec.resize(vvc.size());
57          for(size_t i = 0; i < tv_vec.size(); i++){
58              tv_vec[i].v = vvc[i];
59              tv_vec[i].t = 0.0;
60          }
61      }
```

3.1.2.2 virtual nw::_Event::~_Event ( ) [inline], [virtual]

Destructor.

```
63 {};
```

3.1.3 Member Function Documentation

3.1.3.1 void nw::_Event::add_dep_list (_Event * *e*) [inline]

Adds an _Event pointer to the dependency list.

Parameters

e	pointer to event that depends on this event

Is called during the generation of a dependency graph that connects dependent events with each other to speed up the update procedure. An _Event A depends on another _Event B, if at least one of the educts of A is also a product of B.

```
95 {dep_list.push_back(e);}
```

3.1.3.2 virtual void nw::_Event::execute ( ) [pure virtual]

Event execution function.

Executes (fires) this _Event. It looks in the tv_struct for the voxel with the smallest associated tau value and calls its update_state() function.

Implemented in nw::Diffusion_Evt, nw::Reaction_Evt, and nw::SubUnitSwitch_Rct_Evt.

3.1.3.3 virtual double nw::_Event::get_a ( ) [pure virtual]

returns current _Event propensity a.

Returns

_Event propensity a.

This function returns the a_value of the event at the current system state. This information is mainly used for analytical purposes

Implemented in nw::Diffusion_Evt, and nw::Reaction_Evt.

3.1.3.4 vector<_Event*> nw::_Event::get_dep_list ( ) [inline]

```
108 {return dep_list;}
```

3.1.3.5 long nw::_Event::get_id ( ) [inline]

```
105 {return id;}
```

3.1.3.6 string nw::_Event::get_name ( ) [inline]

```
104 {return name;}
```

3.1.3.7 vector<long> nw::_Event::get_sc_vec ( ) [inline]

```
107 {return sc_vec;}
```

3.1.3.8 double nw::_Event::get_tau ( ) [inline]

```
106 {return tv_vec[nextVoxel].t;}
```

3.1.3.9 virtual void nw::_Event::init ( ) [pure virtual]

Event initialization.

This function builds the educt vector of an event and transforms its probability rate according to its reaction order. reaction order = educt_vetor.size()!

Implemented in nw::Diffusion_Evt, nw::Reaction_Evt, and nw::ChFlux_Rct_Evt.

3.1.3.10 void nw::_Event::set_flag (bool *b*) [inline]

set the dirty flag.

Parameters

b	boolean that is assigned to dirty_flag.

As a _Voxel, every _Event has a dirty flag to indicate weather it has to be updated or not.

```
101 {dirty_flag=b;}
```

3.1.3.11 virtual double nw::_Event::update (double *last_tau*) [pure virtual]

Updates the event.

Returns

New tau value for this _Event

Parameters

last_tau	Tau value of previous firing _Event

Implementation of the update procedure. The tau value represents the time that has to pass until an event fires again. It has to be updated during every simulation step. An _Event with dirty_flag = TRUE has to generate a new random number to recalculate a tau value. An _Event with dirty_flag = FALSE is adapted to the global time scale by subtracting the last tau from their (necessarily larger) current tau. No negative tau values should be returned!

Implemented in nw::Diffusion_Evt, nw::Reaction_Evt, and nw::ChFlux_Rct_Evt.

3.1.4 Member Data Documentation

3.1.4.1 double nw::_Event::c [protected]

Adapted _Event rate constant. Based on the type of _Event c is depends on _Voxel volume (bimoleuclar reactions) or _Voxel box length (diffusion)

3.1.4.2 vector<_Event*> nw::_Event::dep_list [protected]

Dependency vector that stores references to _Events that depend on this _Event.

3.1.4.3 bool nw::_Event::dirty_flag [protected]

indicates wheather or not the tau value has to be recalculated (TRUE) or adapted to the absolute time scale (FALSE)

3.1.4.4 long nw::_Event::id [protected]

ID.

3.1.4.5 double nw::_Event::k [protected]

Rate constant.

3.1.4.6 string nw::_Event::name [protected]

Name.

3.1.4.7 long nw::_Event::nextVoxel [protected]

Index of next _Voxel to be executed.

3.1.4.8 Uni_Rnd* nw::_Event::rg [protected]

Uniform random number generator.

3.1.4.9 vector<long> nw::_Event::sc_vec [protected]

State change vector. Defines an event in terms of a vector, representing the molecular change for each existing _Species.

3.1.4.10 vector<tv_struct> nw::_Event::tv_vec [protected]

Tau voxel vector. Container for tv_structs, each representing a _Voxel where this _Event can occur.

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

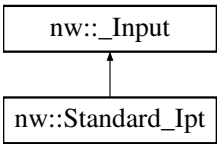
- Events/_Event.h

3.2 nw::_Input Class Reference

Abstract base class for input procedures.

`#include <_Input.h>`

Inheritance diagram for nw::_Input:



Public Member Functions

- virtual ~_Input ()
- virtual nw::_System * get_System ()=0

  Translates input files into a _System.

Protected Attributes

- nw::_System * s

3.2.1 Detailed Description

Abstract base class for input procedures.

Derivatives of `_Input` can implement different input interfaces to ensure compatibility with different input formats

3.2.2 Constructor & Destructor Documentation

3.2.2.1 virtual nw::_Input::~_Input ( ) [inline], [virtual]

```
13 {};
```

3.2.3 Member Function Documentation

3.2.3.1 virtual nw::_System* nw::_Input::get_System ( ) [pure virtual]

Translates input files into a _System.

Returns

_System based on an input file.

Implemented in nw::Standard_Ipt.

3.2.4 Member Data Documentation

3.2.4.1 nw::_System* nw::_Input::s [protected]

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

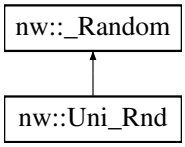
- Input/_Input.h

3.3 nw::_Random Class Reference

abstract base class to define an interface for Random Generators

`#include <_Random.h>`

Inheritance diagram for nw::_Random:



Public Member Functions

- virtual double get_Uni_Rnd ()=0

  Generate uniformly distributed random number between 0.0 and 1.0.
- virtual ~_Random ()

  Destructor.

Private Attributes

- double r

3.3.1 Detailed Description

abstract base class to define an interface for Random Generators

3.3.2 Constructor & Destructor Documentation

3.3.2.1 virtual nw::_Random::~_Random ( ) [inline], [virtual]

Destructor.

```
13 {};
```

3.3.3 Member Function Documentation

3.3.3.1 virtual double nw::_Random::get_Uni_Rnd ( ) [pure virtual]

Generate uniformly distributed random number between 0.0 and 1.0.

Returns

Uniformly distributed random number between 0.0 and 1.0

Implemented in nw::Uni_Rnd.

3.3.4 Member Data Documentation

3.3.4.1 double nw::_Random::r [private]

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

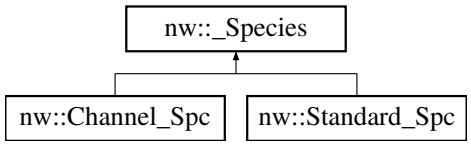
- Random/_Random.h

3.4 nw::_Species Class Reference

Abstract base class for the molecular species of a system.

`#include <_Species.h>`

Inheritance diagram for nw::_Species:



Public Member Functions

- _Species (long id, string name, double init_conc)
  Constructor.
- virtual ~_Species ()
  Destructor.
- virtual long get_n_molecules ()=0
  Get current number of molecules.
- virtual void mod_n_molecules (long n)=0
  Modify number of molecules (n_molecules).
- virtual _Species * copy ()=0

Copy method.

- virtual void set_n_molecules (double volume)

  Set initial molecular count (n_molecules).
- long get_id ()
- string get_name ()
- double get_initial_conc ()
- bool get_dirty_flag ()
- void set_dirty_flag (bool b)

Protected Attributes

- long id

  Species ID.
- string name

  Species name.
- bool dirty_flag

  Indicates whether this _Species needs to be updated or not.
- long n_molecules

  Current number of molecules.
- double initial_conc

  Initial concentration.
- double volume

  Volume of the _Voxel containing this _Species.

3.4.1 Detailed Description

Abstract base class for the molecular species of a system.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 nw::_Species::_Species (long *id*, string *name*, double *init_conc*) [inline]

Constructor.

Parameters

id	Species ID
name	Species name
init_conc	initial concentration

```
24                                                  :
25         id(id),
26         name(name),
27         dirty_flag(true),
28         initial_conc(init_conc) {}
```

3.4.2.2 virtual nw::_Species::~_Species ( ) [inline], [virtual]

Destructor.

```
30 {};
```

3.4.3 Member Function Documentation

3.4.3.1 virtual _Species* nw::_Species::copy ( ) [pure virtual]

Copy method.

Returns

Reference to an object identical to this

Copy method that generates a copy of this. It is is a virtual function to ensure that whenever this function is called, the most specific object (derived from this abstract base class) is copied.

Implemented in nw::Channel_Spc, and nw::Standard_Spc.

3.4.3.2 bool nw::_Species::get_dirty_flag ( ) [inline]

```
60 {return dirty_flag;}
```

3.4.3.3 long nw::_Species::get_id ( ) [inline]

```
57 {return id;}
```

3.4.3.4 double nw::_Species::get_initial_conc ( ) [inline]

```
59 {return initial_conc;}
```

3.4.3.5 virtual long nw::_Species::get_n_molecules ( ) [pure virtual]

Get current number of molecules.

Returns

current number of molecules

Implemented in nw::Channel_Spc, and nw::Standard_Spc.

3.4.3.6 string nw::_Species::get_name ( ) [inline]

```
58 {return name;}
```

3.4.3.7 virtual void nw::_Species::mod_n_molecules (long *n*) [pure virtual]

Modify number of molecules (n_molecules).

Parameters

n	Summand that is added to the current number of molecules.

If an event occurs, this function is called to update the number of molecules based on the state change vector of the firing _Event. If `n` is positive the number of molecules is increased, if it is negative the number of molecules in decrease.

Implemented in nw::Channel_Spc, and nw::Standard_Spc.

3.4.3.8 void nw::_Species::set_dirty_flag (bool *b*) [inline]

```
61 {dirty_flag = b;}
```

3.4.3.9 virtual void nw::_Species::set_n_molecules (double *volume*) [inline], [virtual]

Set initial molecular count (n_molecules).

Called in the constructor of every _Voxel to set the correct (_Voxel volume adapted) number of molecules. This makes it easier to define _Species and _Voxel in the input file.

Reimplemented in nw::Channel_Spc.

```
53                                          {
54         this->n_molecules = volume*1e-6*N_AVO*initial_conc;}
```

3.4.4 Member Data Documentation

3.4.4.1 bool nw::_Species::dirty_flag [protected]

Indicates whether this _Species needs to be updated or not.

3.4.4.2 long nw::_Species::id [protected]

Species ID.

3.4.4.3 double nw::_Species::initial_conc [protected]

Initial concentration.

3.4.4.4 long nw::_Species::n_molecules [protected]

Current number of molecules.

3.4.4.5 string nw::_Species::name [protected]

Species name.

3.4.4.6 double nw::_Species::volume [protected]

Volume of the _Voxel containing this _Species.

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

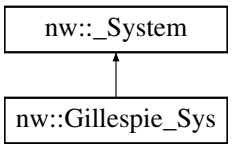
- Species/_Species.h

3.5 nw::_System Class Reference

Abstract base class for the implementation of simulation algorithms.

`#include <_System.h>`

Inheritance diagram for nw::_System:



Public Member Functions

- _System ()
- virtual ~_System ()
- virtual void go (long n_it)=0

  executes simulation algorithm

3.5.1 Detailed Description

Abstract base class for the implementation of simulation algorithms.

Algorithms integrated in this software framework are derived from _System. It solely requires the existence of the function go() that executes the algorithm.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 nw::_System::_System ( ) [inline]

```
13 {};
```

3.5.2.2 virtual nw::_System::~_System ( ) [inline], [virtual]

```
14 {};
```

3.5.3 Member Function Documentation

3.5.3.1 virtual void nw::_System::go (long *n_it*) [pure virtual]

executes simulation algorithm

Parameters

n_it	Current number of simulation run (required for output file names to distinguish between multiple trajectories resulting from multiple simulation runs during one program call.)

Implemented in nw::Gillespie_Sys.

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

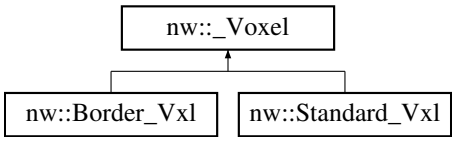
- System/_System.h

3.6 nw::_Voxel Class Reference

Abstract base class for all different types of voxel.

`#include <_Voxel.h>`

Inheritance diagram for nw::_Voxel:



Public Member Functions

- _Voxel (long id, double box_lenght, SpeciesVector state_vec)

  Constructor.
- virtual ~_Voxel ()

  Destructor.
- virtual void update_state (vector< long > sc_vec)=0

  Update the state vector of voxel.
- void clean ()

  Clean dirty flag (set dirty_flag = false).
- void add_neighbour (_Voxel *v)

  Add voxel to diffusion vector.
- bool get_dirty_flag ()
- double get_volume ()
- double get_box_length ()
- long get_id ()
- SpeciesVector * get_state_vec ()
- vector< _Voxel * > * get_diff_vec ()

Protected Attributes

- long id

  _Voxel ID
- double box_length

  _Voxel box length [micro m]
- double volume

  _Voxel volume [l]
- bool dirty_flag

  Indicates if the state vector of this _Voxel has been changed.
- SpeciesVector state_vec

  Reference list for all existing Standard_Spc in this _Voxel, defining the state of this _Voxel.
- vector< _Voxel * > diff_vec

  Reference list of all diffusion neighbors of this _Voxel.

3.6.1 Detailed Description

Abstract base class for all different types of voxel.

A _Voxel is characterized by its spatial extend (volume), its state vector and its vicinity relations. To access the state vector, the function update_state() updates the state according to a state_change_vector (sc_vec). It is absolutely necessary that the sc_vec has the same format as the state vector of the _Voxel. Since only _Voxels affected by the previous _Event need to be updated, a dirty_flag system in analogy to the dirty_flag system in _Event exists. To connect different _Voxel for Diffusion _Events, each _Voxel contains a reference list to all adjacent _Voxels (diff_vec).

3.6.2 Constructor & Destructor Documentation

3.6.2.1 nw::_Voxel::_Voxel (long *id,* double *box_lenght,* SpeciesVector *state_vec*) [inline]

Constructor.

Parameters

id	Voxel ID
box_lenght	Assuming a cubic _Voxel geometry, the box_lenght is used to define the size of a _Voxel.
state_vec	Species vector representing the state of this _Voxel.

```
31                                                                  :
32         id(id),
33         box_length(box_lenght),
34         dirty_flag(true),
35         state_vec(state_vec){
36         this->volume = pow(this->box_length,3)*1e3; // convert m^3 to liter (factor 1e3)
37         for (long i = 0; i < (long)state_vec.size();i++) { // set the number of molecules of all
       species
38             this->state_vec[i]->set_n_molecules(this->volume); /*std::cout <<
       state_vec[i]->get_name() << ": " << state_vec[i]->get_n_molecules() << std::endl;*/
39         }
40     }
```

3.6.2.2 virtual nw::_Voxel::~_Voxel ( ) [inline], [virtual]

Destructor.

```
42                     {
43         for (size_t i = 0; i < state_vec.size(); ++i){if(state_vec.at(i)){delete
       state_vec.at(i);state_vec.at(i)=NULL;}}
44     };
```

3.6.3 Member Function Documentation

3.6.3.1 void nw::_Voxel::add_neighbour (_Voxel * *v*) [inline]

Add voxel to diffusion vector.

The diffusion vector defines the spatial connection between a set of voxel. It serves as lookup table for diffusion events to decide which neighbour voxel becomes the diffusion partner. The add_neighbour function is called after all Voxel have been created.

```
62 {diff_vec.push_back(v);}
```

3.6.3.2 void nw::_Voxel::clean ( ) [inline]

Clean dirty flag (set dirty_flag = false).

The dirty flag is indicates if this _Voxel needs to be updated. Every time an event occurs inside this _Voxel, the dirty flag is set to TRUE and indicates the tau values of this _Voxel needs to be updated (this concerns only _Events that depend on the previous fired _Event).

