Reference Manual

Generated by Doxygen 1.6.2-20100208

Wed Jun 29 12:27:58 2011

Contents

Chapter 1

Class Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

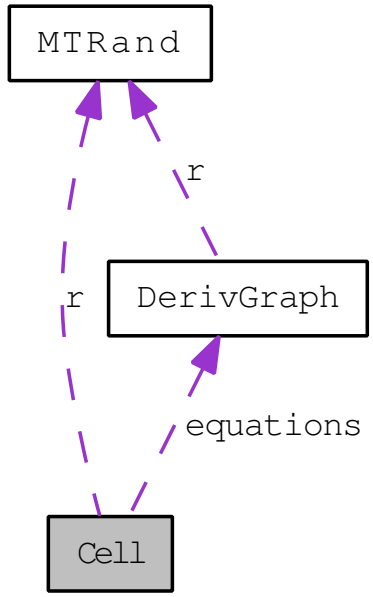
Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Cell Class Reference

`#include <Cell.h>`Collaboration diagram for Cell:



Public Member Functions

- Cell ()
- ~Cell ()
- int mutate ()
- void outputDotImage ()
- void outputDataPlot ()

4.1.1 Detailed Description

Cell header file.

4.1.2 Constructor & Destructor Documentation

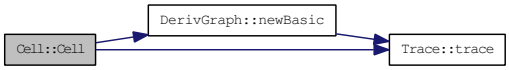
4.1.2.1 Cell::Cell ()

Cell::Cell()

Cell default constructor.

Allocates: 1 derivGraph object

Here is the call graph for this function:



4.1.2.2 Cell::~Cell ()

Cell::~Cell()

Cell default destructor.

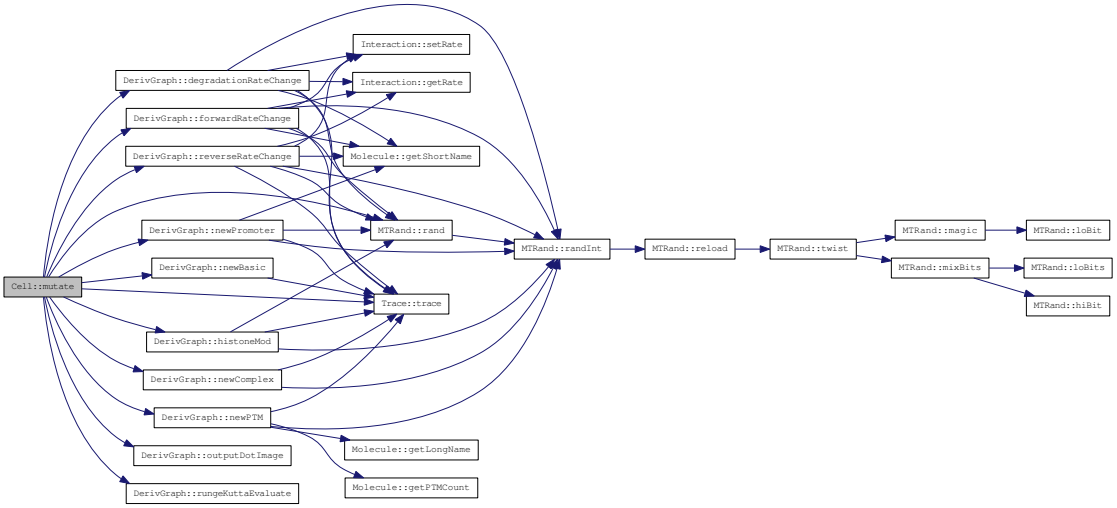
Frees: 1 derivGraph object

Here is the call graph for this function:

4.1.3 Member Function Documentation

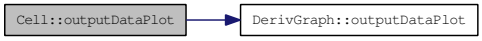
4.1.3.1 int Cell::mutate ()

Here is the call graph for this function:



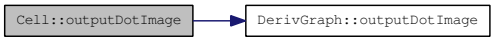
4.1.3.2 void Cell::outputDataPlot ()

Here is the call graph for this function:



4.1.3.3 void Cell::outputDotImage ()

Here is the call graph for this function:



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

- Cell.h
- Cell.cpp

4.2 cmp_str Struct Reference

`#include <Trace.h>`

Public Member Functions

- bool operator() (const char *a, const char *b)

4.2.1 Member Function Documentation

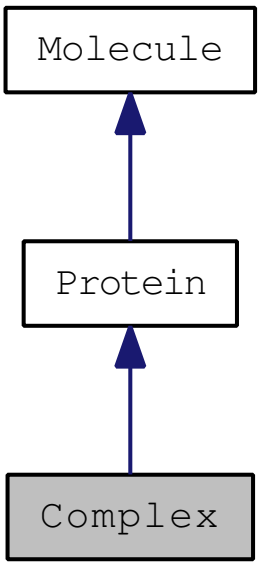
4.2.1.1 bool cmp_str::operator() (const char * *a*, const char * *b*) [inline]

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

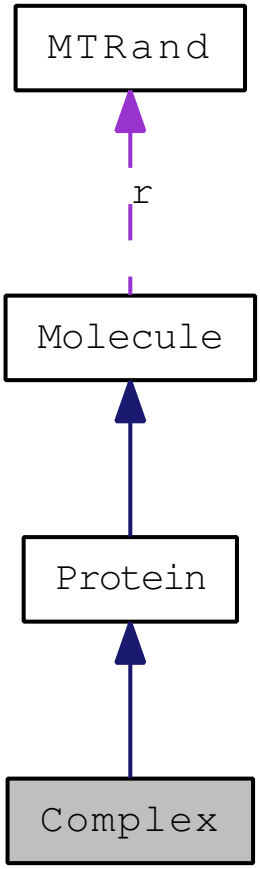
- Trace.h

4.3 Complex Class Reference

`#include <CustomMolecules.h>`Inheritance diagram for Complex:



Collaboration diagram for Complex:



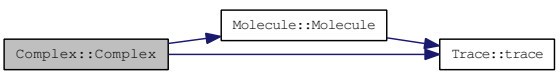
Public Member Functions

- Complex (int, int)
- ~Complex ()
- int getComponentId (int)

4.3.1 Constructor & Destructor Documentation

4.3.1.1 Complex::Complex (int *n1*, int *n2*)

Here is the call graph for this function:



4.3.1.2 Complex::∼Complex ()

4.3.2 Member Function Documentation

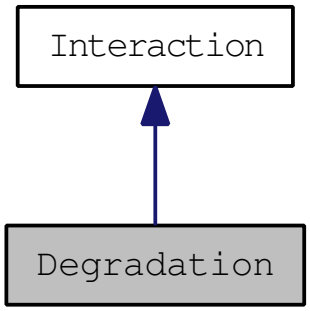
4.3.2.1 int Complex::getComponentId (int *i*)

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

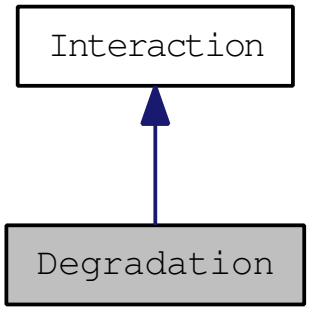
- CustomMolecules.h
- CustomMolecules.cpp

4.4 Degradation Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for Degradation:



Collaboration diagram for Degradation:



Public Member Functions

- Degradation ()
- ~Degradation ()
- virtual float getEffect (ListDigraph *, ListDigraph::NodeMap< Molecule * > *, ListDigraph::ArcMap< Interaction * > *, ListDigraph::Node, int, float)

4.4.1 Constructor & Destructor Documentation

4.4.1.1 Degradation::Degradation ()

4.4.1.2 Degradation::~Degradation ()

4.4.2 Member Function Documentation

4.4.2.1 float Degradation::getEffect (ListDigraph * *g*, ListDigraph::NodeMap< Molecule * > * *m*, ListDigraph::ArcMap< Interaction * > * *i*, ListDigraph::Node *a*, int *rkIter*, float *rkStep*) [virtual]

float Interaction::getEffect(ListDigraph* , NodeMap<Molecule*>* , ArcMap<Interaction*>* , Node , int, float)

Get the effect this interaction has on a particular node.

This method defines the behavior of an interaction which connects two molecules. The effect on Node a can be dependent on any other molecule, which can be accessed using the ListDigraph, NodeMap, and ArcMap parameters.

Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary to call Molecule::rkApprox(stepsize, iteration) rather than Molecule::getValue() to get the current Iteration's approximated concentration.

Parameters:

g The graph object containing Node-Node relationships.

m The NodeMap object containing Node-Molecule mappings.

i The ArcMap object containing Arc-Interaction mappings.

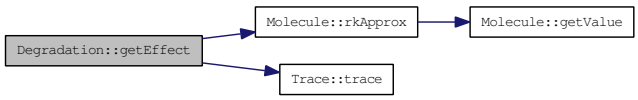
a The Node to calculate the effect for

rkIter The current iteration of Runge-Kutta [0,3]

rkStep The stepsize of Runge-Kutta

Reimplemented from Interaction.

Here is the call graph for this function:

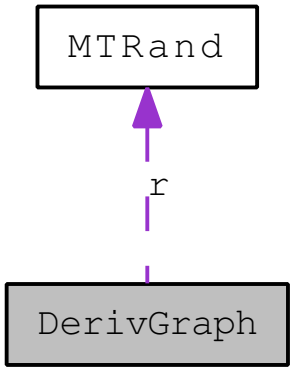


The documentation for this class was generated from the following files:

- CustomInteractions.h
- CustomInteractions.cpp

4.5 DerivGraph Class Reference

`#include <DerivGraph.h>`Collaboration diagram for DerivGraph:



Public Member Functions

- DerivGraph ()
- ~DerivGraph ()
- void test ()
- void rungeKuttaEvaluate (float)
- void outputDotImage (int, int)
- void outputDataPlot (int, int, float)
- ListDigraph * getListDigraph ()
- ListDigraph::NodeMap< Molecule * > * getNodeMap ()
- ListDigraph::ArcMap< Interaction * > * getArcMap ()
- void newBasic ()
- void forwardRateChange ()
- void reverseRateChange ()
- void degradationRateChange ()
- DNA * histoneMod ()
- void newComplex ()
- void newPromoter ()
- void newPTM ()

Public Attributes

- MTRand r

4.5.1 Constructor & Destructor Documentation

4.5.1.1 DerivGraph::DerivGraph ()

DerivGraph::DerivGraph()

DerivGraph constructor.

The DerivGraph holds LEMON objects such as ListDigraph, NodeMap, and ArcMap. It also holds the data produced by Runge-Kutta and facilitates plotting using Gnuplot.

The derivatives describing the concentration of molecules in the cell can be represented as a directed graph.

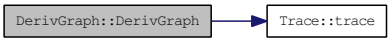
Each Node represents a type of molecule in the cell, and each Arc represents an interaction which has an effect on the Nodes which it connects.

Allocates: 1 ListDigraph() object 1 ListDigraph::NodeMap objects 1 ListDigraph::ArcMap object

Test code / newBasic(); newBasic(); newBasic(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); newPTM(); test();

rungeKuttaEvaluate(.5);

Here is the call graph for this function:



4.5.1.2 DerivGraph::~DerivGraph ()

DerivGraph::~DerivGraph()

DerivGraph Destructor.

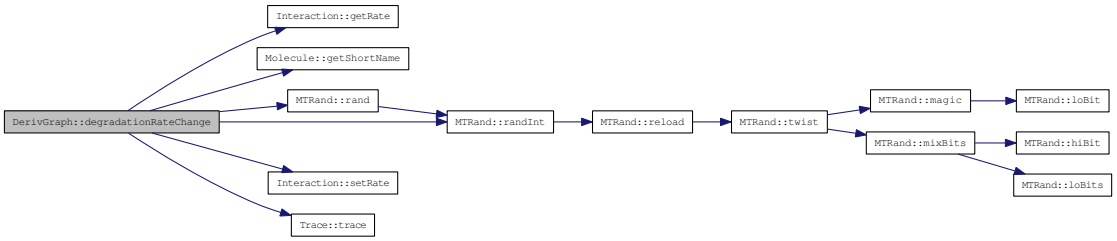
Frees: 1 NodeMap object n contained Molecule objects 1 ArcMap object m contained Interaction objects 1 ListDigraph object

Here is the call graph for this function:

4.5.2 Member Function Documentation

4.5.2.1 void DerivGraph::degradationRateChange ()

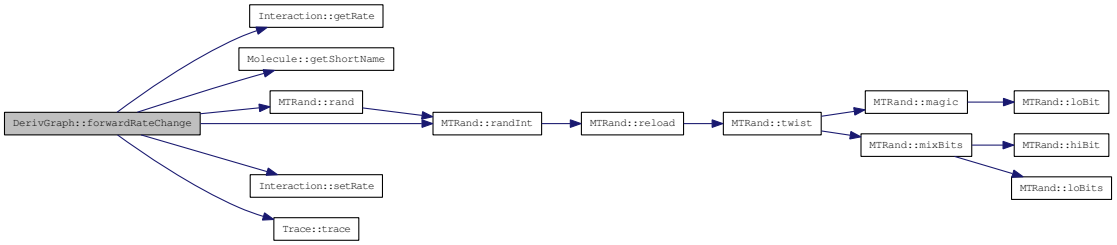
Here is the call graph for this function:



4.5.2.2 void DerivGraph::forwardRateChange ()

Randomly select a forward interaction and modify its rate.

Here is the call graph for this function:



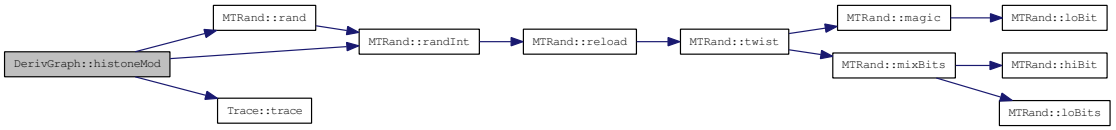
4.5.2.3 ListDigraph::ArcMap<Interaction*>* DerivGraph::getArcMap ()

4.5.2.4 ListDigraph* DerivGraph::getListDigraph ()

4.5.2.5 ListDigraph::NodeMap<Molecule*>* DerivGraph::getNodeMap ()

4.5.2.6 DNA * DerivGraph::histoneMod ()

Here is the call graph for this function:



4.5.2.7 void DerivGraph::newBasic ()

DerivGraph::newBasic()

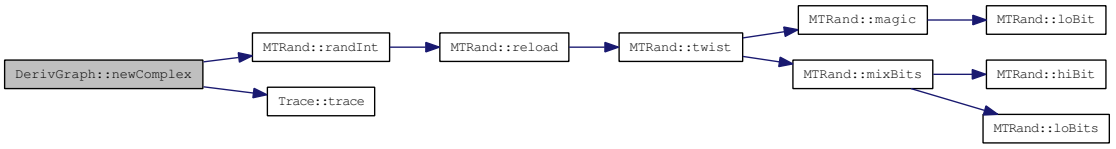
Create a new DNA, mRNA, and protein in the cell.