```
56 {dirty_flag = false;}
```

3.6.3.3 double nw::_Voxel::get_box_length ( ) [inline]

```
67 {return box_length;}
```

3.6.3.4 vector<_Voxel*>* nw::_Voxel::get_diff_vec ( ) [inline]

```
70 {return  &diff_vec;}
```

3.6.3.5 bool nw::_Voxel::get_dirty_flag ( ) [inline]

```
65 {return dirty_flag;}
```

3.6.3.6 long nw::_Voxel::get_id ( ) [inline]

```
68 {return id;}
```

3.6.3.7 SpeciesVector* nw::_Voxel::get_state_vec ( ) [inline]

```
69 {return &state_vec;}
```

3.6.3.8 double nw::_Voxel::get_volume ( ) [inline]

```
66 {return volume;}
```

3.6.3.9 virtual void nw::_Voxel::update_state (vector< long > *sc_vec*) [pure virtual]

Update the state vector of voxel.

Parameters

sc_vec	State change vector of firing event

If an event changes the molecular number of _Species in a voxel, it uses this function to update the state vector with its state change vector.

Implemented in nw::Standard_Vxl.

3.6.4 Member Data Documentation

3.6.4.1 double nw::_Voxel::box_length [protected]

_Voxel box length [micro m]

3.6.4.2 vector<_Voxel*> nw::_Voxel::diff_vec [protected]

Reference list of all diffusion neighbors of this _Voxel.

3.6.4.3 bool nw::_Voxel::dirty_flag [protected]

Indicates if the state vector of this _Voxel has been changed.

3.6.4.4 long nw::_Voxel::id [protected]

_Voxel ID

3.6.4.5 SpeciesVector nw::_Voxel::state_vec [protected]

Reference list for all existing Standard_Spc in this _Voxel, defining the state of this _Voxel.

3.6.4.6 double nw::_Voxel::volume [protected]

_Voxel volume [l]

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

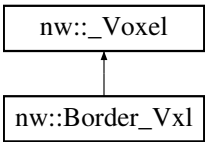
- Voxel/_Voxel.h

3.7 nw::Border_Vxl Class Reference

The border Voxel defines the border condition of a whole voxel-system.

`#include <Border_Vxl.h>`

Inheritance diagram for nw::Border_Vxl:



Public Member Functions

- Border_Vxl (long id, double box_length, SpeciesVector state_vec)
  Constructor of Border Voxel.
- ~Border_Vxl ()
- void update_state (std::vector< long >)

Additional Inherited Members

3.7.1 Detailed Description

The border Voxel defines the border condition of a whole voxel-system.

The easiest version of a border condition is an open system which never changes its state. If the state equals the equation concentration of every species, the whole system will develop to an equilibrium.
That means, that even if there is a diffusion to the Border Voxel, the stat doesn't change at all.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 nw::Border_Vxl::Border_Vxl (long *id*, double *box_length*, SpeciesVector *state_vec*) [inline]

Constructor of Border Voxel.

Parameters

id	Set ID for every Voxel for analysis.
box_length	assuming a square shaped system volume this represents the box lenght.
state_vec	holds all the information about the number of molecules of every species represented in the Voxel.

```
22                                                                    :
23         _Voxel(id, box_length, state_vec) {}
```

3.7.2.2 nw::Border_Vxl::~Border_Vxl ( ) [inline]

```
24 {}
```

3.7.3 Member Function Documentation

3.7.3.1 void nw::Border_Vxl::update_state (std::vector< long >)

```
7 {
8     this->dirty_flag = true;
9
10     for (size_t i = 0; i < sc_vec.size(); ++i) {
11         if (sc_vec[i] != 0) {
12             state_vec[i]->set_dirty_flag(true);
13         }
14     }
15 }
```

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

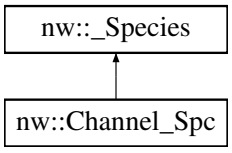
- Voxel/Border_Vxl.h
- Voxel/Border_Vxl.cpp

3.8 nw::Channel_Spc Class Reference

Channel Species. This class implements a channel species. This class realizes channel proteins build of a certain number of subunits which open if a defined number of those subunits is in an open state.

`#include <Channel_Spc.h>`

Inheritance diagram for nw::Channel_Spc:



Public Member Functions

- Channel_Spc (long id, string name, double initial_conc, int n_Subunits, int n_Subunits_To_Open)

  Constructor of Channel Species.
- ~Channel_Spc ()
- long get_n_molecules ()

  return number of open channel. returns only the number of open (active) channel. The total number of channels includes the closed ones that are not interesting in this context.
- void mod_n_molecules (long n)

  don't modify number of molecules. this function is empty, because at the moment there is no way how the number of channels could be modified but through the subunitswitch reaction which calls activate_Subunit() resp. inactivate_Subunit()
- _Species * copy ()

- void set_n_molecules (double volume)

  Calculate number of channels. this function initializes the actual number of participating channel. Therefore, a vector is assigned (ch_Vec) that holds the number of active subunits. If the number of active subunits is equal or bigger than the number of subunits required to open a channel, the number of activated channel is increased.
- long get_n_Activable_Ch ()
- long get_n_Inactivable_Ch ()
- void activate_Subunit (long)

  Activates a subunit if the SubUnitSwitch Reaction Event increases the number of active subunits.
- void inactivate_Subunit (long)

  Inactivates a subunit if the SubUnitSwitch Reaction Event decreases the number of active subunits.

Private Attributes

- int n_Subunits

  number of identical subunits of this channel species.
- int n_Subunits_To_Open

  number of activated subunits that are required to open the channel.
- long n_active_Ch

  number of currently active channels
- long n_fully_activated_Ch

  number of completely activated channels (no subunit activation possible).
- long n_fully_inactivated_Ch

  number of completely inactivated channels (no subunit inactivation possible)
- vector< long > ch_Vec

  one item representing the active subunits for one channel
- vector< long >::iterator ch_it

  vector iterator which iterates through the channel vector to update it

Additional Inherited Members

3.8.1 Detailed Description

Channel Species. This class implements a channel species. This class realizes channel proteins build of a certain number of subunits which open if a defined number of those subunits is in an open state.

This species works a little different from others, because it is only a "imaginary" species. It only subsumes a certain number of other Species (subunits). The molecular count only changes if there are enough active subunits (diffusion not implemented yet). Thus a channel in this model only exists if it is open. To keep track of active subunits, there is an event called subunit switch reaction, which defines the transition between an inactive and an active subunit. If this event occurs, this event typecasts the Species at the channel species id (ch_spc_id) to a channel species and calls activate_subunit() resp. inactivate_subunit(). This is obviously a shitty realization. I plan to introduce an event handling system to deal properly with complex events.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 nw::Channel_Spc::Channel_Spc (long *id*, string *name*, double *initial_conc*, int *n_Subunits*, int *n_Subunits_To_Open*) [inline]

Constructor of Channel Species.

Parameters

id	id of the Species
name	
initial_conc	is the initial concentration of the Species which is converted to a integer value by calling the derived function set_n_molecules().
n_Subunits	sets the number of existing subunits the channel is build of.
n_Subunits_To_Open	this integer sets the number of active subunits are required for the channel to switch in an open state.

```
35                                                                                                  :
36          _Species(id,name,initial_conc),
37          n_Subunits(n_Subunits),
38          n_Subunits_To_Open(n_Subunits_To_Open),
39          n_active_Ch(0),
40          n_fully_activated_Ch(0),
41          n_fully_inactivated_Ch(0){}
```

3.8.2.2 nw::Channel_Spc::~Channel_Spc ( ) [inline]

```
42 {};
```

3.8.3 Member Function Documentation

3.8.3.1 void nw::Channel_Spc::activate_Subunit (long *rnd_id*)

Activates a subunit if the SubUnitSwitch Reaction Event increases the number of active subunits.

Parameters

rnd_id	random number to pick a channel thats subunit is activated. If a subunit is activated, first of all a random number has to be generated (see SubUnitSwitch_Rct_evt) to pick one of the existing channels thats subunit is activated. Note, that one cannot increase the number of active channel subunits if all subunits are in the active state. This makes this function a little bit more complex.

```
12                                                  {
13      ch_it = ch_Vec.begin();
14
15      for (size_t i = 0; i <= size_t(rnd_id);++i){        // iterate to an activatable channel
16          if(i!=0){ch_it++;}
17          while(*ch_it == n_Subunits){
18              ch_it++;
19          }
20      }
21
22      if(*ch_it == 0){n_fully_inactivated_Ch--;
        dirty_flag = true;}           // decrease number of completely inactivated Channel if it was a
         Channel with 0 active subunits
23      if(++*ch_it == n_Subunits_To_Open){n_active_Ch++;
        dirty_flag = true;}   // increase number of active subunits and check if the channel is open!
24      if (*ch_it == n_Subunits) {n_fully_activated_Ch++;
        dirty_flag = true;}   // if the channel is completely activated, increase number of fully
         activated channels
25
26  //  print channel vector + active channels
27  //  for(int i=0;i<(int)ch_Vec.size();i++){cout<<ch_Vec[i]<<"|";}
28  //  cout << "-> "<< n_active_Ch <<endl;
29  }
```

3.8.3.2 _Species* nw::Channel_Spc::copy ( ) [inline], [virtual]

copy method.

Implements nw::_Species.

```
53                      {
54          _Species* s = new Channel_Spc(id,name,
      initial_conc,n_Subunits,n_Subunits_To_Open);
55          return s;}
```

3.8.3.3 long nw::Channel_Spc::get_n_Activable_Ch ( ) [inline]

```
66 {return n_molecules - n_fully_activated_Ch;}
```

3.8.3.4 long nw::Channel_Spc::get_n_Inactivable_Ch ( ) [inline]

```
67 {return n_molecules - n_fully_inactivated_Ch;}
```

3.8.3.5 long nw::Channel_Spc::get_n_molecules ( ) [inline], [virtual]

return number of open channel. returns only the number of open (active) channel. The total number of channels includes the closed ones that are not interesting in this context.

Implements nw::_Species.

```
47 {return n_active_Ch;}
```

3.8.3.6 void nw::Channel_Spc::inactivate_Subunit (long *rnd_id*)

Inactivates a subunit if the SubUnitSwitch Reaction Event decreases the number of active subunits.

Parameters

rnd_id	random number to pick a channel thats subunit is inactivated. If a subunit is inactivated, first of all a random number has to be generated (see SubUnitSwitch_Rct_evt) to pick one of the existing channels thats subunit is inactivated. Note, that one cannot decrease the number of active channel subunits if all subunits are inactivated. This makes this function a little bit more complex.

```
30                                                  {
31      ch_it = ch_Vec.begin();
32      for (size_t i = 0; i <= (size_t)rnd_id;++i){        // iterate to an inactivatable channel
33          if(i!=0){ch_it++;}
34          while(*ch_it == 0){
35              ch_it++;
36          }
37      }
38
39      if(*ch_it == n_Subunits){n_fully_activated_Ch--;
       dirty_flag = true;}   // decrease number of fully activated Channel if it was a Channel with
        maximum active subunits
40      if(*ch_it == n_Subunits_To_Open){n_active_Ch--;
       dirty_flag = true;}       // decrease number of active subunits and check if the channel is open!
41      if (--*ch_it == 0) {n_fully_inactivated_Ch++;
       dirty_flag = true;}       // if the channel is completely inactivated, increase number of fully
        inactivated channels
42
43 //  print channel vector + active channels
44 //  for(int i=0;i<(int)ch_Vec.size();i++){cout<<ch_Vec[i]<<"|";}
45 //  cout << "-> "<< n_active_Ch <<endl;
46 }
```

3.8.3.7 void nw::Channel_Spc::mod_n_molecules (long *n*) [inline], [virtual]

don't modify number of molecules. this function is empty, because at the moment there is no way how the number of channels could be modified but through the subunitswitch reaction which calls activate_Subunit() resp. inactivate_Subunit()

Implements nw::_Species.

```
51 {}
```

3.8.3.8 void nw::Channel_Spc::set_n_molecules (double *volume*) [inline], [virtual]

Calculate number of channels. this function initializes the actual number of participating channel. Therefore, a vector is assigned (ch_Vec) that holds the number of active subunits. If the number of active subunits is equal or bigger than the number of subunits required to open a channel, the number of activated channel is increased.

Reimplemented from nw::_Species.

```
60                                      {
61     n_molecules = volume*1e-6*N_AVO*initial_conc;
62     // cout << "number of channel:" << n_molecules << endl;
63     ch_Vec.assign(n_molecules,0);
64     this->n_fully_inactivated_Ch = n_molecules;}
```

3.8.4 Member Data Documentation

3.8.4.1 vector<long>::iterator nw::Channel_Spc::ch_it [private]

vector iterator which iterates through the channel vector to update it

3.8.4.2 vector<long> nw::Channel_Spc::ch_Vec [private]

one item representing the active subunits for one channel

3.8.4.3 long nw::Channel_Spc::n_active_Ch [private]

number of currently active channels

3.8.4.4 long nw::Channel_Spc::n_fully_activated_Ch [private]

number of completely activated channels (no subunit activation possible).

3.8.4.5 long nw::Channel_Spc::n_fully_inactivated_Ch [private]

number of completely inactivated channels (no subunit inactivation possible)

3.8.4.6 int nw::Channel_Spc::n_Subunits [private]

number of identical subunits of this channel species.

3.8.4.7 int nw::Channel_Spc::n_Subunits_To_Open [private]

number of activated subunits that are required to open the channel.

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

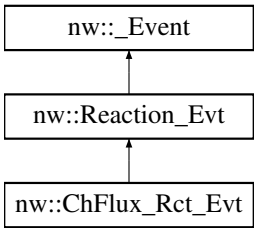
- Species/Channel_Spc.h
- Species/Channel_Spc.cpp

3.9 nw::ChFlux_Rct_Evt Class Reference

Channel flux reaction event. This event is derived by the reaction event and introduces the flux of a molecular species through a channel. It differs from an ordinary reaction event due to the fact that the educt (the channel) is not modified by the event itself. Thus the educt vector has to be generated manually in the init() function.

`#include <ChFlux_Rct_Evt.h>`

Inheritance diagram for nw::ChFlux_Rct_Evt:



Public Member Functions

- ChFlux_Rct_Evt (long id, string name, double k, vector< long > sc_vec, long ch_spc_id, VoxelVector vvc, Uni_Rnd *rg)
- virtual ~ChFlux_Rct_Evt ()
- void init ()

  Event initialization.
- double update (double)

  Updates the event.

Private Member Functions

- double calc_tau (long vid)

Private Attributes

- long ch_spc_id

Additional Inherited Members

3.9.1 Detailed Description

Channel flux reaction event. This event is derived by the reaction event and introduces the flux of a molecular species through a channel. It differs from an ordinary reaction event due to the fact that the educt (the channel) is not modified by the event itself. Thus the educt vector has to be generated manually in the init() function.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 nw::ChFlux_Rct_Evt::ChFlux_Rct_Evt (long *id*, string *name*, double *k*, vector< long > *sc_vec*, long *ch_spc_id*, VoxelVector *vvc*, Uni_Rnd * *rg*) [inline]

```
14      :Reaction_Evt(id,name,k,sc_vec,vvc,rg){
15          this->ch_spc_id = ch_spc_id;
16      }
```

3.9.2.2 virtual nw::ChFlux_Rct_Evt::~ChFlux_Rct_Evt ( ) [inline], [virtual]

```
17 {};
```

3.9.3 Member Function Documentation

3.9.3.1 double nw::ChFlux_Rct_Evt::calc_tau (long *vid*) [private]

```
42                                          {
43      tv_vec[vid].t = -log(rg->get_Uni_Rnd()) / (tv_vec.at(vid).v->get_state_vec()->
        at(educt_vec.at(0))->get_n_molecules() * c);
44      return tv_vec[vid].t;
45  }
```

3.9.3.2 void nw::ChFlux_Rct_Evt::init ( ) [virtual]

Event initialization.

This function builds the educt vector of an event and transforms its probability rate according to its reaction order. reaction order = educt_vetor.size()!

Implements nw::_Event.

```
6                           {
7       educt_vec.push_back(ch_spc_id);
8       c=k;
9       update(0);
10  }
```

3.9.3.3 double nw::ChFlux_Rct_Evt::update (double *last_tau*) [virtual]

Updates the event.

Returns

New tau value for this _Event

Parameters

last_tau	Tau value of previous firing _Event

Implementation of the update procedure. The tau value represents the time that has to pass until an event fires again. It has to be updated during every simulation step. An _Event with dirty_flag = TRUE has to generate a new random number to recalculate a tau value. An _Event with dirty_flag = FALSE is adapted to the global time scale by subtracting the last tau from their (necessarily larger) current tau. No negative tau values should be returned!

Implements nw::_Event.