DNA ---> mRNA ----> Protein | | v v Deg Deg

Here is the call graph for this function:

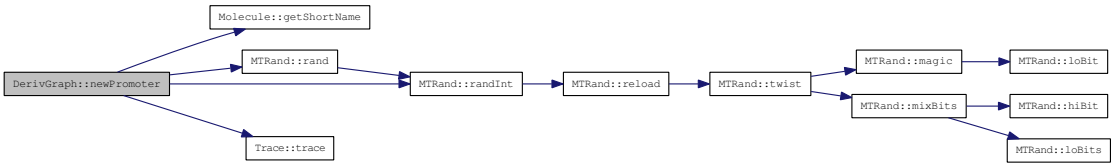
4.5.2.8 void DerivGraph::newComplex ()

Here is the call graph for this function:



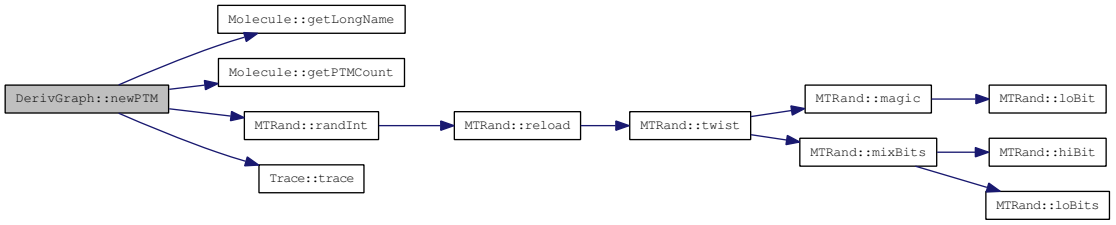
4.5.2.9 void DerivGraph::newPromoter ()

Here is the call graph for this function:



4.5.2.10 void DerivGraph::newPTM ()

Here is the call graph for this function:



4.5.2.11 void DerivGraph::outputDataPlot (int *cellNum*, int *gen*, float *step*)

4.5.2.12 void DerivGraph::outputDotImage (int *cellNum*, int *gen*)

void DerivGraph::outputDotImage(int, int)

Output a png image of the current graph structure using GraphViz.

A process running GraphViz is forked and a pipe opened to its standard in. The general layout of the output file can be changed below. The Node and Arc names are defined within the Molecule and Interaction classes.

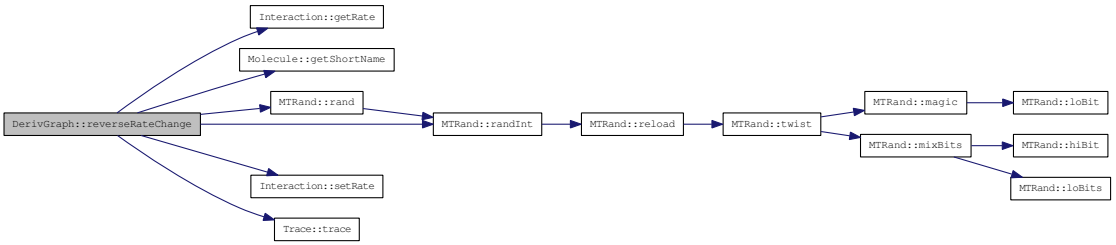
Parameters:

cellNum the cell number to put in the filename

gen the generation number to put in the filename

4.5.2.13 void DerivGraph::reverseRateChange ()

Here is the call graph for this function:



4.5.2.14 void DerivGraph::rungeKuttaEvaluate (float *rkStep*)

Uses the Runge-Kutta fourth order method to approximate the solutions to the system of differential equations

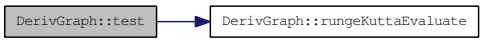
The result of this algorithm is the vector rungeKuttaSolution within each Molecule object containing the approximation of the concentration at each timestep.

Parameters:

rkStep the timestep (precision) between calculated points

4.5.2.15 void DerivGraph::test ()

Here is the call graph for this function:



4.5.3 Member Data Documentation

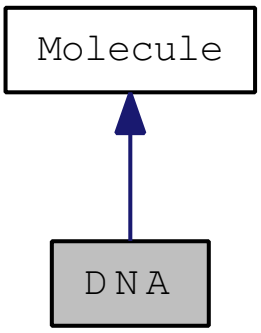
4.5.3.1 MTRand DerivGraph::r

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

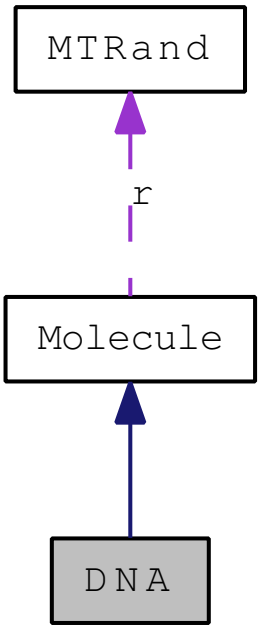
- DerivGraph.h
- DerivGraph.cpp

4.6 DNA Class Reference

`#include <CustomMolecules.h>`Inheritance diagram for DNA:



Collaboration diagram for DNA:



Public Member Functions

- DNA ()
- ~DNA ()
- float getValue ()
- float rkApprox (int, float)
- void setHistoneModValue (float)

Public Attributes

- int promoterId
- int hill

4.6.1 Constructor & Destructor Documentation

4.6.1.1 DNA::DNA ()

CustomMolecules implementation file.

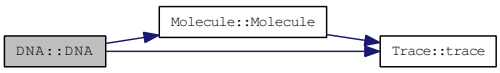
Custom Molecules allow modification of the default behavior of molecules.

Each molecule must be defined in the CustomMolecules.h header file. DNA::DNA()

Default Constructor

Derived from Molecule.

Here is the call graph for this function:



4.6.1.2 DNA::~DNA ()

DNA::~DNA()

Default Destructor

4.6.2 Member Function Documentation

4.6.2.1 float DNA::getValue () `[virtual]`

Overload of virtual method Molecule::getValue

Returns:

Goodwin term describing the probability that the DNA is available for transcription.

Reimplemented from Molecule.

4.6.2.2 float DNA::rkApprox (int *rkIteration*, float *rkStepSize*) `[virtual]`

float Molecule::rkApprox(int, float)

Reimplemented from Molecule.

Here is the call graph for this function:

4.6.2.3 void DNA::setHistoneModValue (float *newVal*)

4.6.3 Member Data Documentation

4.6.3.1 int DNA::hill

4.6.3.2 int DNA::promoterId

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

- CustomMolecules.h
- CustomMolecules.cpp

4.7 Experiment Class Reference

`#include <Experiment.h>`

Public Member Functions

- Experiment (int, int)
- ~Experiment ()
- void start ()

Friends

- class ExperimentTests

4.7.1 Constructor & Destructor Documentation

4.7.1.1 Experiment::Experiment (int *ncells*, int *generations*)

Experiment::Experiment(int, int)

Experiment constructor.

Parameters:

number of Cell objects to be created.

number of Generations the Experiment will run for.

Here is the call graph for this function:

4.7.1.2 Experiment::~Experiment ()

Experiment::~Experiment(int, int)

Experiment destructor.

Deletes the Cell objects from the cells vector, then deletes the vector itself.

Here is the call graph for this function:

4.7.2 Member Function Documentation

4.7.2.1 void Experiment::start ()

Deprecated

Here is the call graph for this function:

4.7.3 Friends And Related Function Documentation

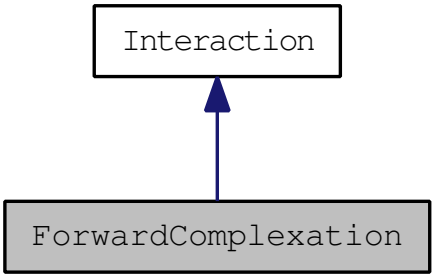
4.7.3.1 friend class ExperimentTests [friend]

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

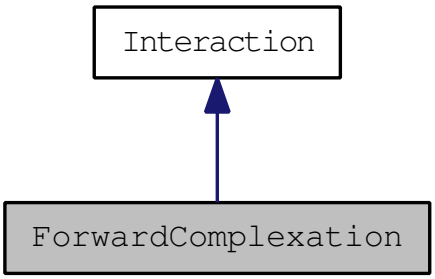
- Experiment.h
- Experiment.cpp

4.8 ForwardComplexation Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for ForwardComplexation:



Collaboration diagram for ForwardComplexation:



Public Member Functions

- ForwardComplexation (int, int)
- ~ForwardComplexation ()
- virtual float getEffect (ListDigraph *, ListDigraph::NodeMap< Molecule * > *, ListDigraph::ArcMap< Interaction * > *, ListDigraph::Node, int, float)

Public Attributes

- int firstNodeID
- int secondNodeID

4.8.1 Constructor & Destructor Documentation

4.8.1.1 ForwardComplexation::ForwardComplexation (int *n1*, int *n2*)

Here is the call graph for this function:

4.8.1.2 ForwardComplexation::~ForwardComplexation ()

4.8.2 Member Function Documentation

4.8.2.1 float ForwardComplexation::getEffect (ListDigraph * *g*, ListDigraph::NodeMap< Molecule * > * *m*, ListDigraph::ArcMap< Interaction * > * *i*, ListDigraph::Node *a*, int *rkIter*, float *rkStep*) [virtual]

float Interaction::getEffect(ListDigraph* , NodeMap<Molecule*>* , ArcMap<Interaction*>* , Node , int, float)

Get the effect this interaction has on a particular node.