```
17                                          {
18
19      double minTau = INFINITY;
20      double actTau;
```

```
21
22      for (long i = 0; i < (long) tv_vec.size(); i++) {
23          if (tv_vec.at(i).v->get_dirty_flag()) {
24              actTau = calc_tau(i);
25              if (actTau < minTau) {
26                  minTau = actTau;
27                  nextVoxel = i;
28              }
29          } else {
30              tv_vec.at(i).t -= last_tau;
31              if (tv_vec[i].t < minTau) {
32                  minTau = tv_vec[i].t;
33                  nextVoxel = i;
34              }
35          }
36      }
37      this->dirty_flag = false;
38
39      return minTau;
40  }
```

3.9.4 Member Data Documentation

3.9.4.1 long nw::ChFlux_Rct_Evt::ch_spc_id [private]

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

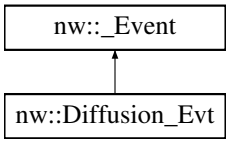
- Events/ChFlux_Rct_Evt.h
- Events/ChFlux_Rct_Evt.cpp

3.10 nw::Diffusion_Evt Class Reference

The diffusion event realizes the diffusion between two voxel.

`#include <Diffusion_Evt.h>`

Inheritance diagram for nw::Diffusion_Evt:



Public Member Functions

- Diffusion_Evt (long id, string name, double k, VoxelVector vvc, long diff_spc_id, Uni_Rnd *rg)

  Constructor of Diffusion Event.
- virtual ~Diffusion_Evt ()
- double update (double)

  overloaded function update() to reset propensities.
- void execute ()

  overloaded function execute(). The overloaded function execute() has to update two voxel. First of all the one which was chosen by the System. As a second step one of the neighbors, randomly has to be chosen and updated. The last step is to set dirty_flags of dependent events.
- void init ()

  overload function init(). initializes a diffusion event by calculating the propensity as well as calls the first update(). The state change vectors are created.
- double get_a ()

  returns a0

Protected Member Functions

- double calc_tau (long)

  function to calculate next tau value using the affected voxel id.

- void build_sc_vec ()

  builds the state change vector and the inverse state change vector of the diffusion event. this is just a little helper function to build an inverse sc_vector. Molecules that are increased in the chosen Voxel have to be decreased, while Molecules in the chosen neighbor are increased.

Protected Attributes

- vector< long > inv_sc_vec

  inverse sc_vec to update origin diffusion voxel (∗ -1)

- long diff_spec_id

  id of the species which diffusion is represented by this Event

3.10.1 Detailed Description

The diffusion event realizes the diffusion between two voxel.

Diffuion_Event uses the interface _Event to be regularly handled in the System class as an event. The Diffusion_↩ Event is besides the Reaction_Event the second fundamental Event in a System. Both together describe the basic molecular movement and development in a simulated system.
In contrast to the Reaction_Event the Diffusion_Event operates on two Voxel. Furthermore the execute() function needs another random number to choose the diffusion partner (anther voxel).

3.10.2 Constructor & Destructor Documentation

3.10.2.1 nw::Diffusion_Evt::Diffusion_Evt (long *id*, string *name*, double *k*, VoxelVector *vvc*, long *diff_spc_id*, Uni_Rnd ∗ *rg*) [inline]

Constructor of Diffusion Event.

Parameters

id	Every event has an ID for analysis purpose
name	Another characterization for analysis
k	rate constant of an event.
vvc	vector of all voxel where this event can occur
diff_spc_id	indicates the id of the diffusing molecule species
rg	Pointer to the Random_Generator

```
29                                                                                  :
30          _Event(id,name,k,vvc,rg){
31          this->diff_spec_id = diff_spc_id;
32          build_sc_vec();
33      }
```

3.10.2.2 virtual nw::Diffusion_Evt::~Diffusion_Evt ( ) [inline], [virtual]

```
35 {};
```

3.10.3 Member Function Documentation

3.10.3.1 void nw::Diffusion_Evt::build_sc_vec ( ) [protected]

builds the state change vector and the inverse state change vector of the diffusion event. this is just a little helper function to build an inverse sc_vector. Molecules that are increased in the chosen Voxel have to be decreased, while Molecules in the chosen neighbor are increased.

builds a state change vector using diff_spec_id

```
76                                  {
77 //  build sc_vec
78     sc_vec.assign(tv_vec[0].v->get_state_vec()->size(),0);
79     inv_sc_vec.assign(tv_vec[0].v->get_state_vec()->size(),0);
80     sc_vec[diff_spec_id] = -1;
81     inv_sc_vec[diff_spec_id] = 1;
82
83 }
```

3.10.3.2 double nw::Diffusion_Evt::calc_tau (long *vid*) [protected]

function to calculate next tau value using the affected voxel id.

Parameters

vid	the Voxel id of the voxel which has to be updated

Returns

new tau value

```
85                                      {
86 //  adapt the c value to the number of diffusion sites!
87     double c_adapted =  c * tv_vec.at(vid).v->get_diff_vec()->size();
88 //  calculate tau
89 //  cout << "tau V" << vid <<": " << tv_vec.at(vid).t << endl;
90     tv_vec[vid].t = -log(rg->get_Uni_Rnd()) / (tv_vec[vid].v->get_state_vec()->at(
       diff_spec_id)->get_n_molecules() * c_adapted);
91 //  cout << "updated Event: " << this->id << "Vxl: " << vid << endl;
92
93     // Monitoring update procedure
94 //  cout << this->id << ":" << endl;
95 //  cout << "vxl: " << vid << " -> tau: " << tv_vec[vid].t << endl;
96
97     return tv_vec[vid].t;
98 }
```

3.10.3.3 void nw::Diffusion_Evt::execute ( ) [virtual]

overloaded function execute(). The overloaded function execute() has to update two voxel. First of all the one which was chosen by the System. As a second step one of the neighbors, randomly has to be chosen and updated. The last step is to set dirty_flags of dependent events.

Implements nw::_Event.

```
21                              {
22 //  update diffusion origin voxel with the inverse state change vector
23     tv_vec[nextVoxel].v->update_state(sc_vec);
24
25 //  choose diffusion partner
26     double r = rg->get_Uni_Rnd();
27     double d; // tmp
28     for (size_t i = 0; i < tv_vec[nextVoxel].v->get_diff_vec()->size(); ++i){
29         d = (double)(i+1)/tv_vec[nextVoxel].v->get_diff_vec()->size();
30         if(r-d < 0){
31             tv_vec[nextVoxel].v->get_diff_vec()->at(i)->update_state(
       inv_sc_vec);
32             break;
33         }
```

```
34     }
35
36 //  set dirty flag to indicate that dependent events have to be updated properly
37     for(size_t i = 0; i < dep_list.size(); i++){
38         dep_list[i]->set_flag(true);
39     }
40 }
```

3.10.3.4 double nw::Diffusion_Evt::get_a ( ) [virtual]

returns a0

Returns

a0(sum over all a) this function reads the a values of every event and returns the sum over all (a0).

Implements nw::_Event.

```
100                             {
101     double a_ges = 0;
102     // calc a0.c
103     for(long i = 0; i < (long) tv_vec.size(); i++){
104         a_ges += c*tv_vec[i].v->get_state_vec()->at(diff_spec_id)->get_n_molecules() *
      tv_vec[i].v->get_diff_vec()->size();
105     }
106     return a_ges;
107 }
```

3.10.3.5 void nw::Diffusion_Evt::init ( ) [virtual]

overload function init(). initializes a diffusion event by calculating the propensity as well as calls the first update(). The state change vectors are created.

Implements nw::_Event.

```
9                         {
10
11 //  calculate the correct diffusion probability constant according to the voxel volume
12     this->c = (k*1e-3)/pow(tv_vec[0].v->get_box_length()*1e6,2);
13
14 //  build state change vector
15     build_sc_vec();
16
17 //  update the first time
18     update(0);
19 }
```

3.10.3.6 double nw::Diffusion_Evt::update (double *last_tau*) [virtual]

overloaded function update() to reset propensities.

Parameters

last_tau	tau value of the previous step The overloaded function update(), recalculates all tau values for every Voxel. Two cases have to be considered: • The tau values of those voxel where the diff_spec_id Molecules have been changed, the tau value has to be recalculated completely. • Tau values of all other voxel have to be decreased by the previous tau value (last_tau).

Implements nw::_Event.

```
42                                          {
43
44     double minTau = INFINITY, actTau;
45
46     if (this->dirty_flag) {
47         for (size_t i = 0; i < tv_vec.size(); ++i) {
48             if (tv_vec[i].v->get_dirty_flag()) {
49                 actTau = calc_tau(i);
50                 if (actTau < minTau) {
51                     minTau = actTau;
52                     nextVoxel = i;
53                 }
54             } else {
55                 tv_vec[i].t -= last_tau;
56                 if (tv_vec[i].t < minTau) {
57                     minTau = tv_vec[i].t;
58                     nextVoxel = i;
59                 }
60             }
61         }
62     } else {
63         for (size_t i = 0; i < tv_vec.size(); ++i) {
64             tv_vec[i].t -= last_tau;
65             if (tv_vec[i].t < minTau) {
66                 minTau = tv_vec[i].t;
67                 nextVoxel = i;
68             }
69         }
70     }
71
72     this->dirty_flag = false;
73     return minTau;
74 }
```

3.10.4 Member Data Documentation

3.10.4.1 long nw::Diffusion_Evt::diff_spec_id [protected]

id of the species which diffusion is represented by this Event

3.10.4.2 vector<long> nw::Diffusion_Evt::inv_sc_vec [protected]

inverse sc_vec to update origin diffusion voxel (∗ -1)

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

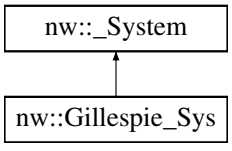
- Events/Diffusion_Evt.h
- Events/Diffusion_Evt.cpp

3.11 nw::Gillespie_Sys Class Reference

The Gillespie_Sys coordinates Gillespie's SSA and the whole output procedure. The whole logic of the Simulation Software is combined at this point. All Events and all Voxel are represented and important functions like build_↩ dependency_graph are located here.
Important fact is, that Gillespie_Sys just uses the Interfaces _Voxel and _Event what makes it really easy to extend the Simulation.

`#include <Gillespie_Sys.h>`

Inheritance diagram for nw::Gillespie_Sys:



Public Member Functions

- Gillespie_Sys (long, long, double, double, EventVector $*$, VoxelVector $*$, string, vector$<$ long $>$, vector$<$ long $>$)

  Instatiate the Environment for Gillespie's algorithm.
- virtual $\sim$Gillespie_Sys ()
- void go (long)

  Run simulation algorithm.

Private Member Functions

- void system_output ()

  file output of the system.
- void build_dep_graph ()

  build the dependency graph with defined Events.
- void print_system_state ()

  print current Gillespie_Sys State to console

Private Attributes

- long output_mode

  timing mode: [1] optimized data output for time series analysis (set sim time and n_steps according to 1/a0)
- size_t n_steps

  holds the number of simulation steps
- long opcntr

  output counter
- long channel_state

  tracks the channel state during the simulation to trigger the open/close timing output
- double max_Sim_Time

  maximal simulation time
- double rec_interval

  time interval, the Gillespie_Sys State is recorded
- double curr_time

  current time
- double rec_time

  recording interval counter
- double a_0

  the initial a_0 value of the system.
- string output_dir_path

  output directory path
- stringstream data_path

  data output path
- stringstream info_path

  simulation info output path
- stringstream oc_path

  event logging output path
- stringstream dcol_path

  data collection parameter path
- ofstream data_output_file

  output file stream for simulation data output
- ofstream info_output_file

output file stream for simulation info output

- ofstream oc_log_file

  output file stream for event logging

- ofstream dcol_file

  output file stream for data collection parameters

- vector< long > evt_cntr

  counts number of Event execution.

- vector< long > distr_vec

  holds Ca distribution

- vector< long > osid

  output species id vector

- vector< long > ovid

  output voxel id vector

- vector< _Event * > * evt_vec

  container for all applied Events to the Gillespie_Sys

- vector< _Voxel * > * vxl_vec

  container for all applied Voxel to the Gillespie_Sys

3.11.1 Detailed Description

The Gillespie_Sys coordinates Gillespie's SSA and the whole output procedure. The whole logic of the Simulation Software is combined at this point. All Events and all Voxel are represented and important functions like build_↩ dependency_graph are located here.
Important fact is, that Gillespie_Sys just uses the Interfaces _Voxel and _Event what makes it really easy to extend the Simulation.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 nw::Gillespie_Sys::Gillespie_Sys (long *output_mode*, long *n_steps*, double *max_Sim_Time*, double *rec_interval*, EventVector * *evt_vec*, VoxelVector * *vxl_vec*, string *output_dir_path*, vector< long > *osid*, vector< long > *ovid*)

Instatiate the Environment for Gillespie's algorithm.

Parameters

output_mode	determines weather
n_steps	number of simulation steps.
max_Sim_Time	maximum simulation time.
rec_interval	time interval, the whole system is recorded.
evt_vec	pointer to a vector of pointers to all defined Events.
vxl_vec	pointer to a vector of pointers to all defined Voxel.
osid	Output Species ID Vector
ovid	Output Voxel ID Vector
output_dir_path	Output Directory Path

```
17                                                                  :
18     output_mode(output_mode),
19     n_steps(n_steps),
20     opcntr(0),
21     max_Sim_Time(max_Sim_Time),
22     rec_interval(rec_interval),
23     curr_time(0),
24     rec_time(0),
25     a_0(0),
26     output_dir_path(output_dir_path),
27     osid(osid),
28     ovid(ovid),
```

```
29      evt_vec(evt_vec),
30      vxl_vec(vxl_vec)
31  {
32
33      evt_cntr.assign(evt_vec->size(), 0);      //  initialize event counter vector
34      distr_vec.assign(500, 0);               //  initialize distribution vector
35
36
37  //  initialize all Events
38      for (size_t i = 0; i < evt_vec->size(); ++i) {
39          evt_vec->at(i)->init();
40      }
41
42      this->a_0 = 0;
43  //  Calculate a_0 from all system events
44      for (size_t i = 0; i < this->evt_vec->size() ;++i){
45          this->a_0 += this->evt_vec->at(i)->get_a();
46      }
47
48  //  Adapt exit conditions and record interval according to the timing_mode
49      if (this->output_mode == 1){
50          // set the record interval to 10 * average system waiting time
51          this->rec_interval = 10 / a_0;
52      } else if (this->output_mode == 2){
53          this->rec_interval = 0;
54      }
55
56  //  print data output parameters for data analysis
57      this->dcol_path << output_dir_path << "sim_param.py";
58      const string &tmp_dcol = this->dcol_path.str();
59      this->dcol_file.open(tmp_dcol.c_str());
60      this->dcol_file << "mean_tau = " << 1/this->a_0 << endl << "dt = "
61                      << this->rec_interval << endl << "n_steps= " << this->
    n_steps
62                      << endl << "T_max = " << this->max_Sim_Time << endl << endl;
63      this->dcol_file.close();
64      cout    << "\n### Timing Parameters ###\n" << "mean_tau = " << 1/this->a_0
65              << endl << "dt = " << this->rec_interval << endl << "n_steps= "
66              << this->n_steps << endl << "T_max = " << this->max_Sim_Time
67              << "\n#######################" << endl << endl;
68
69  //  initialize channel state log
70      channel_state = 0;
71
72  //  build the dependency graph
73      build_dep_graph();
74
75  }
```

3.11.2.2 virtual nw::Gillespie_Sys::~Gillespie_Sys ( ) [inline], [virtual]

```
45 {}
```

3.11.3 Member Function Documentation

3.11.3.1 void nw::Gillespie_Sys::build_dep_graph ( ) [private]

build the dependency graph with defined Events.

The dependency graph is an important structure for the update procedure. Through analyzing the state_change_↩ vectors, dependencies are recognized and saved.
All sc-vecs are compared to each other. If sc_vec a at index k is != 0 and sc_vec b at index k is < 0, b is dependent on a.

```
230                                     {
231
232      try{
233  //      temporary vectors
234          vector<long> sc_a, sc_b;
235
236  //      find dependency relations based on state change vectors
237          for (size_t i = 0; i < evt_vec->size(); ++i){
238              sc_a = evt_vec->at(i)->get_sc_vec();
239              for (size_t j = 0; j < evt_vec->size(); ++j){
240                  sc_b = evt_vec->at(j)->get_sc_vec();
241                  if(evt_vec->at(i) != evt_vec->at(j)){
```

```
242                     for (size_t k = 0; k < sc_a.size(); ++k){
243                         if(sc_a.at(k) != 0 && sc_b.at(k) < 0 ){
244                             evt_vec->at(i)->add_dep_list(evt_vec->at(j));
245                             break;
246                         }
247                     }
248                 }
249             }
250         }
251     }
252
253     catch(exception& e){
254         cout << "System::build_dep_graph()" << e.what();
255     }
256 }
```

3.11.3.2 void nw::Gillespie_Sys::go (long *n_it*) [virtual]

Run simulation algorithm.

go() is the main function of the system. It contains the simulation algorithm and runs the whole simulation until the step number is reached. The algorithm consists of following steps: -> sort the event vector ascending of its tau values -> execute first Element of the event vector -> update all events -> clean voxel (set dirty_flags = FALSE) -> loop

Implements nw::_System.