This method defines the behavior of an interaction which connects two molecules. The effect on Node a can be dependent on any other molecule, which can be accessed using the ListDigraph, NodeMap, and ArcMap parameters.

Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary to call Molecule::rkApprox(stepsize, iteration) rather than Molecule::getValue() to get the current Iteration's approximated concentration.

Parameters:

g The graph object containing Node-Node relationships.

m The NodeMap object containing Node-Molecule mappings.

i The ArcMap object containing Arc-Interaction mappings.

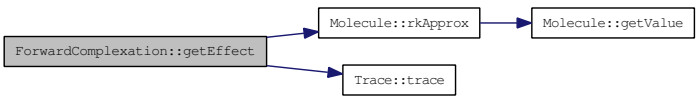
a The Node to calculate the effect for

rkIter The current iteration of Runge-Kutta [0,3]

rkStep The stepsize of Runge-Kutta

Reimplemented from Interaction.

Here is the call graph for this function:



4.8.3 Member Data Documentation

4.8.3.1 int ForwardComplexation::firstNodeID

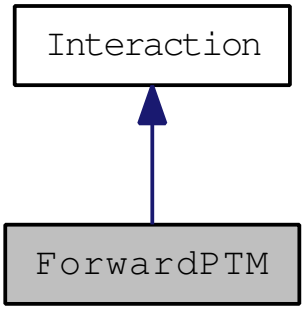
4.8.3.2 int ForwardComplexation::secondNodeID

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

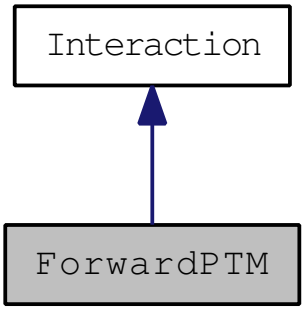
- CustomInteractions.h
- CustomInteractions.cpp

4.9 ForwardPTM Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for ForwardPTM:



Collaboration diagram for ForwardPTM:



Public Member Functions

- ForwardPTM ()
- ~ForwardPTM ()

4.9.1 Constructor & Destructor Documentation

4.9.1.1 ForwardPTM::ForwardPTM ()

Here is the call graph for this function:

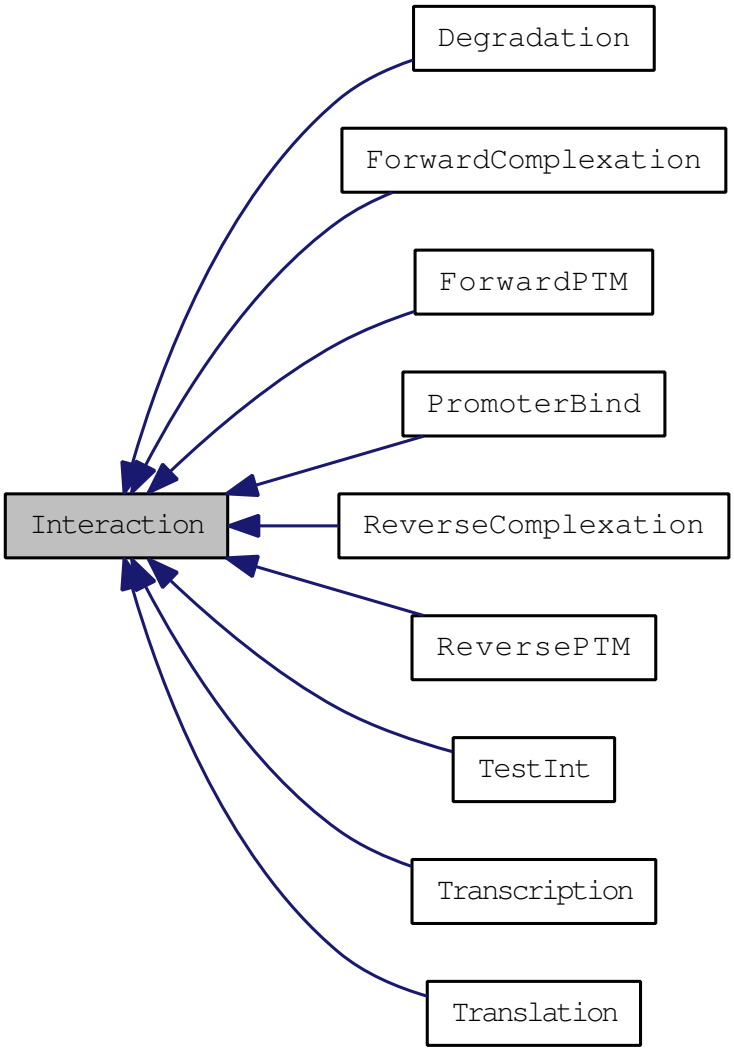
4.9.1.2 ForwardPTM::~ForwardPTM ()

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

- CustomInteractions.h
- CustomInteractions.cpp

4.10 Interaction Class Reference

`#include <Interaction.h>`Inheritance diagram for Interaction:



Public Member Functions

- Interaction ()
- ~Interaction ()
- virtual float getEffect (ListDigraph *, ListDigraph::NodeMap< Molecule * > *, ListDigraph::ArcMap< Interaction * > *, ListDigraph::Node, int, float)
- const char * getName ()
- float setRate (float)
- virtual float getRate ()

Public Attributes

- const char * name

- int arcID

Protected Attributes

- float rate

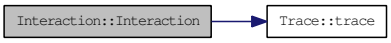
4.10.1 Constructor & Destructor Documentation

4.10.1.1 Interaction::Interaction ()

Interaction base class implementation Interaction::Interaction()

Interaction default constructor.

Here is the call graph for this function:



4.10.1.2 Interaction::~Interaction ()

Interaction::~Interaction()

Interaction default destructor.

Here is the call graph for this function:

4.10.2 Member Function Documentation

4.10.2.1 float Interaction::getEffect (ListDigraph * *g*, ListDigraph::NodeMap< Molecule * > * *m*, ListDigraph::ArcMap< Interaction * > * *i*, ListDigraph::Node *a*, int *rkIter*, float *rkStep*) [virtual]

float Interaction::getEffect(ListDigraph* , NodeMap<Molecule*>* , ArcMap<Interaction*>* , Node , int, float)

Get the effect this interaction has on a particular node.

This method defines the behavior of an interaction which connects two molecules. The effect on Node a can be dependent on any other molecule, which can be accessed using the ListDigraph, NodeMap, and ArcMap parameters.

Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary to call Molecule::rkApprox(stepsize, iteration) rather than Molecule::getValue() to get the current Iteration's approximated concentration.

Parameters:

g The graph object containing Node-Node relationships.

m The NodeMap object containing Node-Molecule mappings.

i The ArcMap object containing Arc-Interaction mappings.

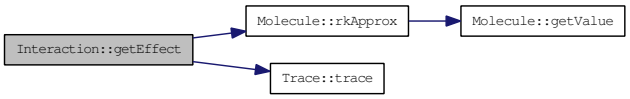
a The Node to calculate the effect for

rkIter The current iteration of Runge-Kutta [0,3]

rkStep The stepsize of Runge-Kutta

Reimplemented in TestInt, Transcription, Degradation, ForwardComplexation, ReverseComplexation, and PromoterBind.