```
79 {
80     try{
81
82 //      build path to output files. n_it represents the current simulation run.
83
84 //      data files containing molecular numbers
85         data_path << output_dir_path << "data_" << n_it << ".e";
86 //      open/close log for channel dynamics
87         oc_path << output_dir_path << "oc_log_" << n_it << ".e";
88 //      independent result log of every simulation run
89         info_path << output_dir_path << "sim_res.e";
90
91         const string &tmp_data = data_path.str();
92         const string &tmp_info = info_path.str();
93         const string &tmp_oclog = oc_path.str();
94
95 //      connect streams to output file
96         data_output_file.open(tmp_data.c_str());
97         oc_log_file.open(tmp_oclog.c_str());
98         oc_log_file << std::setprecision(10);
99
100 //      reconnect to existing file (adding text)
101         if (n_it == 1){
102             info_output_file.open(tmp_info.c_str());
103         }else{
104             info_output_file.open(tmp_info.c_str(), ios::in | ios::ate);
105         }
106
107 //      console output: indicate simulation start
108         info_output_file<< "###### Run: " << n_it << " ######" << endl;
109         cout << "--SIMULATION BEGIN-- \n" << endl;
110         info_output_file << "--SIMULATION RESULTS-- \n" << endl;
111
112 //      print the initial system state
113         print_system_state();
114
115 //      ##############################
116 //      ## START THE SIMULATION LOOP ##
117 //      ##############################
118
119 //      set times
120         double tau = 0, begin, end;
121         begin = time(0);
122
123 //      check exit conditions
124         for (size_t i = 0; i < n_steps; ++i) {
125             if (curr_time <= max_Sim_Time) {
126
127                 long nextIndex = 0;
128                 double minTau = INFINITY;
129
130 //              event update procedure
131                 for (size_t j = 0; j < evt_vec->size(); ++j) {
```

```
132                     double actTau = evt_vec->at(j)->update(tau);
133 //                  determine next Event Index
134                     if (actTau < minTau) {
135                         minTau = actTau;
136                         nextIndex = j;
137                     }
138                 }
139
140 //              update system times
141                 tau = minTau;
142                 rec_time += tau;
143                 curr_time += tau;
144
145 //              update event counter
146                 evt_cntr[nextIndex]++;
147
148 //              execute next Event
149                 evt_vec->at(nextIndex)->execute();
150
151 //              print system state if necessary
152                 system_output();
153
154             } else{
155                 n_steps = i;
156                 break;
157             }
158         }
159
160 //      ############################
161 //      ## END OF SIMULATION LOOP ##
162 //      ############################
163
164         end = time(0);
165
166 //      output of event distribution in info_output_file
167         cout << "Event distribution:\t";
168         info_output_file << "Event distribution:\t";
169         for (long p = 0; p < (long)evt_cntr.size(); p++){
170             cout << evt_cntr.at(p) << ",";
171             info_output_file << evt_cntr.at(p) << ",";
172         }
173         cout << endl;
174         info_output_file << endl;
175
176 //      system information output to info_output_file.
177         cout << "Simulated time:\t\t" << curr_time << " ms" << endl;
178         info_output_file << "Simulated time:\t\t" << curr_time << " ms" << endl;
179         cout << "Number of steps: \t" << n_steps << endl;
180         info_output_file << "Number of steps: \t" << n_steps << endl;
181         cout << "Computation time:\t" << end - begin << " s" << endl;
182         info_output_file << "Computation time:\t" << end - begin << " s" << endl;
183
184 //      console output: indicate simulation end
185         print_system_state();
186         cout << "\n--SIMULATION END--" << endl;
187         info_output_file << "\n--SIMULATION END--" << endl << endl << endl;
188
189 //      disconnect stream from output file
190         data_output_file.close();
191         info_output_file.close();
192         oc_log_file.close();
193     }
194
195     catch(exception& e){
196         cout << "System::go(): " << e.what() << endl;
197     }
198 }
```

3.11.3.3 void nw::Gillespie_Sys::print_system_state ( ) [private]

print current Gillespie_Sys State to console

```
258                                     {
259
260 //  print current system state
261     cout << "---System State---\n";
262     info_output_file << "---System State---\n";
263     for (long i = 0; i < (long)vxl_vec->size();i++){
264         cout << "Voxel " << i << ": ";
265         info_output_file << "Voxel " << i << ": ";
266         for (long j = 0; j < (long)vxl_vec->at(i)->get_state_vec()->size(); j++){
267             cout << vxl_vec->at(i)->get_state_vec()->at(j)->get_n_molecules() << ",";
```

```
268             info_output_file << vxl_vec->at(i)->get_state_vec()->at(j)->
    get_n_molecules() << ",";
269         }
270         cout << endl;
271         info_output_file << endl;
272     }
273     cout << "------------------\n";
274     info_output_file << "------------------\n";
275 }
```

3.11.3.4 void nw::Gillespie_Sys::system_output ( ) [inline], [private]

file output of the system.

The function system_output() takes care, that the whole system is recorded (rec_interval). Furthermore the number of Molecules are converted in concentration values.

```
200                                     {
201 //  print system state with a fixed time step dt (recording interval)
202     if (rec_time >= rec_interval){
203 //          run through output voxel vector
204             for(size_t k = 0; k < ovid.size(); ++k){
205 //              run through output species id vector
206                 for(size_t i = 0; i < osid.size(); ++i){
207                     data_output_file<<  vxl_vec->at(ovid[k])->get_state_vec()
208                                     ->at(osid[i])->get_n_molecules() << ",";
209                 }
210             }
211 //      }
212         if (this->output_mode == 2){
213             data_output_file << curr_time << ",";
214         }
215         opcntr++;
216         rec_time -= rec_interval;
217     }
218
219 //  check if the channel state changed and if so log the open and close times
220     if( vxl_vec->at(0)->get_state_vec()->at(vxl_vec->at(0)
221         ->get_state_vec()->size()-1)->get_n_molecules() != channel_state){
222 //      update current channel state
223         channel_state = vxl_vec->at(0)->get_state_vec()->at(
    vxl_vec->at(0)
224                         ->get_state_vec()->size() - 1)->get_n_molecules();
225 //      write current system time to oc_log.e
226         oc_log_file << curr_time << ",";
227     }
228 }
```

3.11.4 Member Data Documentation

3.11.4.1 double nw::Gillespie_Sys::a_0 [private]

the initial a_0 value of the system.

3.11.4.2 long nw::Gillespie_Sys::channel_state [private]

tracks the channel state during the simulation to trigger the open/close timing output

3.11.4.3 double nw::Gillespie_Sys::curr_time [private]

current time

3.11.4.4 ofstream nw::Gillespie_Sys::data_output_file [private]

output file stream for simulation data output

3.11.4.5 stringstream nw::Gillespie_Sys::data_path [private]

data output path

3.11.4.6 ofstream nw::Gillespie_Sys::dcol_file [private]

output file stream for data collection parameters

3.11.4.7 stringstream nw::Gillespie_Sys::dcol_path [private]

data collection parameter path

3.11.4.8 vector<long> nw::Gillespie_Sys::distr_vec [private]

holds Ca distribution

3.11.4.9 vector<long> nw::Gillespie_Sys::evt_cntr [private]

counts number of Event execution.

3.11.4.10 vector<_Event*>* nw::Gillespie_Sys::evt_vec [private]

container for all applied Events to the Gillespie_Sys

3.11.4.11 ofstream nw::Gillespie_Sys::info_output_file [private]

output file stream for simulation info output

3.11.4.12 stringstream nw::Gillespie_Sys::info_path [private]

simulation info output path

3.11.4.13 double nw::Gillespie_Sys::max_Sim_Time [private]

maximal simulation time

3.11.4.14 size_t nw::Gillespie_Sys::n_steps [private]

holds the number of simulation steps

3.11.4.15 ofstream nw::Gillespie_Sys::oc_log_file [private]

output file stream for event logging

3.11.4.16 stringstream nw::Gillespie_Sys::oc_path [private]

event logging output path

3.11.4.17 long nw::Gillespie_Sys::opcntr [private]

output counter

3.11.4.18 vector<long> nw::Gillespie_Sys::osid [private]

output species id vector

3.11.4.19 string nw::Gillespie_Sys::output_dir_path [private]

output directory path

3.11.4.20 long nw::Gillespie_Sys::output_mode [private]

timing mode: [1] optimized data output for time series analysis (set sim time and n_steps according to 1/a0)

3.11.4.21 vector<long> nw::Gillespie_Sys::ovid [private]

output voxel id vector

3.11.4.22 double nw::Gillespie_Sys::rec_interval [private]

time interval, the Gillespie_Sys State is recorded

3.11.4.23 double nw::Gillespie_Sys::rec_time [private]

recording interval counter

3.11.4.24 vector<_Voxel*>* nw::Gillespie_Sys::vxl_vec [private]

container for all applied Voxel to the Gillespie_Sys

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

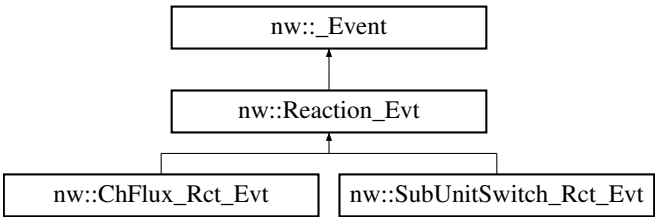
- System/Gillespie_Sys.h
- System/Gillespie_Sys.cpp

3.12 nw::Reaction_Evt Class Reference

The Reaction_Event realizes the Reaction of one ore two molecules.

`#include <Reaction_Evt.h>`

Inheritance diagram for nw::Reaction_Evt:



Public Member Functions

- Reaction_Evt (long id, string name, double k, vector< long > sc_vec, VoxelVector vvc, Uni_Rnd *rg)

  Constructor of Reaction Event.
- virtual ~Reaction_Evt ()
- double update (double)

  update procedure.
- void execute ()

  Event execution function.
- void init ()

  Event initialization.
- double get_a ()

  returns a0

Protected Member Functions

- void build_educt_vector ()

  builds the educt vector. The educt vector is a structure that subsumes pointer to the species that are educts of this event. Thus it is easy to check those species in changes. If so, this event gets ditry_flag = true and it has to be updated with a new random number
- double calc_tau (long)

  calculates a new tau value using a random number. The calc_tau function is dependent on the reaction order:

Protected Attributes

- vector< long > educt_vec

  The educt vector is a structure that subsumes pointer to the species that are educts of this event.

3.12.1 Detailed Description

The Reaction_Event realizes the Reaction of one ore two molecules.

Reaction_Event uses the interface _Event to be regularly handled in the System class as an event. The Reaction↩ _Event is besides the Diffusion_Event the second fundamental Event in a System. Both together describe the basic molecular movement and development in a simulated system.
In contrast to the Diffusion_Event the Reaction_Event just operates only on one Voxel at a time.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 nw::Reaction_Evt::Reaction_Evt (long *id*, string *name*, double *k*, vector< long > *sc_vec*, VoxelVector *vvc*, Uni_Rnd * *rg*) [inline]

Constructor of Reaction Event.

Parameters

id	ID of the Reaction_Event.
name	name or symbolic formula.
k	rate constant of an Event.

vvc	vector of all voxel where this event can occur.
sc_vec	state change vector of the Reaction Event.
rg	Pointer to the Random_Generator.

```
26                                                                                                  :
      _Event(id,name,k,vvc,rg){
27          this->sc_vec = sc_vec;
28      }
```

3.12.2.2 virtual nw::Reaction_Evt::~Reaction_Evt ( ) [inline],[virtual]

```
29 {};
```

3.12.3 Member Function Documentation

3.12.3.1 void nw::Reaction_Evt::build_educt_vector ( ) [protected]

builds the educt vector. The educt vector is a structure that subsumes pointer to the species that are educts of this event. Thus it is easy to check those species in changes. If so, this event gets ditry_flag = true and it has to be updated with a new random number

```
89                                      {
90      for(size_t i = 0; i < sc_vec.size(); ++i){
91          if(sc_vec[i] < 0)
92              educt_vec.push_back(i);
93      }
94  }
```

3.12.3.2 double nw::Reaction_Evt::calc_tau (long *vid*) [protected]

calculates a new tau value using a random number. The calc_tau function is dependent on the reaction order:

- 1. order: -ln(r)/a(j)(x,t) = -ln(r)/(educt ∗ c(j))
- 2. order: -ln(r)/a(j)(x,t) = -ln(r)/(educt(1) ∗ educt (2) ∗ c(j)) only first and second order reactions are allowed!

```
96                                      {
97      /*  INFOS
98      *   it is possible to implement  a dynamically calculated c value here, so that different voxel volumes
        are allowed. Problem is, more
99      *   computing effort.
100
101      *   a = (product of educts) * c
102      *   tau = -log(rnd)/a
103      */
104
105      if(educt_vec.size() > 0 && educt_vec.size() <= 2){    // make sure the educt vector
        is valid -> only first -and second order reactions are allowed
106          double a = c;
107          for (size_t i = 0; i < educt_vec.size(); ++i){
108              a *= tv_vec[vid].v->get_state_vec()->at(educt_vec[i])->get_n_molecules();
109          }
110          tv_vec[vid].t = -log(rg->get_Uni_Rnd()) / a;
111
112          // Monitoring update procedure
113  //      cout << this->id << ":" << endl;
114  //      cout << "vxl: " << vid << " -> tau: " << tv_vec[vid].t << endl;
115
116          return tv_vec[vid].t;
117      } else {
118          cout <<"ERROR: Only first and second order reactions are allowed! Check sc_vec of:" << this->
       name <<endl;
119          exit(0);
120      }
121
122  //  OLDER VERSION
123  //  if (educt_vec.size() == 1){         // 1. order Reaction
```

```
124 //      tv_vec.at(vid).t = -log(rg->get_Uni_Rnd()) /
        (tv_vec.at(vid).v->get_state_vec()->at(educt_vec.at(0))->get_n_molecules() * c);
125 //  }
126 //  else if (educt_vec.size() == 2){    // 2. order Reaction
127 //      tv_vec.at(vid).t = -log(rg->get_Uni_Rnd()) /
        (tv_vec.at(vid).v->get_state_vec()->at(educt_vec.at(0))->get_n_molecules() * tv_vec.at(vid).v->get_state_vec()->at(educ
128 //  }
129 //  else
130 //      cout <<"ERROR: Only first and second order reactions are allowed! Check sc_vec of:" << this->name
        <<endl;
131 }
```

3.12.3.3 void nw::Reaction_Evt::execute ( ) [virtual]

Event execution function.

Executes (fires) this _Event. It looks in the tv_struct for the voxel with the smallest associated tau value and calls its update_state() function.

Implements nw::_Event.

Reimplemented in nw::SubUnitSwitch_Rct_Evt.

```
28                          {
29      try{
30 //       execute event by using the state change vector
31          tv_vec[nextVoxel].v->update_state(sc_vec);
32
33 //       set dirty flag to indicate that dependent events have to be updated
34          this->dirty_flag = true;
35          for(size_t i = 0; i < dep_list.size(); i++){
36              dep_list[i]->set_flag(true);
37          }
38      }
39      catch(exception& e){
40          cout << "Reaction_Evt::execute(): " << e.what();
41      }
42 }
```

3.12.3.4 double nw::Reaction_Evt::get_a ( ) [virtual]

returns a0

Returns

a0(sum over all a) this function reads the a values of every event and returns the sum over all (a0).

Implements nw::_Event.

```
77                          {
78      double a_ges = 0;
79      for(size_t i = 0; i < tv_vec.size(); ++i){
80          double a = c;
81          for (size_t j = 0; j < educt_vec.size(); ++j){
82              a *= tv_vec[i].v->get_state_vec()->at(educt_vec[j])->get_n_molecules();
83          }
84          a_ges += a;
85      }
86      return a_ges;
87 }
```

3.12.3.5 void nw::Reaction_Evt::init ( ) [virtual]

Event initialization.

This function builds the educt vector of an event and transforms its probability rate according to its reaction order. reaction order = educt_vetor.size()!

Implements nw::_Event.