Here is the call graph for this function:



4.10.2.2 const char * Interaction::getName ()

4.10.2.3 float Interaction::getRate () [virtual]

4.10.2.4 float Interaction::setRate (float *f*)

void Interaction::setRate(float)

Change the kinetic rate of the Interaction

Parameters:

f the new rate for the interaction

Returns:

the old rate for the interaction

4.10.3 Member Data Documentation

4.10.3.1 int Interaction::arcID

4.10.3.2 const char* Interaction::name

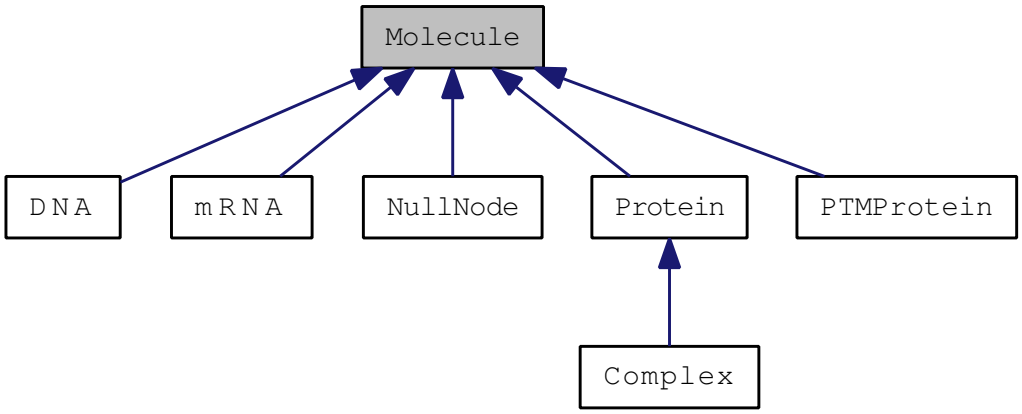
4.10.3.3 float Interaction::rate [protected]

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

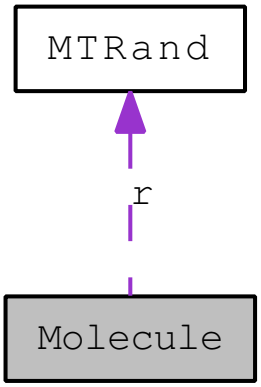
- Interaction.h
- Interaction.cpp

4.11 Molecule Class Reference

`#include <Molecule.h>`Inheritance diagram for Molecule:



Collaboration diagram for Molecule:



Public Member Functions

- Molecule ()
- virtual ~Molecule ()
- virtual float getValue ()
- void updateRkVal (int, float)
- void nextPoint (float)
- void setValue (float)
- void outputRK ()
- float getrkVal (int)
- vector< float > * getRungeKuttaSolution ()
- virtual float rkApprox (int, float)
- virtual char * getShortName ()
- virtual char * getLongName ()
- void setID (int)

- int getID ()
- void reset ()
- int getScore ()
- int getPTMCount (int, int)
- virtual int getPTMCount (int)

Public Attributes

- int nodeID
- int wasPTM
- int PTMArray [4]
- MTRand r

Protected Attributes

- char buf [200]
- const char * longName
- const char * shortName
- int moleculeID
- vector< float > rungeKuttaSolution

4.11.1 Constructor & Destructor Documentation

4.11.1.1 Molecule::Molecule ()

Molecule::Molecule()

Default Constructor

Here is the call graph for this function:

4.11.1.2 Molecule::~Molecule () [virtual]

Molecule::~Molecule()

Default Destructor

4.11.2 Member Function Documentation

4.11.2.1 int Molecule::getID ()

4.11.2.2 char * Molecule::getLongName () [virtual]

Reimplemented in PTMProtein.

4.11.2.3 int Molecule::getPTMCount (int *index*) [virtual]

4.11.2.4 int Molecule::getPTMCount (int, int)

4.11.2.5 float Molecule::getrkVal (int *k*)

float Molecule::getrkVal(int)

Get the value of the intermediate Runge-Kutta value for a particular iteration.

Parameters:

k Which iterations rkVal to return

Returns:

The value of this molecules rkVal[k]

4.11.2.6 vector< float > * Molecule::getRungeKuttaSolution ()

4.11.2.7 int Molecule::getScore ()

4.11.2.8 char * Molecule::getShortName () [virtual]

4.11.2.9 float Molecule::getValue () [virtual]

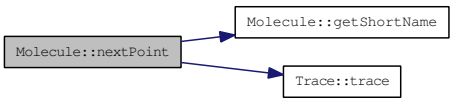
float Molecule::getValue()

Reimplemented in DNA, and NullNode.

4.11.2.10 void Molecule::nextPoint (float *step*)

void Molecule::nextPoint(float)

Here is the call graph for this function:

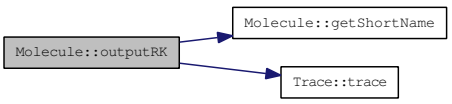


4.11.2.11 void Molecule::outputRK ()

void Molecule::outputRK()

TEST METHOD

Here is the call graph for this function:



4.11.2.12 void Molecule::reset ()

4.11.2.13 float Molecule::rkApprox (int *rkIteration*, float *rkStepSize*) [virtual]

float Molecule::rkApprox(int, float)

Reimplemented in DNA.

Here is the call graph for this function:

4.11.2.14 void Molecule::setID (int *i*)

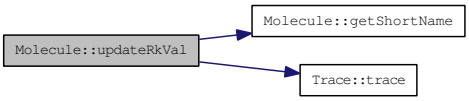
4.11.2.15 void Molecule::setValue (float *v*)

void Molecule::setValue(float)

4.11.2.16 void Molecule::updateRkVal (int *index*, float *amount*)

void Molecule::updateRkVal(int, float)

Here is the call graph for this function:



4.11.3 Member Data Documentation

4.11.3.1 char Molecule::buf[200] [protected]

4.11.3.2 const char* Molecule::longName [protected]

4.11.3.3 int Molecule::moleculeID [protected]

4.11.3.4 int Molecule::nodeID

4.11.3.5 int Molecule::PTMArray[4]

4.11.3.6 MTRand Molecule::r

4.11.3.7 vector<float> Molecule::rungeKuttaSolution [protected]

4.11.3.8 const char* Molecule::shortName [protected]

4.11.3.9 int Molecule::wasPTM

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

- Molecule.h
- Molecule.cpp

4.12 MoleculeType Class Reference

`#include <MoleculeType.h>`

Public Member Functions

- MoleculeType ()
- ~MoleculeType ()

4.12.1 Detailed Description

MoleculeType.h

MoleculeType holds default information about a type of molecule.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 MoleculeType::MoleculeType ()

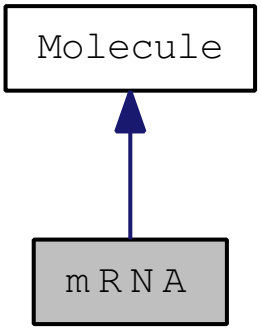
4.12.2.2 MoleculeType::~MoleculeType ()

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

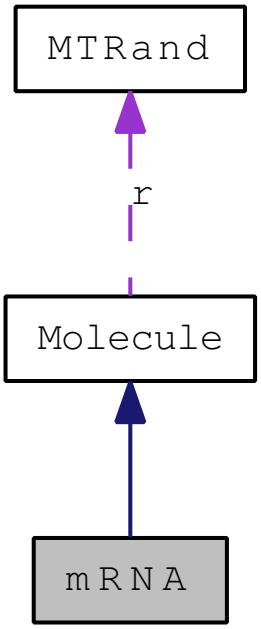
- MoleculeType.h

4.13 mRNA Class Reference

`#include <CustomMolecules.h>`Inheritance diagram for mRNA:



Collaboration diagram for mRNA:



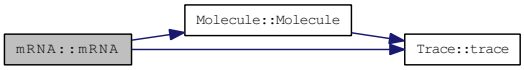
Public Member Functions

- mRNA ()
- ~mRNA ()

4.13.1 Constructor & Destructor Documentation

4.13.1.1 mRNA::mRNA ()

Here is the call graph for this function:



4.13.1.2 mRNA::~mRNA ()