```
12                      {
13
14 //  build educt vector
15     build_educt_vector();
16
17 //  calculate the correct probability rate constant depending on the reaction order.
18     if (educt_vec.size() == 1){
19         this->c = k;
20     }else if(educt_vec.size() == 2){
21         this->c = k/(tv_vec[0].v->get_volume()*N_AVO*1e-6);
22     }
23
24 //  update the Reaction Event the first time
25     update(0);
26 }
```

3.12.3.6 double nw::Reaction_Evt::update (double *last_tau*) [virtual]

update procedure.

Parameters

last_tau	last tau value to update the system time.

Returns

The new tau value, so system can sort it right away. If dirty_flag is true, this event is dependent on the event, just executed before. In every voxel which was modified, the tau has to be updated properly (with new rnd number!). All the other tau values have to be adapted by subtracting the last tau value to keep the waiting times valid (updateing the system time) (Gibbson & Bruck).

Implements nw::_Event.

```
44                                          {
45
46     double minTau = INFINITY;
47
48     if(this->dirty_flag){
49         for (size_t i = 0; i < tv_vec.size(); ++i){
50             if (tv_vec[i].v->get_dirty_flag()){
51                 if(calc_tau(i) < minTau){
52                     minTau = tv_vec[i].t;
53                     nextVoxel = i;
54                 }
55             } else {
56                 tv_vec[i].t -= last_tau;
57                 if(tv_vec[i].t < minTau){
58                     minTau = tv_vec[i].t;
59                     nextVoxel = i;
60                 }
61             }
62         }
63     } else  {
64         for (size_t i = 0; i < tv_vec.size(); ++i) {
65             tv_vec[i].t -= last_tau;
66             if(tv_vec[i].t < minTau){
67                 minTau = tv_vec[i].t;
68                 nextVoxel = i;
69             }
70         }
71     }
72
73     this->dirty_flag = false;
74     return minTau;
75 }
```

3.12.4 Member Data Documentation

3.12.4.1 vector<long> nw::Reaction_Evt::educt_vec [protected]

The educt vector is a structure that subsumes pointer to the species that are educts of this event.

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

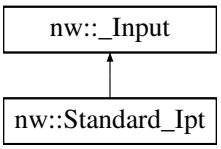
- Events/Reaction_Evt.h
- Events/Reaction_Evt.cpp

3.13 nw::Standard_Ipt Class Reference

Input class that parses a .xml files and generates a Gillespie_Sys.

`#include <Standard_Ipt.h>`

Inheritance diagram for nw::Standard_Ipt:



Classes

- struct xml_Ch_Spc

  Structure with attributes of the Channel_Spc class.
- struct xml_ChFlux_Rct_Evt

  Structure with attributes of the ChFlux_Rct_Evt class.
- struct xml_Dif_Evt

  Structure with attributes of the Diffusion_Evt class.
- struct xml_evt

  Structure with attributes of the _Event class.
- struct xml_gen_data

  Structure with attributes representing general data necessary for a system.
- struct xml_Rct_Evt

  Structure with attributes of the Reaction_Evt class.
- struct xml_spc

  Structure with attributes of the _Species class.
- struct xml_SuSwitch_Rct_Evt

  Structure with attributes of the SubUnitSwitch_Rct_Evt class.
- struct xml_vxl

  Structure with attributes of the _Voxel class.

Public Member Functions

- Standard_Ipt (string input_path, string output_dir_path)

  Constructor.
- ~Standard_Ipt ()

  Destructor.
- _System * get_System ()

  Create Gillespie_Sys from input .xml file.

Private Member Functions

- void read_Sim_Data ()

  Extract general simulation data and store it in the xgd struct.
- void read_Events (pugi::xml_node const &nod)

  Read events from parse result.
- void read_Voxel (pugi::xml_node const &nod)

  Read voxels from parse result.
- void read_Species (pugi::xml_node const &nod)

  Read species from parse result.
- void alloc_Events ()

  Create _Event objects.
- void alloc_Voxel ()

  Create _Voxel objects.
- void alloc_Species ()

  Create _Species objects.
- double get_a0 ()

  Calculate the sum of all event propensities a0.
- vector< _Voxel * > build_vxl_vec (xml_evt *const &evt)

  Build _Voxel vector for _Event objects.
- vector< long > build_sc_vector (xml_Rct_Evt *const &cxe)

  Build state change vector for Reaction_Evt objects @ param cxe xml_Rct_Evt @ return Reaction_Evt specific state change vector.
- vector< _Species * > clone_svc ()

  Clone _Species vector.
- string build_header ()

  Builds console output header.

Private Attributes

- string output_dir_path

  Output directory path.
- xml_gen_data xgd

  Structure holding general simulation data.
- pugi::xml_document doc

  DOM tree root (pugixml)
- char * p
- ofstream info_path

  Off-stream for sim_info file that summarizes parsed xml data.
- Uni_Rnd * rg

  Uniform random number generator.
- vector< _Event * > evc

  _Event vector
- vector< _Voxel * > vvc

  _Voxel vector
- vector< _Species * > svc

  _Specied vector

Additional Inherited Members

3.13.1 Detailed Description

Input class that parses a .xml files and generates a Gillespie_Sys.

The structure of the input .xml file represents the objects that need to be created to set up a model. The open source xml parser *pugixml* (http://pugixml.org/) version 0.9 is used. The input procedure is subdivided into three steps:

1. .xml data is parsed with *pugixml*.
2. Parse result is analyzed and model data temporarily stored in structures.
3. Objects are created using attributes of respective structures as parameters.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 nw::Standard_Ipt::Standard_Ipt (string *input_path*, string *output_dir_path*) [inline]

Constructor.

Parameters

input_path	Path to input file (.xml)
output_dir_path	Path to output directory

```
31                                                          {
32         this->p = (char*)input_path.c_str();
33         xml_parse_result result = doc.load_file(p); // load xml file
34         this->output_dir_path = output_dir_path;
35         rg = new Uni_Rnd();
36     }
```

3.13.2.2 nw::Standard_Ipt::~Standard_Ipt ( )

Destructor.

```
10                            {
11     if(rg){delete rg;rg=NULL;}
12     for (size_t i=0;i<xgd.xml_spc_list.size();++i){
13         if(xgd.xml_spc_list[i]){delete xgd.xml_spc_list[i];
       xgd.xml_spc_list[i]=NULL;}}
14     for (size_t i=0;i<xgd.xml_ch_spc_list.size();++i){
15         if(xgd.xml_ch_spc_list[i]){delete xgd.
       xml_ch_spc_list[i];xgd.xml_ch_spc_list[i]=NULL;}}
16     for (size_t i=0;i<xgd.xml_vxl_list.size();++i){if(xgd.
       xml_vxl_list[i]){
17         delete xgd.xml_vxl_list[i];xgd.xml_vxl_list[i]=NULL;}}
18     for (size_t i=0;i<xgd.xml_rct_evt_list.size();++i){
19         if(xgd.xml_rct_evt_list[i]){delete xgd.
       xml_rct_evt_list[i];xgd.xml_rct_evt_list[i]=NULL;}}
20     for (size_t i=0;i<xgd.xml_suSwitch_rct_evt_list.size();++i){
21         if(xgd.xml_suSwitch_rct_evt_list[i]){delete
       xgd.xml_suSwitch_rct_evt_list[i];
22         xgd.xml_suSwitch_rct_evt_list[i]=NULL;}}
23     for (size_t i=0;i<xgd.xml_chFlux_rct_evt_list.size();++i){
24         if(xgd.xml_chFlux_rct_evt_list[i]){delete xgd.
       xml_chFlux_rct_evt_list[i];
25         xgd.xml_chFlux_rct_evt_list[i]=NULL;}}
26     for (size_t i=0;i<xgd.xml_dif_evt_list.size();++i){if(xgd.
       xml_dif_evt_list[i]){
27         delete xgd.xml_dif_evt_list[i];xgd.
       xml_dif_evt_list[i]=NULL;}}
28     for (size_t i=0;i<svc.size();++i){if(svc[i]){delete svc[i];svc[i]=NULL;}}
29     for (size_t i=0;i<vvc.size();++i){if(vvc[i]){delete vvc[i];vvc[i]=NULL;}}
30     for (size_t i=0;i<evc.size();++i){if(evc[i]){delete evc[i];evc[i]=NULL;}}
31     if(s){delete s;s=NULL;}
32 }
```

3.13.3 Member Function Documentation

3.13.3.1 void nw::Standard_Ipt::alloc_Events ( ) [private]

Create _Event objects.

```
324                                {
325     cout << "alloc Events... ";
326
327 //  create Reaction Event Voxel Vector
328     vector<_Voxel*> re_vvc = vvc;     // reaction event voxel vector
329     re_vvc.pop_back();               // doesn't contain the border voxel!
330
331 //  add all Reaction Events to the Event vector
332     for (long i = 0; i < (long)xgd.xml_rct_evt_list.size(); i++){
333         xml_Rct_Evt* cxe = xgd.xml_rct_evt_list[i];
334         _Event* tre =   new Reaction_Evt(cxe->xml_id, cxe->xml_name, cxe->xml_k,
335                         build_sc_vector(cxe), build_vxl_vec(cxe),
    rg);
336         evc.push_back(tre);
337     }
338
339 //  add all Subunit Switch Reaction Events to the Event vector
340     for (long i = 0; i < (long)xgd.xml_suSwitch_rct_evt_list.size(); i++){
341         xml_SuSwitch_Rct_Evt* cxe = xgd.xml_suSwitch_rct_evt_list[i];
342         _Event* tsusre =    new SubUnitSwitch_Rct_Evt(cxe->xml_id, cxe->xml_name, cxe->xml_k,
343                             build_sc_vector(cxe), cxe->xml_actSu_id, cxe->xml_channel_id,
344                             build_vxl_vec(cxe), rg);
345         evc.push_back(tsusre);
346     }
347
348 //  add all Channel Flux Reaction Events to the Event vector
349     for (long i = 0; i < (long)xgd.xml_chFlux_rct_evt_list.size(); i++){
350         xml_ChFlux_Rct_Evt* cxe = xgd.xml_chFlux_rct_evt_list[i];
351         _Event* tcre =  new ChFlux_Rct_Evt(cxe->xml_id, cxe->xml_name, cxe->xml_k,
352                         build_sc_vector(cxe), cxe->xml_channel_id,
    build_vxl_vec(cxe), rg);
353         evc.push_back(tcre);
354     }
355
356 //  add all Diffusion Events to the Event vector
357     for (long i = 0; i < (long)xgd.xml_dif_evt_list.size(); i++){
358         xml_Dif_Evt* cxe = xgd.xml_dif_evt_list[i];
359         _Event* tde =   new Diffusion_Evt(cxe->xml_id, cxe->xml_name, cxe->xml_k,
360                         build_vxl_vec(cxe), cxe->xml_diff_spc,rg);
361         evc.push_back(tde);
362     }
363
364 //  output
365     info_path << "*** State change vectors ***" << endl;
366     for (long i = 0; i < (long)evc.size(); i++){
367         info_path<< evc.at(i)->get_id() << " :" << evc.at(i)->get_name() << " sc_vec: \t";
368         for (int j = 0; j < (int)evc.at(i)->get_sc_vec().size(); j++)
369             info_path<< evc.at(i)->get_sc_vec().at(j) << "," ;
370         info_path<< endl;
371     }
372
373     cout << "DONE" << endl;
374 }
```

3.13.3.2 void nw::Standard_Ipt::alloc_Species ( ) [private]

Create _Species objects.

```
258                                {
259     cout << "alloc Species... ";
260
261     for (long i = 0; i < (long) xgd.xml_spc_list.size(); i++){
262         _Species* ts =  new Standard_Spc(xgd.xml_spc_list[i]->xml_id,
    xgd.xml_spc_list[i]->xml_name,
263                         xgd.xml_spc_list[i]->xml_init_conc);
264         svc.push_back(ts);
265     }
266     for (long i = 0; i < (long) xgd.xml_ch_spc_list.size(); i++){
267         _Species* tcs = new Channel_Spc(xgd.xml_ch_spc_list[i]->xml_id,
    xgd.xml_ch_spc_list[i]->xml_name,
268                         xgd.xml_ch_spc_list[i]->xml_init_conc,
    xgd.xml_ch_spc_list[i]->xml_n_subunits,
269                         xgd.xml_ch_spc_list[i]->xml_n_suToOpen);
```

```
270         svc.push_back(tcs);
271     }
272 
273     cout << "DONE" << endl;
274 }
```

3.13.3.3 void nw::Standard_Ipt::alloc_Voxel ( ) [private]

Create _Voxel objects.

```
276                             {
277     cout << "alloc Voxel... ";
278 
279 //  border condition is equilibrium
280     if(xgd.xml_border_condition == "equilibrium"){
281 
282 //      run through voxl list and create Standard_Voxel
283         for (long i = 0; i < (long)xgd.xml_vxl_list.size(); i++){
284             _Voxel* tvxl = new Standard_Vxl(xgd.xml_vxl_list[i]->xml_id,
    xgd.xml_box_lenght, clone_svc());
285             vvc.push_back(tvxl);
286         }
287 
288 //      add border Voxel vvc.push_back(tbvxl);
289         _Voxel* bv = new Border_Vxl(vvc.size(), xgd.xml_box_lenght,
    clone_svc());
290         vvc.push_back(bv);
291 
292 //      initialize the diffusion neighbors including the border voxel
293         for (long i = 0; i < (long)xgd.xml_vxl_list.size(); i++){
294             for (long j = 0; j < (long)xgd.xml_vxl_list[i]->xml_vxl_neighbours.size(); j++)
295                 vvc[i]->add_neighbour(vvc.at(xgd.xml_vxl_list[i]->xml_vxl_neighbours[j
    ]));
296             for (long j = vvc[i]->get_diff_vec()->size(); j < 6; j++){
297 //              fill every free spot of the diffusion neighbour vector with border voxel
298 //              (last element of the voxel vector)
299                 vvc[i]->add_neighbour(vvc.at(vvc.size()-1));
300 //              add current voxel to the diffusion neighbour vector of the border voxel
301                 vvc[vvc.size()-1]->add_neighbour(vvc[i]);
302             }
303         }
304 
305 //  border condition is not equilibrium
306     } else{
307 //      run through voxel list and create Standard_Voxel
308         for (long i = 0; i < (long)xgd.xml_vxl_list.size(); i++){
309             _Voxel* tvxl = new Standard_Vxl(xgd.xml_vxl_list[i]->xml_id,
    xgd.xml_box_lenght, clone_svc());
310             vvc.push_back(tvxl);
311         }
312 //      initialize the diffusion neighbours including the border voxel
313         for (long i = 0; i < (long)xgd.xml_vxl_list.size(); i++){
314             for (long j = 0; j < (long)xgd.xml_vxl_list[i]->xml_vxl_neighbours.size(); j++){
315                 vvc[i]->add_neighbour(vvc.at(xgd.xml_vxl_list[i]->xml_vxl_neighbours[j
    ]));
316             }
317         }
318         cout << " *** NO BORDER VOXEL DEFINED *** ";
319     }
320 
321     cout << "DONE" << endl;
322 }
```

3.13.3.4 string nw::Standard_Ipt::build_header ( ) [private]

Builds console output header.

```
418                                 {
419     time_t rawtime;
420     time ( &rawtime );
421 
422     string tmp =    "STOCHASTIC SIMULATION SOFTWARE \n ------------------"
423                     "--------------------------------\n developed by Nicolas Wieder and "
424                     "Frederic von Wegner \n Medical Biophysics Group, Institute of "
425                     "Physiology and Pathophysiology,\n University of Heidelberg, "
426                     "69120 Heidelberg, Germany\n-----------------------------------"
427                     "----------------\n\n";
```

```
428     tmp += ctime (&rawtime);
429     return tmp;
430 }
```

3.13.3.5 vector< long > nw::Standard_Ipt::build_sc_vector (xml_Rct_Evt *const & *cxe*) [private]

Build state change vector for Reaction_Evt objects @ param cxe xml_Rct_Evt @ return Reaction_Evt specific state change vector.

```
383                                                                 {
384     vector<long>* scv  = new vector<long>;
385     scv->resize(vvc[cxe->xml_vxl[0]]->get_state_vec()->size());
386     long i;
387
388     for (i = 0; i < (long)cxe->xml_educts.size(); i++)
389         scv->at(cxe->xml_educts.at(i)) = -1;
390     for (i = 0; i < (long)cxe->xml_products.size(); i++)
391         scv->at(cxe->xml_products.at(i)) = 1;
392
393     return *scv;
394 }
```

3.13.3.6 vector< _Voxel * > nw::Standard_Ipt::build_vxl_vec (xml_evt *const & *evt*) [private]

Build _Voxel vector for _Event objects.

Parameters

evt	xml_evt structure

Returns

_Event specific voxel vector

The event definition inside the .xml file includes a list of _Voxel ids that represent the subset of _Voxel where the respective _Event can occur.
If the id list only contains one element with a value of -1, the event is can occur in all existing _Voxel.