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

- CustomMolecules.h
- CustomMolecules.cpp

4.14 MTRand Class Reference

`#include <MersenneTwister.h>`

Public Types

- enum { N = 624 }
- enum { SAVE = N + 1 }
- typedef unsigned long uint32

Public Member Functions

- MTRand (const uint32 oneSeed)
- MTRand (uint32 *const bigSeed, uint32 const seedLength=N)
- MTRand ()
- MTRand (const MTRand &o)
- uint32 randInt ()
- uint32 randInt (const uint32 n)
- double rand ()
- double rand (const double n)
- double randExc ()
- double randExc (const double n)
- double randDblExc ()
- double randDblExc (const double n)
- double operator() ()
- double rand53 ()
- double randNorm (const double mean=0.0, const double stddev=1.0)
- void seed (const uint32 oneSeed)
- void seed (uint32 *const bigSeed, const uint32 seedLength=N)
- void seed ()
- void save (uint32 *saveArray) const
- void load (uint32 *const loadArray)
- MTRand & operator= (const MTRand &o)

Protected Types

- enum { M = 397 }

Protected Member Functions

- void initialize (const uint32 oneSeed)
- void reload ()
- uint32 hiBit (const uint32 u) const
- uint32 loBit (const uint32 u) const
- uint32 loBits (const uint32 u) const
- uint32 mixBits (const uint32 u, const uint32 v) const
- uint32 magic (const uint32 u) const
- uint32 twist (const uint32 m, const uint32 s0, const uint32 s1) const

Static Protected Member Functions

- static uint32 hash (time_t t, clock_t c)

Protected Attributes

- uint32 state [N]
- uint32 * pNext
- int left

Friends

- std::ostream & operator<< (std::ostream &os, const MTRand &mtrand)
- std::istream & operator>> (std::istream &is, MTRand &mtrand)

4.14.1 Member Typedef Documentation

4.14.1.1 typedef unsigned long MTRand::uint32

4.14.2 Member Enumeration Documentation

4.14.2.1 anonymous enum

Enumerator:

N

4.14.2.2 anonymous enum

Enumerator:

SAVE

4.14.2.3 anonymous enum `[protected]`

Enumerator:

M

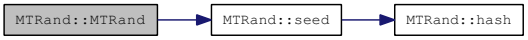
4.14.3 Constructor & Destructor Documentation

4.14.3.1 MTRand::MTRand (const uint32 *oneSeed*) `[inline]`

Here is the call graph for this function:

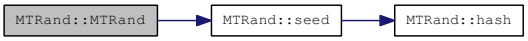
4.14.3.2 MTRand::MTRand (uint32 ∗const *bigSeed*, uint32 const *seedLength* = N) [inline]

Here is the call graph for this function:



4.14.3.3 MTRand::MTRand () [inline]

Here is the call graph for this function:



4.14.3.4 MTRand::MTRand (const MTRand & *o*) [inline]

4.14.4 Member Function Documentation

4.14.4.1 MTRand::uint32 MTRand::hash (time_t *t*, clock_t *c*) [inline, static, protected]

4.14.4.2 uint32 MTRand::hiBit (const uint32 *u*) const [inline, protected]

4.14.4.3 void MTRand::initialize (const uint32 *oneSeed*) [inline, protected]

4.14.4.4 void MTRand::load (uint32 ∗const *loadArray*) [inline]

4.14.4.5 uint32 MTRand::loBit (const uint32 *u*) const [inline, protected]

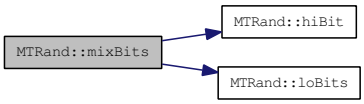
4.14.4.6 uint32 MTRand::loBits (const uint32 *u*) const [inline, protected]

4.14.4.7 uint32 MTRand::magic (const uint32 *u*) const [inline, protected]

Here is the call graph for this function:

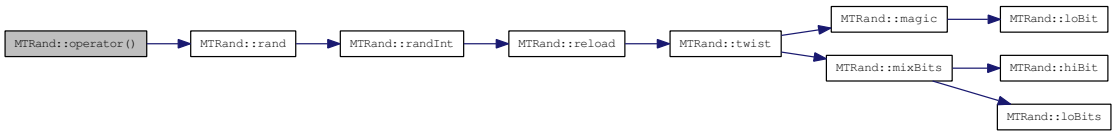
4.14.4.8 uint32 MTRand::mixBits (const uint32 *u*, const uint32 *v*) const [inline, protected]

Here is the call graph for this function:



4.14.4.9 double MTRand::operator() () [inline]

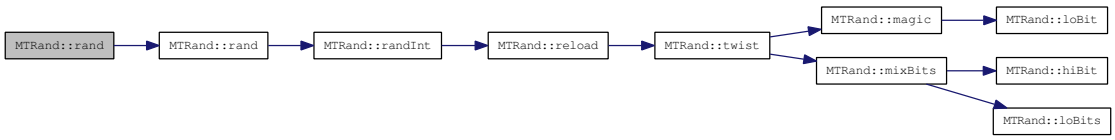
Here is the call graph for this function:



4.14.4.10 MTRand & MTRand::operator= (const MTRand & *o*) [inline]

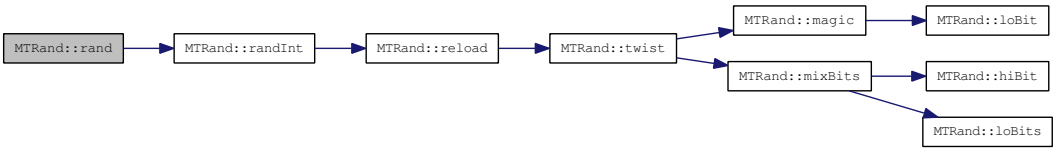
4.14.4.11 double MTRand::rand (const double *n*) [inline]

Here is the call graph for this function:



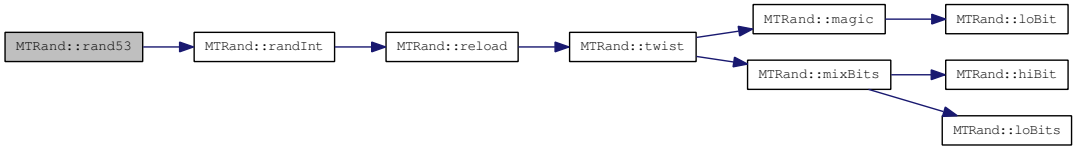
4.14.4.12 double MTRand::rand () [inline]

Here is the call graph for this function:



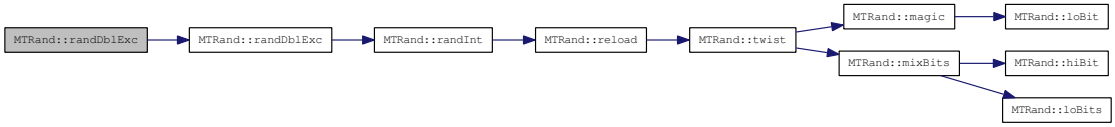
4.14.4.13 double MTRand::rand53 () [inline]

Here is the call graph for this function:



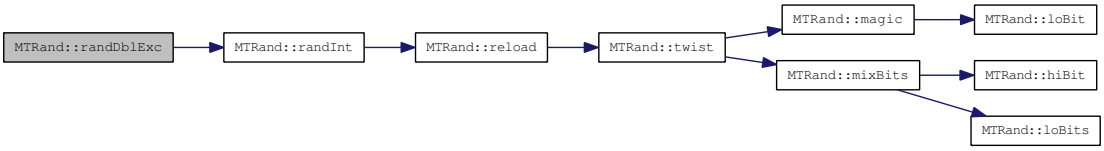
4.14.4.14 double MTRand::randDblExc (const double *n*) [inline]

Here is the call graph for this function:



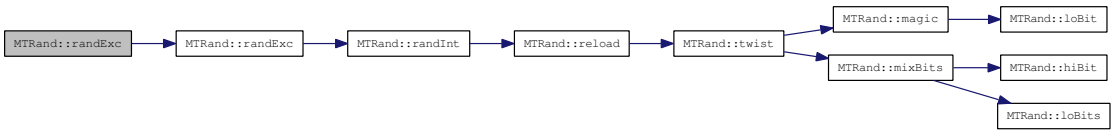
4.14.4.15 double MTRand::randDblExc () [inline]