```
396                                                             {
397     vector<_Voxel*> act_vvc;
398
399     if(evt->xml_vxl[0] == -1){
400         return vvc;
401     }
402
403     for(size_t i = 0; i < evt->xml_vxl.size(); ++i){
404         act_vvc.push_back(vvc[evt->xml_vxl[i]]);
405     }
406     return act_vvc;
407 }
```

3.13.3.7 vector< _Species * > nw::Standard_Ipt::clone_svc ( ) [private]

Clone _Species vector.

Returns

new instance of the _Species vector

```
409                                         {
410     vector<_Species*> svcClone;
411     svcClone.resize(svc.size());
412
413     for (long i = 0; i < (long) svc.size(); i++)
414         svcClone[i] = svc[i]->copy();
415     return svcClone;
416 }
```

3.13.3.8 double nw::Standard_Ipt::get_a0 ( ) [private]

Calculate the sum of all event propensities a0.

Returns

total event propensity a0

```
376                                {
377
378
379
380     return 0;
381 }
```

3.13.3.9 _System * nw::Standard_Ipt::get_System ( ) [virtual]

Create Gillespie_Sys from input .xml file.

Returns

Gillespie_Sys

Implements nw::_Input.

```
34                                      {
35     read_Sim_Data();
36 //  create Gillespie_Sys
37     s = new Gillespie_Sys(xgd.xml_output_mode, xgd.
       xml_max_sim_steps, xgd.xml_max_sim_time, xgd.
       xml_sample_rate,&evc, &vvc,output_dir_path,
       xgd.xml_output_Spc, xgd.xml_output_Vxl);
38     return s;
39 }
```

3.13.3.10 void nw::Standard_Ipt::read_Events (pugi::xml_node const & *nod*) [private]

Read events from parse result.

Parameters

nod	XML node *event_List*

```
155                                                 {
156     cout << "read_Events... ";
157     info_path << "############ EVENTS ############" << endl;
158     xml_node_iterator evt_it, li_it; // event iterator, special event iterator, list iterator
159
160 //  runs through every item of the Reaction Event List and creates the corresponding struct
161     for(evt_it = nod.child("reactionEvtList").begin(); evt_it != nod.child("reactionEvtList").end(); ++
       evt_it){
162         Standard_Ipt::xml_Rct_Evt* xevt = new Standard_Ipt::xml_Rct_Evt;
163         xevt->xml_id = atol(evt_it->attribute("id").value()); info_path<< "id: " << xevt->xml_id;
164         xevt->xml_name = (string)evt_it->child("name").child_value(); info_path<< ", name: " <<
       xevt->xml_name;
165         xevt->xml_k = atof(evt_it->child("k").child_value()); info_path<< ", k: " <<xevt->xml_k;
166         info_path<< ", Voxel: ";
167         for (li_it = evt_it->child("voxelVector").begin(); li_it != evt_it->child("voxelVector").end(); ++
       li_it){
168             xevt->xml_vxl.push_back(atol(li_it->child_value())); info_path<< li_it->child_value();
       }
169         info_path<< ", Educts: ";
170         for (li_it = evt_it->child("educts").begin(); li_it != evt_it->child("educts").end(); ++li_it){
171             xevt->xml_educts.push_back(atol(li_it->child_value()));info_path<< li_it->child_value(
       );}
172         info_path<< ", Products: ";
173         for (li_it = evt_it->child("products").begin(); li_it != evt_it->child("products").end(); ++li_it){
174             xevt->xml_products.push_back(atol(li_it->child_value()));info_path<< li_it->
       child_value();}
175         info_path<< endl;
```

```
176         xgd.xml_rct_evt_list.push_back(xevt);
177     }
178 
179 //  runs through every item of the Reaction Sub Unit Switch Reaction Event List and
180 //  creates the corresponding struct
181     for(evt_it = nod.child("subUnitSwitchRctEvtList").begin(); evt_it != nod.child("subUnitSwitchRctEvtList
    ").end(); ++evt_it){
182         Standard_Ipt::xml_SuSwitch_Rct_Evt* xevt = new Standard_Ipt::xml_SuSwitch_Rct_Evt;
183         xevt->xml_id = atol(evt_it->attribute("id").value());
184         info_path<< "id: " << xevt->xml_id;
185         xevt->xml_name = (string)evt_it->child("name").child_value();
186         info_path<< ", name: " << xevt->xml_name;
187         xevt->xml_k = atof(evt_it->child("k").child_value());
188         info_path<< ", k: " <<xevt->xml_k;
189         xevt->xml_channel_id = atol(evt_it->child("channel").child_value());
190         info_path<< ", channel; " << xevt->xml_channel_id;
191         xevt->xml_actSu_id = atol(evt_it->child("actSubUnitID").child_value());
192         info_path<< ", act su ID: " << xevt->xml_actSu_id;
193 
194         info_path<< ", Voxel: ";
195         for (li_it = evt_it->child("voxelVector").begin(); li_it != evt_it->child("voxelVector").end(); ++
    li_it){
196             xevt->xml_vxl.push_back(atol(li_it->child_value())); info_path<< li_it->child_value();
197         }
198         info_path<< ", Educts: ";
199         for (li_it = evt_it->child("educts").begin(); li_it != evt_it->child("educts").end(); ++li_it){
200             xevt->xml_educts.push_back(atol(li_it->child_value()));info_path<< li_it->child_value(
    );
201         }
202         info_path<< ", Products: ";
203         for (li_it = evt_it->child("products").begin(); li_it != evt_it->child("products").end(); ++li_it){
204             xevt->xml_products.push_back(atol(li_it->child_value()));info_path<< li_it->
    child_value();
205         }
206         info_path<< endl;
207         xgd.xml_suSwitch_rct_evt_list.push_back(xevt);
208     }
209 
210 //  runs through every item of the Channel Flux Reaction Event List and creates the corresponding struct
211     for(evt_it = nod.child("channelFluxEvtList").begin(); evt_it != nod.child("channelFluxEvtList").end();
    ++evt_it){
212         Standard_Ipt::xml_ChFlux_Rct_Evt* xevt = new Standard_Ipt::xml_ChFlux_Rct_Evt;
213         xevt->xml_id = atol(evt_it->attribute("id").value()); info_path<< "id: " << xevt->xml_id;
214         xevt->xml_name = (string)evt_it->child("name").child_value();
215         info_path<< ", name: " << xevt->xml_name;
216         xevt->xml_k = atof(evt_it->child("k").child_value()); info_path<< ", k: " <<xevt->xml_k;
217         xevt->xml_channel_id = atol(evt_it->child("channel").child_value());
218         info_path<< ", channel; " << xevt->xml_channel_id;
219 
220         info_path<< ", Voxel: ";
221         for (li_it = evt_it->child("voxelVector").begin(); li_it != evt_it->child("voxelVector").end(); ++
    li_it){
222             xevt->xml_vxl.push_back(atol(li_it->child_value())); info_path<< li_it->child_value();
223         }
224 
225         info_path<< ", Products: ";
226         for (li_it = evt_it->child("products").begin(); li_it != evt_it->child("products").end(); ++li_it){
227             xevt->xml_products.push_back(atol(li_it->child_value()));info_path<< li_it->
    child_value();
228         }
229         info_path<< endl;
230         xgd.xml_chFlux_rct_evt_list.push_back(xevt);
231     }
232 
233 //  runs through every item of the Diffusion Event List and creates the corresponding struct
234     for(evt_it = nod.child("diffusionEvtList").begin(); evt_it != nod.child("diffusionEvtList").end(); ++
    evt_it){
235         Standard_Ipt::xml_Dif_Evt* xevt = new Standard_Ipt::xml_Dif_Evt;
236         xevt->xml_id = atol(evt_it->attribute("id").value()); info_path<< "id: " << xevt->xml_id;
237         xevt->xml_name = (string)evt_it->child("name").child_value(); info_path<< ", name: " <<
    xevt->xml_name;
238         xevt->xml_k = atof(evt_it->child("k").child_value()); info_path<< ", k: " <<xevt->xml_k;
239         xevt->xml_diff_spc = atol(evt_it->child("diffusionSpcID").child_value());
240         info_path<< ", diffSpcID: " << xevt->xml_diff_spc;
241 
242         info_path<< ", Voxel: ";
243         for (li_it = evt_it->child("voxelVector").begin(); li_it != evt_it->child("voxelVector").end(); ++
    li_it){
244             xevt->xml_vxl.push_back(atol(li_it->child_value())); info_path<< li_it->child_value();
245         }
246 //      add the border voxel to the voxel vector of all diffusion events.
247         if (xgd.xml_border_condition == "equilibrium"){
248 //      since the border voxel doesn't exist yet, xgd.xml_vxl_list.size() is the correct index.
249             xevt->xml_vxl.push_back(xgd.xml_vxl_list.size());
    info_path << (xgd.xml_vxl_list.size());
250         }
251         info_path<< endl;
```

```
252
253         xgd.xml_dif_evt_list.push_back(xevt);
254     }
255     cout << "DONE" << endl;
256 }
```

3.13.3.11 void nw::Standard_Ipt::read_Sim_Data ( ) [private]

Extract general simulation data and store it in the xgd struct.

```
41                                {
42
43     xml_node top = doc.child("Sim");
44     xml_node_iterator it = doc.begin();
45     xml_node_iterator li_it; // iterator for the outputSpeciesList
46
47 //  open output file stream
48     string tmp = output_dir_path + "sim_info" + ".e";
49     info_path.open(tmp.c_str());
50     info_path << build_header() << endl;
51     info_path << "############ GENERAL DATA ############" << endl;
52
53 //  read the general data and save to xgd
54     xgd.xml_output_mode = atol(it->child_value("outputMode"));
55     info_path << "output mode: " << xgd.xml_output_mode << endl;
56     xgd.xml_max_sim_steps = atof(it->child_value("maxSimulationSteps"));
57     info_path << "max simulation steps: " << xgd.xml_max_sim_steps << endl;
58     xgd.xml_max_sim_time = atof(it->child_value("maxSimTime"));
59     info_path<< "max simulation time: " << xgd.xml_max_sim_time << endl;
60     xgd.xml_sample_rate = atof(it->child_value("sampleRate"));
61     info_path<< "sample rate: " << xgd.xml_sample_rate << endl;
62     xgd.xml_box_lenght = atof(it->child_value("voxelBoxLenght"));
63     info_path<< "voxel box lenght: " << xgd.xml_box_lenght << endl;
64     xgd.xml_border_condition = (string)it->child_value("borderCondition");
65     info_path<< "border condition: " << xgd.xml_border_condition << endl;
66
67 //  print output _Species and _Voxel ids to sim_info.e
68     info_path<< "output species: ";
69     for (li_it = it->child("outputSpcList").begin(); li_it != it->child("outputSpcList").end(); ++li_it){
70         xgd.xml_output_Spc.push_back(atol(li_it->child_value()));
       info_path << li_it->child_value() << ",";
71     }
72     info_path << endl << "output voxel: " ;
73     for (li_it = it->child("outputVxlList").begin(); li_it != it->child("outputVxlList").end(); ++li_it){
74         xgd.xml_output_Vxl.push_back(atol(li_it->child_value()));
      info_path<< li_it->child_value() << ",";
75     }
76     info_path << endl;
77
78     read_Species(it->child("speciesList"));
79     read_Voxel(it->child("voxelList"));
80     read_Events(it->child("eventList"));
81
82     alloc_Species();
83     alloc_Voxel();
84     alloc_Events();
85
86 //  close the file stream
87     info_path << endl << endl << endl;
88     info_path.close();
89 }
```

3.13.3.12 void nw::Standard_Ipt::read_Species (pugi::xml_node const & *nod*) [private]

Read species from parse result.

Parameters

nod	XML node *species_list*

```
91                                          {
92     cout << "read_Species...",
93     info_path << "############ SPECIES ############" << endl;
94
95     xml_node_iterator spc_it;  //voxel iterator, voxel element iterator, list iterator
```

```
96
97 //  read all Standard Species
98     for (spc_it = nod.child("standardSpcList").begin(); spc_it != nod.child("standardSpcList").end(); ++
       spc_it){
99 //      create new standard species struct
100         Standard_Ipt::xml_spc* xspc = new Standard_Ipt::xml_spc;
101         xspc->xml_id = atol(spc_it->attribute("id").value());
102         info_path<< "id: " << xspc->xml_id;
103         xspc->xml_name = (string)spc_it->child_value("name");
104         info_path<< ", name: " << xspc->xml_name;
105         xspc->xml_init_conc = atof(spc_it->child_value("initialConcentration"));
106         info_path<< ", initConc:" << xspc->xml_init_conc;
107         info_path<< endl;
108         xgd.xml_spc_list.push_back(xspc);
109     }
110
111 //  read all Channel Species
112     for (spc_it = nod.child("channelSpcList").begin(); spc_it != nod.child("channelSpcList").end(); ++
       spc_it){
113 //      create new channel species struct
114         Standard_Ipt::xml_Ch_Spc* xspc = new Standard_Ipt::xml_Ch_Spc;
115         xspc->xml_id = atol(spc_it->attribute("id").value());
116         info_path<< "id: " << xspc->xml_id;
117         xspc->xml_name = (string)spc_it->child_value("name");
118         info_path<< ", name: " << xspc->xml_name;
119         xspc->xml_init_conc = atof(spc_it->child_value("initialConcentration"));
120         info_path<< ", initConc:" << xspc->xml_init_conc;
121         xspc->xml_n_subunits = atol(spc_it->child_value("nSubUnits"));
122         info_path<< ", n_subunits:" << xspc->xml_n_subunits;
123         xspc->xml_n_suToOpen = atol(spc_it->child_value("nSubUnitsToOpen"));
124         info_path<< ", n_suToOpen:" << xspc->xml_n_suToOpen;
125         info_path<< endl;
126         xgd.xml_ch_spc_list.push_back(xspc);
127     }
128
129     cout << "DONE" << endl;
130 }
```

3.13.3.13 void nw::Standard_Ipt::read_Voxel (pugi::xml_node const & *nod*) [private]

Read voxels from parse result.

Parameters

nod	XML node *voxel_List*

```
132                                                 {
133     cout << "read_Voxel...";
134     info_path << "############ VOXEL ############" << endl;
135     xml_node_iterator vxl_it, li_it; //voxel iterator
136
137 //  read all Standard Voxel
138     for(vxl_it = nod.child("standardVxlList").begin(); vxl_it != nod.child("standardVxlList").end(); ++
       vxl_it){
139         Standard_Ipt::xml_vxl* xvxl = new Standard_Ipt::xml_vxl;
140         xvxl->xml_id = atol(vxl_it->attribute("id").value()); info_path<< "id: " << xvxl->xml_id;
141         info_path<< ", Initial State: ";
142         for (li_it = vxl_it->child("initialState").begin(); li_it != vxl_it->child("initialState").end(); +
       +li_it){
143             xvxl->xml_init_state.push_back(atol(li_it->child_value()));
       info_path<< li_it->child_value();
144         }
145         info_path<< ", Neighbours: ";
146         for (li_it = vxl_it->child("neighbours").begin(); li_it != vxl_it->child("neighbours").end(); ++
       li_it){
147             xvxl->xml_vxl_neighbours.push_back(atol(li_it->child_value()));
       info_path<< li_it->child_value();
148         }
149         info_path<< endl;
150         xgd.xml_vxl_list.push_back(xvxl);
151     }
152     cout << "DONE" << endl;
153 }
```

3.13.4 Member Data Documentation

3.13.4.1 pugi::xml_document nw::Standard_Ipt::doc [private]

DOM tree root (pugixml)

3.13.4.2 vector<_Event*> nw::Standard_Ipt::evc [private]

_Event vector

3.13.4.3 ofstream nw::Standard_Ipt::info_path [private]

Off-stream for sim_info file that summarizes parsed xml data.

3.13.4.4 string nw::Standard_Ipt::output_dir_path [private]

Output directory path.

3.13.4.5 char* nw::Standard_Ipt::p [private]

3.13.4.6 Uni_Rnd* nw::Standard_Ipt::rg [private]

Uniform random number generator.

3.13.4.7 vector<_Species*> nw::Standard_Ipt::svc [private]

_Specied vector

3.13.4.8 vector<_Voxel*> nw::Standard_Ipt::vvc [private]

_Voxel vector

3.13.4.9 xml_gen_data nw::Standard_Ipt::xgd [private]

Structure holding general simulation data.

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

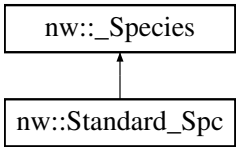
- Input/Standard_Ipt.h
- Input/Standard_Ipt.cpp

3.14 nw::Standard_Spc Class Reference

A Standard_Spc is an object that holds the properties of a special kind of molecules.

`#include <Standard_Spc.h>`

Inheritance diagram for nw::Standard_Spc:



Public Member Functions

- Standard_Spc (long id, std::string name, double initial_conc)

  Constructor of Standard_Spc.
- ~Standard_Spc ()
- long get_n_molecules ()

  Get current number of molecules.
- void mod_n_molecules (long n)

  Modify number of molecules (n_molecules).
- _Species * copy ()

  Copy method.

Additional Inherited Members

3.14.1 Detailed Description

A Standard_Spc is an object that holds the properties of a special kind of molecules.

Basic unit of a Gillespie_Sys. Every _Voxel has a state_vector that consists of molecular _Species. A _Species is defined by its current number of molecules, its id and name, and a dirty_flag. The dirty flag indicates that the species has been updated during the previous _Event.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 nw::Standard_Spc::Standard_Spc (long *id*, std::string *name*, double *initial_conc*) [inline]

Constructor of Standard_Spc.

Parameters

id	Species ID. Note that the ID of a molecular species hast to be equal to its voxel state vector index.
name	Species name
initial_conc	Initial concentration

```
22 :_Species(id,name,initial_conc){}
```

3.14.2.2 nw::Standard_Spc::~Standard_Spc ( ) [inline]

```
23 {}
```

3.14.3 Member Function Documentation

3.14.3.1 _Species* nw::Standard_Spc::copy ( ) [inline], [virtual]

Copy method.

Returns

Reference to an object identical to this

Copy method that generates a copy of this. It is is a virtual function to ensure that whenever this function is called, the most specific object (derived from this abstract base class) is copied.

Implements nw::_Species.

```
28                      {
29          _Species* s = new Standard_Spc(id,name,
        initial_conc);
30          return s;}
```

3.14.3.2 long nw::Standard_Spc::get_n_molecules ( ) [inline],[virtual]

Get current number of molecules.

Returns

current number of molecules

Implements nw::_Species.

```
25 {return n_molecules;}
```

3.14.3.3 void nw::Standard_Spc::mod_n_molecules (long *n*) [inline],[virtual]

Modify number of molecules (n_molecules).

Parameters

n	Summand that is added to the current number of molecules.

If an event occurs, this function is called to update the number of molecules based on the state change vector of the firing _Event. If `n` is positive the number of molecules is increased, if it is negative the number of molecules in decrease.

Implements nw::_Species.

```
26                                  {
27          n_molecules += n; if(n!=0){dirty_flag = true;}}
```

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

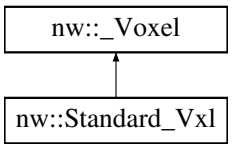
- Species/Standard_Spc.h

3.15 nw::Standard_Vxl Class Reference

Standard voxel implementation.

`#include <Standard_Vxl.h>`

Inheritance diagram for nw::Standard_Vxl:



Public Member Functions

- Standard_Vxl (long id, double box_length, SpeciesVector state_vec)

  Constructor of Standard Voxel.
- ~Standard_Vxl ()
- void update_state (vector< long >)

  updates the state of voxel

Additional Inherited Members

3.15.1 Detailed Description

Standard voxel implementation.

Most importantly it implements the `update_state()` function of its mother class, so that the state vector (state_vec) can be modified by events (in contrast to Border_Vxl).

3.15.2 Constructor & Destructor Documentation

3.15.2.1 nw::Standard_Vxl::Standard_Vxl (long *id*, double *box_length*, SpeciesVector *state_vec*) [inline]

Constructor of Standard Voxel.

Parameters

id	The ID of the voxel
box_length	The box_lenght of the voxel um
state_vec	Represents all Standard_Spc in this voxel

```
22 :_Voxel(id,box_length,state_vec){}
```

3.15.2.2 nw::Standard_Vxl::~Standard_Vxl ( ) [inline]

```
23 {};
```

3.15.3 Member Function Documentation

3.15.3.1 void nw::Standard_Vxl::update_state (vector< long > *sc_vec*) [virtual]

updates the state of voxel

Implements nw::_Voxel.

```
8 {
9     if (sc_vec.size() == state_vec.size()) {
10     //  update the state vector of voxel using the assigned state change vector (usually called by an
       event)
11         for (size_t i = 0; i < state_vec.size(); ++i) {
12             state_vec[i]->mod_n_molecules(sc_vec[i]);
13         }
14         this->dirty_flag = true;
15     } else {
16     //  Debug information
17         cout << "ERROR: state change vector and state vector differ in size";
18     }
19 }
```

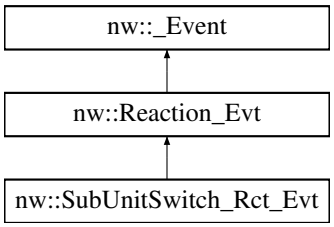
The documentation for this class was generated from the following files:

- Voxel/Standard_Vxl.h
- Voxel/Standard_Vxl.cpp

3.16 nw::SubUnitSwitch_Rct_Evt Class Reference

`#include <SubUnitSwitch_Rct_Evt.h>`

Inheritance diagram for nw::SubUnitSwitch_Rct_Evt:



Public Member Functions

- SubUnitSwitch_Rct_Evt (long id, string name, double k, vector< long > sc_vec, long act_su_id, long ch_↩ spc_id, VoxelVector vvc, Uni_Rnd *rg)
- virtual ~SubUnitSwitch_Rct_Evt ()
- void execute ()

  Event execution function.

Private Member Functions

- void update_Channel_Spc ()

  updates the referenced channel Species every time this event is executed.

Private Attributes

- Channel_Spc * chs
- long act_su_id

  species id of the activated subunit
- long ch_spc_id

Additional Inherited Members

3.16.1 Constructor & Destructor Documentation

3.16.1.1 nw::SubUnitSwitch_Rct_Evt::SubUnitSwitch_Rct_Evt (long *id,* string *name,* double *k,* vector< long > *sc_vec,* long *act_su_id,* long *ch_spc_id,* VoxelVector *vvc,* Uni_Rnd * *rg*) [inline]

```
21
                                        :Reaction_Evt(id, name, k ,
      sc_vec,vvc,rg){
22          this->act_su_id = act_su_id;
23          this->ch_spc_id = ch_spc_id;
24      }
```

3.16.1.2 virtual nw::SubUnitSwitch_Rct_Evt::~SubUnitSwitch_Rct_Evt ( ) [inline], [virtual]

```
25 {};
```

3.16.2 Member Function Documentation

3.16.2.1 void SubUnitSwitch_Rct_Evt::execute ( ) [virtual]

Event execution function.

Executes (fires) this _Event. It looks in the tv_struct for the voxel with the smallest associated tau value and calls its update_state() function.

Reimplemented from nw::Reaction_Evt.

```
15                                      {
16 //  cout << "--- " << tv_vec.at(0).v->get_id() << " ---" << endl;
17
18     try{
19 //      execute event by using the state change vector
20         tv_vec[nextVoxel].v->update_state(sc_vec);
21
22 //      set dirty flag to indicate that dependent events have to be updated properly
23         for(long i = 0; i < (long)dep_list.size(); i++){
24             dep_list[i]->set_flag(true);
25         }
26
27 //      update the activation states of the subunits of chs
28         chs = dynamic_cast<Channel_Spc*>(tv_vec[nextVoxel].v->get_state_vec()
      ->at(ch_spc_id));  // cave: downcasting (_Species->Channel_Spc)...
29         update_Channel_Spc();
30     }
31     catch(exception& e){
32         cout << "SubuintSwitch_Rct_evt::execute(): " << e.what();
33         }
34     }
```

3.16.2.2 void SubUnitSwitch_Rct_Evt::update_Channel_Spc ( ) [private]

updates the referenced channel Species every time this event is executed.

```
36                                              {
37
38     if (sc_vec.at(act_su_id) > 0){
39         long r_chid = (long)(rg->get_Uni_Rnd() * chs->
      get_n_Activable_Ch());
40         chs->activate_Subunit(r_chid);
41     }else if(sc_vec.at(act_su_id) < 0){
42         long r_chid = (long)(rg->get_Uni_Rnd() * chs->
      get_n_Inactivable_Ch());
43         chs->inactivate_Subunit(r_chid);
44     }else{
45         cout << "ERROR: The SubUnitActSwitch_Rct_Evt state change vector doesn't suit the assigned
       active_subUnit_Species_ID. Please check your input!" << endl;
46     }
47 }
```

3.16.3 Member Data Documentation

3.16.3.1 long nw::SubUnitSwitch_Rct_Evt::act_su_id [private]

species id of the activated subunit

3.16.3.2 long nw::SubUnitSwitch_Rct_Evt::ch_spc_id [private]

3.16.3.3 Channel_Spc* nw::SubUnitSwitch_Rct_Evt::chs [private]

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

- Events/SubUnitSwitch_Rct_Evt.h
- Events/SubUnitSwitch_Rct_Evt.cpp

3.17 nw::_Event::tv_struct Struct Reference

tau-voxel-structure (tv_struct) is a structure that associates a tau value with a _Voxel.

`#include <_Event.h>`

Public Attributes

- _Voxel * v
  pointer to an existing voxel
- double t
  corresponding tau value

3.17.1 Detailed Description

tau-voxel-structure (tv_struct) is a structure that associates a tau value with a _Voxel.

`v` points to a _Voxel, while `t` is the corresponding tau value. It further overloads the < operator to implement the 'smaller than' operation of two tv_structs based on their tau value.

3.17.2 Member Data Documentation

3.17.2.1 double nw::_Event::tv_struct::t

corresponding tau value

3.17.2.2 _Voxel* nw::_Event::tv_struct::v

pointer to an existing voxel

The documentation for this struct was generated from the following file:

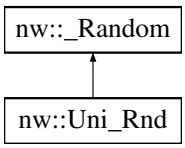
- Events/_Event.h

3.18 nw::Uni_Rnd Class Reference

"Minimal" random number generator of Park and Miller

`#include <Uni_Rnd.h>`

Inheritance diagram for nw::Uni_Rnd:



Public Member Functions

- Uni_Rnd ()
  Seed random generator with current time.
- double get_Uni_Rnd ()
  Generates uniform random variable.

Private Member Functions

- double uni_Rnd (long *)
  generates uniform random variable

Private Attributes

- long seed

  Random generator seed.

3.18.1 Detailed Description

"Minimal" random number generator of Park and Miller

with Bays-Durham shuffle and added safeguards. Returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint values). Call with idum a negative integer to initialize; thereafter, do not alter idum between successive deviates in a sequence. RNMX should approximate the largest floating value that is less than 1.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 nw::Uni_Rnd::Uni_Rnd ( ) [inline]

Seed random generator with current time.

```
19 {seed = long(time(0)) * -1;}
```

3.18.3 Member Function Documentation

3.18.3.1 double Uni_Rnd::get_Uni_Rnd ( ) [virtual]

Generates uniform random variable.

Returns

Uniform random deviate between 0.0 and 1.0

Implements nw::_Random.

```
53                              {
54     double r = uni_Rnd(&seed);
55     return r;
56 }
```

3.18.3.2 double Uni_Rnd::uni_Rnd (long * *idum*) [private]

generates uniform random variable

Returns

uniform random deviate between 0.0 and 1.0

```
22                                  {
23
24     int j;
25     long k;
26     static long iy=0;
27     static long iv[NTAB];
28     double temp;
29     if (*idum <= 0 || !iy){  //Initialize.
30        if (-(*idum) < 1) *idum=1; // Be sure to prevent idum = 0.
31        else *idum = -(*idum);
32        for (j=NTAB+7;j>=0;j--){ //Load the shuffle table (after 8 warm-ups).
33            k=(*idum)/IQ;
34            *idum=IA*(*idum-k*IQ)-IR*k;
35            if (*idum < 0) *idum += IM;
36            if (j < NTAB) iv[j] = *idum;
```