Here is the call graph for this function:



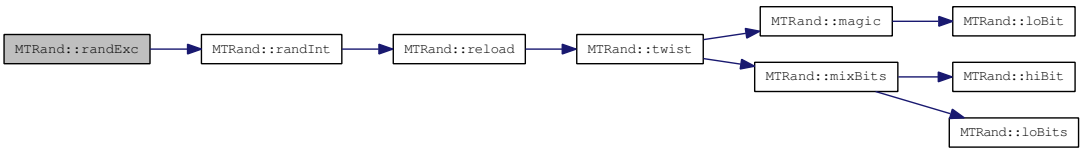
4.14.4.16 double MTRand::randExc (const double *n*) [inline]

Here is the call graph for this function:



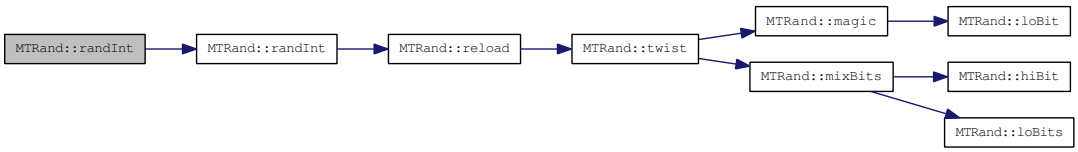
4.14.4.17 double MTRand::randExc () [inline]

Here is the call graph for this function:



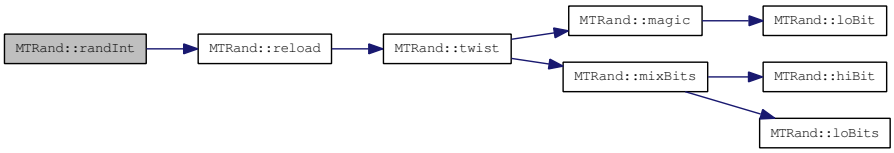
4.14.4.18 MTRand::uint32 MTRand::randInt (const uint32 *n*) [inline]

Here is the call graph for this function:



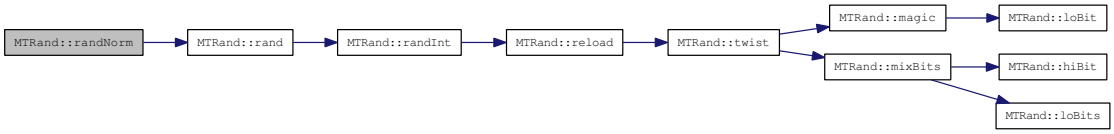
4.14.4.19 MTRand::uint32 MTRand::randInt () [inline]

Here is the call graph for this function:



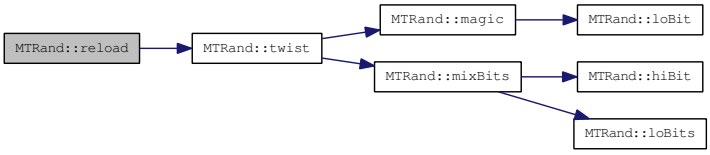
4.14.4.20 double MTRand::randNorm (const double *mean* = 0.0, const double *stddev* = 1.0) [inline]

Here is the call graph for this function:



4.14.4.21 void MTRand::reload () [inline, protected]

Here is the call graph for this function:



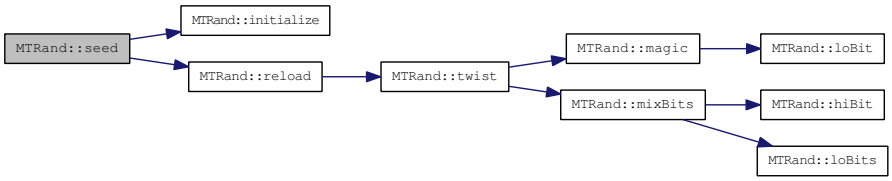
4.14.4.22 void MTRand::save (uint32 * *saveArray*) const [inline]

4.14.4.23 void MTRand::seed () [inline]

Here is the call graph for this function:

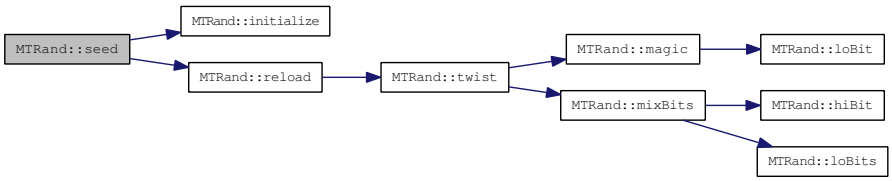
4.14.4.24 void MTRand::seed (uint32 *const *bigSeed*, const uint32 *seedLength* = N) [inline]

Here is the call graph for this function:



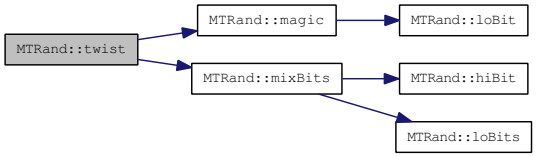
4.14.4.25 void MTRand::seed (const uint32 *oneSeed*) [inline]

Here is the call graph for this function:



4.14.4.26 uint32 MTRand::twist (const uint32 *m*, const uint32 *s0*, const uint32 *s1*) const [inline, protected]

Here is the call graph for this function:



4.14.5 Friends And Related Function Documentation

4.14.5.1 std::ostream& operator<< (std::ostream & *os*, const MTRand & *mtrand*) [friend]

4.14.5.2 std::istream& operator>> (std::istream & *is*, MTRand & *mtrand*) [friend]

4.14.6 Member Data Documentation

4.14.6.1 int MTRand::left [protected]

4.14.6.2 uint32* MTRand::pNext [protected]

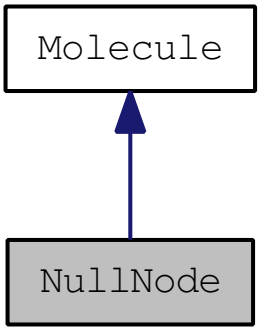
4.14.6.3 uint32 MTRand::state[N] [protected]

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

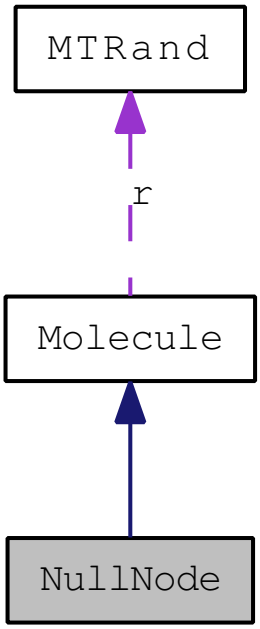
- MersenneTwister.h

4.15 NullNode Class Reference

`#include <CustomMolecules.h>`Inheritance diagram for NullNode:



Collaboration diagram for NullNode:



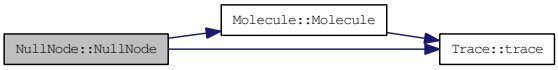
Public Member Functions

- NullNode ()
- ~NullNode ()
- virtual float getValue ()

4.15.1 Constructor & Destructor Documentation

4.15.1.1 NullNode::NullNode ()

Here is the call graph for this function:



4.15.1.2 NullNode::~NullNode ()

4.15.2 Member Function Documentation

4.15.2.1 float NullNode::getValue () [virtual]

float Molecule::getValue()

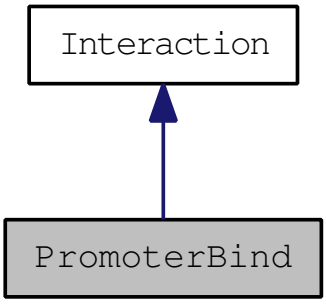
Reimplemented from Molecule.