```
37         }
38         iy=iv[0];
39     }
40     k=(*idum)/IQ; // Start here when not initializing.
41     *idum=IA*(*idum-k*IQ)-IR*k; // Compute idum=(IA*idum) % IM without over
42
43     if (*idum < 0) *idum += IM; // flows by Schrage's method.
44
45     j=iy/NDIV; // Will be in the range 0..NTAB-1.
46     iy=iv[j]; // Output previously stored value and refill the
47     iv[j] = *idum; // shuffle table.
48
49     if ((temp=AM*iy) > RNMX) return RNMX; // Because users don't expect endpoint values.
50     else return temp;
51 }
```

3.18.4 Member Data Documentation

3.18.4.1 long nw::Uni_Rnd::seed [private]

Random generator seed.

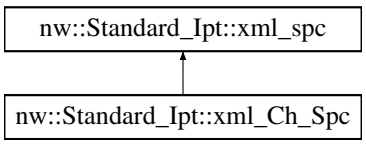
The documentation for this class was generated from the following files:

- Random/Uni_Rnd.h
- Random/Uni_Rnd.cpp

3.19 nw::Standard_Ipt::xml_Ch_Spc Struct Reference

Structure with attributes of the Channel_Spc class.

Inheritance diagram for nw::Standard_Ipt::xml_Ch_Spc:



Public Attributes

- long xml_n_subunits
- long xml_n_suToOpen

3.19.1 Detailed Description

Structure with attributes of the Channel_Spc class.

3.19.2 Member Data Documentation

3.19.2.1 long nw::Standard_Ipt::xml_Ch_Spc::xml_n_subunits

3.19.2.2 long nw::Standard_Ipt::xml_Ch_Spc::xml_n_suToOpen

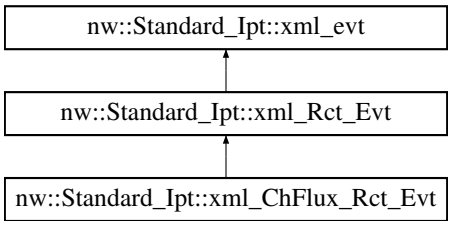
The documentation for this struct was generated from the following file:

- Input/Standard_Ipt.h

3.20 nw::Standard_Ipt::xml_ChFlux_Rct_Evt Struct Reference

Structure with attributes of the ChFlux_Rct_Evt class.

Inheritance diagram for nw::Standard_Ipt::xml_ChFlux_Rct_Evt:



Public Attributes

- long xml_channel_id

3.20.1 Detailed Description

Structure with attributes of the ChFlux_Rct_Evt class.

3.20.2 Member Data Documentation

3.20.2.1 long nw::Standard_Ipt::xml_ChFlux_Rct_Evt::xml_channel_id

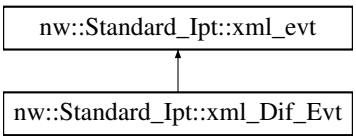
The documentation for this struct was generated from the following file:

- Input/Standard_Ipt.h

3.21 nw::Standard_Ipt::xml_Dif_Evt Struct Reference

Structure with attributes of the Diffusion_Evt class.

Inheritance diagram for nw::Standard_Ipt::xml_Dif_Evt:



Public Attributes

- long xml_diff_spc

3.21.1 Detailed Description

Structure with attributes of the Diffusion_Evt class.

3.21.2 Member Data Documentation

3.21.2.1 long nw::Standard_Ipt::xml_Dif_Evt::xml_diff_spc

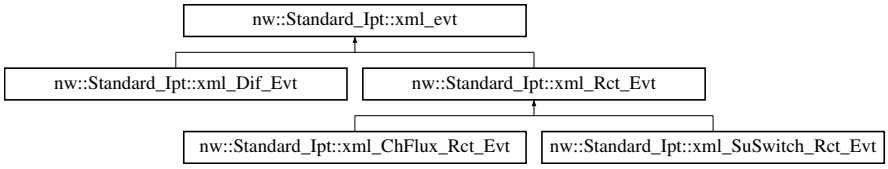
The documentation for this struct was generated from the following file:

- Input/Standard_Ipt.h

3.22 nw::Standard_Ipt::xml_evt Struct Reference

Structure with attributes of the _Event class.

Inheritance diagram for nw::Standard_Ipt::xml_evt:



Public Attributes

- long xml_id
- string xml_name
- double xml_k
- vector< long > xml_vxl

3.22.1 Detailed Description

Structure with attributes of the _Event class.

3.22.2 Member Data Documentation

3.22.2.1 long nw::Standard_Ipt::xml_evt::xml_id

3.22.2.2 double nw::Standard_Ipt::xml_evt::xml_k

3.22.2.3 string nw::Standard_Ipt::xml_evt::xml_name

3.22.2.4 vector<long> nw::Standard_Ipt::xml_evt::xml_vxl

The documentation for this struct was generated from the following file:

- Input/Standard_Ipt.h

3.23 nw::Standard_Ipt::xml_gen_data Struct Reference

Structure with attributes representing general data necessary for a system.

Public Attributes

- string xml_simulation_type
- string xml_border_condition
- long xml_output_mode
- double xml_max_sim_steps
- double xml_max_sim_time
- double xml_sample_rate
- double xml_box_lenght
- vector< long > xml_output_Spc
- vector< long > xml_output_Vxl
- vector< xml_Rct_Evt * > xml_rct_evt_list
- vector< xml_Dif_Evt * > xml_dif_evt_list
- vector< xml_SuSwitch_Rct_Evt * > xml_suSwitch_rct_evt_list
- vector< xml_ChFlux_Rct_Evt * > xml_chFlux_rct_evt_list
- vector< xml_vxl * > xml_vxl_list
- vector< xml_spc * > xml_spc_list
- vector< xml_Ch_Spc * > xml_ch_spc_list

3.23.1 Detailed Description

Structure with attributes representing general data necessary for a system.

3.23.2 Member Data Documentation

3.23.2.1 string nw::Standard_Ipt::xml_gen_data::xml_border_condition

3.23.2.2 double nw::Standard_Ipt::xml_gen_data::xml_box_lenght

3.23.2.3 vector<xml_Ch_Spc*> nw::Standard_Ipt::xml_gen_data::xml_ch_spc_list

3.23.2.4 vector<xml_ChFlux_Rct_Evt*> nw::Standard_Ipt::xml_gen_data::xml_chFlux_rct_evt_list

3.23.2.5 vector<xml_Dif_Evt*> nw::Standard_Ipt::xml_gen_data::xml_dif_evt_list

3.23.2.6 double nw::Standard_Ipt::xml_gen_data::xml_max_sim_steps

3.23.2.7 double nw::Standard_Ipt::xml_gen_data::xml_max_sim_time

3.23.2.8 long nw::Standard_Ipt::xml_gen_data::xml_output_mode

3.23.2.9 vector<long> nw::Standard_Ipt::xml_gen_data::xml_output_Spc

3.23.2.10 vector<long> nw::Standard_Ipt::xml_gen_data::xml_output_Vxl

3.23.2.11 vector<xml_Rct_Evt*> nw::Standard_Ipt::xml_gen_data::xml_rct_evt_list

3.23.2.12 double nw::Standard_Ipt::xml_gen_data::xml_sample_rate

3.23.2.13 string nw::Standard_Ipt::xml_gen_data::xml_simulation_type

3.23.2.14 vector<xml_spc*> nw::Standard_Ipt::xml_gen_data::xml_spc_list

3.23.2.15 vector<xml_SuSwitch_Rct_Evt*> nw::Standard_Ipt::xml_gen_data::xml_suSwitch_rct_evt_list

3.23.2.16 vector<xml_vxl*> nw::Standard_Ipt::xml_gen_data::xml_vxl_list

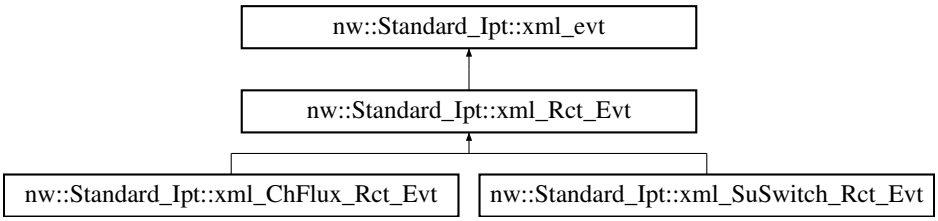
The documentation for this struct was generated from the following file:

- Input/Standard_Ipt.h

3.24 nw::Standard_Ipt::xml_Rct_Evt Struct Reference

Structure with attributes of the Reaction_Evt class.

Inheritance diagram for nw::Standard_Ipt::xml_Rct_Evt:



Public Attributes

- vector< long > xml_educts
- vector< long > xml_products

3.24.1 Detailed Description

Structure with attributes of the Reaction_Evt class.

3.24.2 Member Data Documentation

3.24.2.1 vector<long> nw::Standard_Ipt::xml_Rct_Evt::xml_educts

3.24.2.2 vector<long> nw::Standard_Ipt::xml_Rct_Evt::xml_products

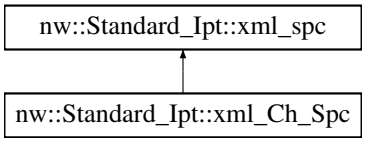
The documentation for this struct was generated from the following file:

- Input/Standard_Ipt.h

3.25 nw::Standard_Ipt::xml_spc Struct Reference

Structure with attributes of the _Species class.

Inheritance diagram for nw::Standard_Ipt::xml_spc:



Public Attributes

- long xml_id
- string xml_name
- double xml_init_conc

3.25.1 Detailed Description

Structure with attributes of the _Species class.

3.25.2 Member Data Documentation

3.25.2.1 long nw::Standard_Ipt::xml_spc::xml_id

3.25.2.2 double nw::Standard_Ipt::xml_spc::xml_init_conc

3.25.2.3 string nw::Standard_Ipt::xml_spc::xml_name

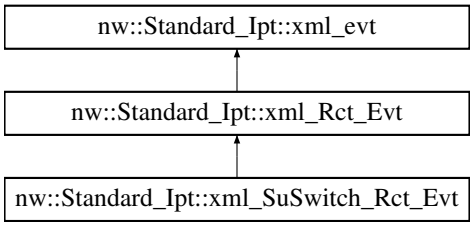
The documentation for this struct was generated from the following file:

- Input/Standard_Ipt.h

3.26 nw::Standard_Ipt::xml_SuSwitch_Rct_Evt Struct Reference

Structure with attributes of the SubUnitSwitch_Rct_Evt class.

Inheritance diagram for nw::Standard_Ipt::xml_SuSwitch_Rct_Evt:



Public Attributes

- long xml_actSu_id
- long xml_channel_id

3.26.1 Detailed Description

Structure with attributes of the SubUnitSwitch_Rct_Evt class.

3.26.2 Member Data Documentation

3.26.2.1 long nw::Standard_Ipt::xml_SuSwitch_Rct_Evt::xml_actSu_id

3.26.2.2 long nw::Standard_Ipt::xml_SuSwitch_Rct_Evt::xml_channel_id

The documentation for this struct was generated from the following file:

- Input/Standard_lpt.h

3.27 nw::Standard_lpt::xml_vxl Struct Reference

Structure with attributes of the _Voxel class.

Public Attributes

- long xml_id
- vector< long > xml_init_state
- vector< long > xml_vxl_neighbours

3.27.1 Detailed Description

Structure with attributes of the _Voxel class.

3.27.2 Member Data Documentation

3.27.2.1 long nw::Standard_lpt::xml_vxl::xml_id

3.27.2.2 vector<long> nw::Standard_lpt::xml_vxl::xml_init_state

3.27.2.3 vector<long> nw::Standard_lpt::xml_vxl::xml_vxl_neighbours

The documentation for this struct was generated from the following file:

- Input/Standard_lpt.h

Chapter 4

File Documentation

4.1 Events/_Event.h File Reference

```
#include <vector>
#include <string>
#include <cmath>
#include <iostream>
#include "../Voxel/_Voxel.h"
#include "../Random/Uni_Rnd.h"
```

Classes

- class nw::_Event

  Abstract base class for system state transitions (events).
- struct nw::_Event::tv_struct

  tau-voxel-structure (tv_struct) is a structure that associates a tau value with a _Voxel.

Namespaces

- nw

Typedefs

- typedef vector< _Voxel * > nw::VoxelVector

  typedefs of Vector related structures
- typedef VoxelVector::iterator nw::VoxelIterator

4.2 Events/ChFlux_Rct_Evt.cpp File Reference

```
#include "ChFlux_Rct_Evt.h"
#include <cmath>
```

Namespaces

- nw

4.3 Events/ChFlux_Rct_Evt.h File Reference

```
#include "Reaction_Evt.h"
```

Classes

- class nw::ChFlux_Rct_Evt

  Channel flux reaction event. This event is derived by the reaction event and introduces the flux of a molecular species through a channel. It differs from an ordinary reaction event due to the fact that the educt (the channel) is not modified by the event itself. Thus the educt vector has to be generated manually in the init() function.

Namespaces

- nw

4.4 Events/Diffusion_Evt.cpp File Reference

```
#include <math.h>
#include <iostream>
#include <exception>
#include "Diffusion_Evt.h"
```

Namespaces

- nw

4.5 Events/Diffusion_Evt.h File Reference

```
#include <vector>
#include <string>
#include "../Voxel/_Voxel.h"
#include "_Event.h"
```

Classes

- class nw::Diffusion_Evt

  The diffusion event realizes the diffusion between two voxel.

Namespaces

- nw

4.6 Events/Reaction_Evt.cpp File Reference

```
#include <math.h>
#include <cmath>
#include <algorithm>
#include <iostream>
#include <exception>
#include "../Voxel/Border_Vxl.h"
#include "Reaction_Evt.h"
```

Namespaces

- nw

4.7 Events/Reaction_Evt.h File Reference

```
#include "_Event.h"
#include <vector>
#include <string>
```

Classes

- class nw::Reaction_Evt

  The Reaction_Event realizes the Reaction of one ore two molecules.

Namespaces

- nw

4.8 Events/SubUnitSwitch_Rct_Evt.cpp File Reference

```
#include "SubUnitSwitch_Rct_Evt.h"
#include <math.h>
#include <iostream>
```

4.9 Events/SubUnitSwitch_Rct_Evt.h File Reference

```
#include "Reaction_Evt.h"
#include "../Species/Channel_Spc.h"
```

Classes

- class nw::SubUnitSwitch_Rct_Evt

Namespaces

- nw

4.10 Input/_Input.h File Reference

```
#include "../System/_System.h"
```

Classes

- class nw::_Input

  Abstract base class for input procedures.

Namespaces

- nw

4.11 Input/Standard_Ipt.cpp File Reference

```
#include "Standard_Ipt.h"
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <time.h>
```

Namespaces

- nw

4.12 Input/Standard_Ipt.h File Reference

```
#include "_Input.h"
#include "../System/Gillespie_Sys.h"
#include "../EventHeader.h"
#include "../VoxelHeader.h"
#include "../SpeciesHeader.h"
#include "../pugi/pugixml.hpp"
```

Classes

- class nw::Standard_Ipt

  Input class that parses a .xml files and generates a Gillespie_Sys.
- struct nw::Standard_Ipt::xml_evt

  Structure with attributes of the _Event class.
- struct nw::Standard_Ipt::xml_Rct_Evt

  Structure with attributes of the Reaction_Evt class.

- struct nw::Standard_Ipt::xml_Dif_Evt

  Structure with attributes of the Diffusion_Evt class.
- struct nw::Standard_Ipt::xml_SuSwitch_Rct_Evt

  Structure with attributes of the SubUnitSwitch_Rct_Evt class.
- struct nw::Standard_Ipt::xml_ChFlux_Rct_Evt

  Structure with attributes of the ChFlux_Rct_Evt class.
- struct nw::Standard_Ipt::xml_vxl

  Structure with attributes of the _Voxel class.
- struct nw::Standard_Ipt::xml_spc

  Structure with attributes of the _Species class.
- struct nw::Standard_Ipt::xml_Ch_Spc

  Structure with attributes of the Channel_Spc class.
- struct nw::Standard_Ipt::xml_gen_data

  Structure with attributes representing general data necessary for a system.

Namespaces

- nw

4.13 mainpage.dox File Reference

4.14 Random/_Random.h File Reference

Classes

- class nw::_Random

  abstract base class to define an interface for Random Generators

Namespaces

- nw

4.15 Random/Uni_Rnd.cpp File Reference

```
#include "Uni_Rnd.h"
#include <iostream>
```

Macros

- #define IA 16807
- #define IM 2147483647
- #define AM (1.0/IM)
- #define IQ 127773
- #define IR 2836
- #define NTAB 32
- #define NDIV (1+(IM-1)/NTAB)
- #define EPS 1.2e-7
- #define RNMX (1.0-EPS)

4.15.1 Macro Definition Documentation

4.15.1.1 #define AM (1.0/IM)

4.15.1.2 #define EPS 1.2e-7

4.15.1.3 #define IA 16807

4.15.1.4 #define IM 2147483647

4.15.1.5 #define IQ 127773

4.15.1.6 #define IR 2836

4.15.1.7 #define NDIV (1+(IM-1)/NTAB)

4.15.1.8 #define NTAB 32

4.15.1.9 #define RNMX (1.0-EPS)

4.16 Random/Uni_Rnd.h File Reference

```
#include "_Random.h"
#include <time.h>
```

Classes

- class nw::Uni_Rnd

  "Minimal" random number generator of Park and Miller

Namespaces

- nw

4.17 Species/_Species.h File Reference

```
#include <string>
```

Classes

- class nw::_Species

  Abstract base class for the molecular species of a system.

Namespaces

- nw

Variables

- static const double nw::N_AVO = 6.022e23

4.18 Species/Channel_Spc.cpp File Reference

```
#include "Channel_Spc.h"
#include <iostream>
```

Namespaces

- nw

4.19 Species/Channel_Spc.h File Reference

```
#include "_Species.h"
#include <iostream>
#include <vector>
```

Classes

- class nw::Channel_Spc

  Channel Species. This class implements a channel species. This class realizes channel proteins build of a certain number of subunits which open if a defined number of those subunits is in an open state.

Namespaces

- nw

4.20 Species/Standard_Spc.h File Reference

```
#include <iostream>
#include "_Species.h"
```

Classes

- class nw::Standard_Spc

  A Standard_Spc is an object that holds the properties of a special kind of molecules.

Namespaces

- nw

4.21 System/_System.h File Reference

Classes

- class nw::_System

  Abstract base class for the implementation of simulation algorithms.

Namespaces

- nw

4.22 System/Gillespie_Sys.cpp File Reference

```
#include <iostream>
#include <algorithm>
#include <exception>
#include <cmath>
#include <ios>
#include <iomanip>
#include "Gillespie_Sys.h"
```

Namespaces

- nw

4.23 System/Gillespie_Sys.h File Reference

```
#include <vector>
#include <fstream>
#include <sstream>
#include "_System.h"
#include "../Events/_Event.h"
#include "../Voxel/_Voxel.h"
```

Classes

- class nw::Gillespie_Sys

  The Gillespie_Sys coordinates Gillespie's SSA and the whole output procedure. The whole logic of the Simulation Software is combined at this point. All Events and all Voxel are represented and important functions like build_dependency_graph are located here.
  Important fact is, that Gillespie_Sys just uses the Interfaces _Voxel and _Event what makes it really easy to extend the Simulation.

Namespaces

- nw

Typedefs

- typedef vector< _Event * > nw::EventVector
  typedefs of Event related structures
- typedef EventVector::iterator nw::EventIterator

Variables

- static const string nw::NEGATIVETAUMSG = "NEGATIVE TAU ERROR LAST EVENT "

4.24 Voxel/_Voxel.h File Reference

```
#include "../Species/_Species.h"
#include <iostream>
#include <math.h>
```

Classes

- class nw::_Voxel
  Abstract base class for all different types of voxel.

Namespaces

- nw

Typedefs

- typedef vector< _Species * > nw::SpeciesVector
  typedefs of Species related structures
- typedef SpeciesVector::iterator nw::SpeciesIterator

4.25 Voxel/Border_Vxl.cpp File Reference

```
#include "Border_Vxl.h"
#include <iostream>
#include <math.h>
```

Namespaces

- nw

4.26 Voxel/Border_Vxl.h File Reference

```
#include <vector>
#include "_Voxel.h"
#include "../Species/_Species.h"
```

Classes

- class nw::Border_Vxl

  The border Voxel defines the border condition of a whole voxel-system.

Namespaces

- nw

4.27 Voxel/Standard_Vxl.cpp File Reference

```
#include "Standard_Vxl.h"
#include <iostream>
#include <math.h>
```

Namespaces

- nw

4.28 Voxel/Standard_Vxl.h File Reference

```
#include <vector>
#include "_Voxel.h"
#include "../Species/_Species.h"
```

Classes

- class nw::Standard_Vxl

  Standard voxel implementation.

Namespaces

- nw