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

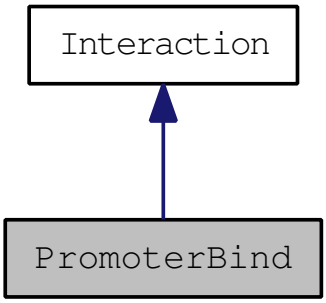
- CustomMolecules.h
- CustomMolecules.cpp

4.16 PromoterBind Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for PromoterBind:



Collaboration diagram for PromoterBind:



Public Member Functions

- PromoterBind (float, float)
- ~PromoterBind ()
- virtual float getEffect (ListDigraph *, ListDigraph::NodeMap< Molecule * > *, ListDigraph::ArcMap< Interaction * > *, ListDigraph::Node, int, float)

Public Attributes

- float kf
- float kr

4.16.1 Constructor & Destructor Documentation

4.16.1.1 PromoterBind::PromoterBind (float *fwdRate*, float *revRate*)

4.16.1.2 PromoterBind::~PromoterBind ()

4.16.2 Member Function Documentation

4.16.2.1 float PromoterBind::getEffect (ListDigraph * *g*, ListDigraph::NodeMap< Molecule * > * *m*, ListDigraph::ArcMap< Interaction * > * *i*, ListDigraph::Node *a*, int *rkIter*, float *rkStep*) [virtual]

float Interaction::getEffect(ListDigraph* , NodeMap<Molecule*>* , ArcMap<Interaction*>* , Node , int, float)

Get the effect this interaction has on a particular node.

This method defines the behavior of an interaction which connects two molecules. The effect on Node a can be dependent on any other molecule, which can be accessed using the ListDigraph, NodeMap, and ArcMap parameters.

Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary to call Molecule::rkApprox(stepsize, iteration) rather than Molecule::getValue() to get the current Iteration's approximated concentration.

Parameters:

g The graph object containing Node-Node relationships.

m The NodeMap object containing Node-Molecule mappings.

i The ArcMap object containing Arc-Interaction mappings.

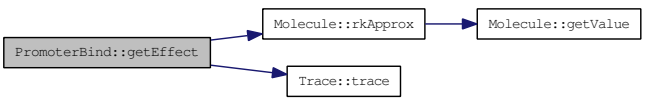
a The Node to calculate the effect for

rkIter The current iteration of Runge-Kutta [0,3]

rkStep The stepsize of Runge-Kutta

Reimplemented from Interaction.

Here is the call graph for this function:



4.16.3 Member Data Documentation

4.16.3.1 float PromoterBind::kf

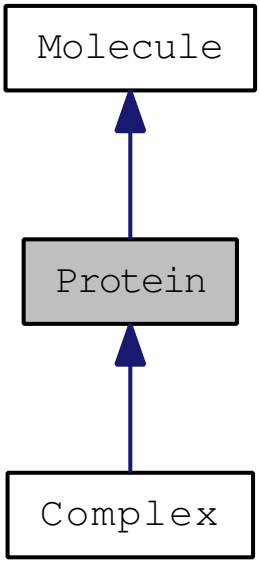
4.16.3.2 float PromoterBind::kr

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

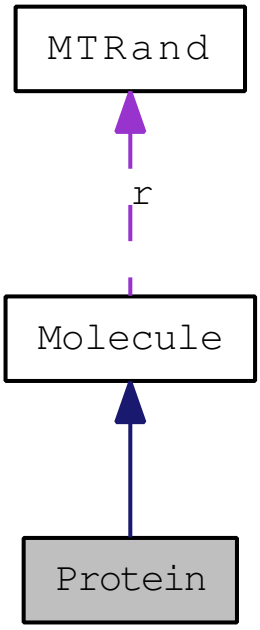
- CustomInteractions.h
- CustomInteractions.cpp

4.17 Protein Class Reference

`#include <CustomMolecules.h>`Inheritance diagram for Protein:



Collaboration diagram for Protein:



Public Member Functions

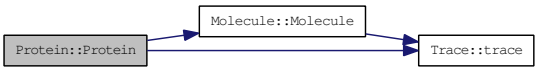
- Protein ()

- ~Protein ()

4.17.1 Constructor & Destructor Documentation

4.17.1.1 Protein::Protein ()

Here is the call graph for this function:



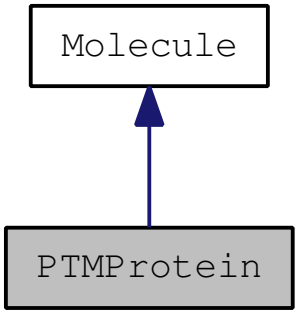
4.17.1.2 Protein::~Protein ()

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

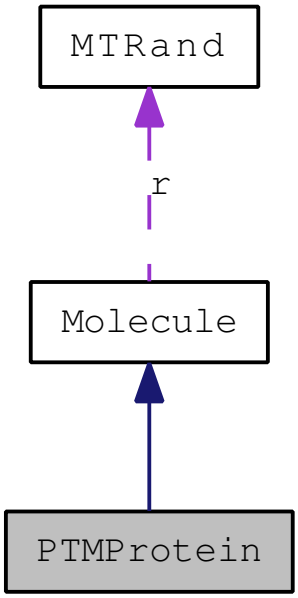
- CustomMolecules.h
- CustomMolecules.cpp

4.18 PTMProtein Class Reference

`#include <CustomMolecules.h>`Inheritance diagram for PTMProtein:



Collaboration diagram for PTMProtein:



Public Member Functions

- PTMProtein ()
- PTMProtein (PTMProtein ∗)
- ∼PTMProtein ()
- char ∗ getLongName ()
- void addRandPTM (int)
- void setPTMCount (int, int)

4.18.1 Constructor & Destructor Documentation

4.18.1.1 PTMProtein::PTMProtein ()

Here is the call graph for this function:

4.18.1.2 PTMProtein::PTMProtein (PTMProtein ∗ *c*)

4.18.1.3 PTMProtein::∼PTMProtein ()

4.18.2 Member Function Documentation

4.18.2.1 void PTMProtein::addRandPTM (int *i*)

4.18.2.2 char ∗ PTMProtein::getLongName () `[virtual]`

Reimplemented from Molecule.

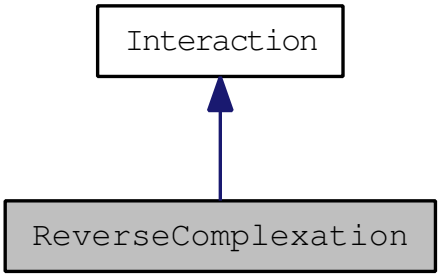
4.18.2.3 void PTMProtein::setPTMCount (int *index*, int *count*)

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

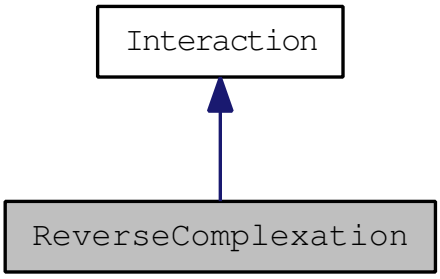
- CustomMolecules.h
- CustomMolecules.cpp

4.19 ReverseComplexation Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for ReverseComplexation:



Collaboration diagram for ReverseComplexation:



Public Member Functions

- ReverseComplexation (int, int)
- ~ReverseComplexation ()
- virtual float getEffect (ListDigraph *, ListDigraph::NodeMap< Molecule * > *, ListDigraph::ArcMap< Interaction * > *, ListDigraph::Node, int, float)

Public Attributes

- int firstNodeID
- int secondNodeID

4.19.1 Constructor & Destructor Documentation

4.19.1.1 ReverseComplexation::ReverseComplexation (int *n1*, int *n2*)

Here is the call graph for this function:

4.19.1.2 ReverseComplexation::∼ReverseComplexation ()

4.19.2 Member Function Documentation

4.19.2.1 float ReverseComplexation::getEffect (ListDigraph ∗ *g*, ListDigraph::NodeMap< Molecule ∗ > ∗ *m*, ListDigraph::ArcMap< Interaction ∗ > ∗ *i*, ListDigraph::Node *a*, int *rkIter*, float *rkStep*) [virtual]

float Interaction::getEffect(ListDigraph∗ , NodeMap<Molecule∗>∗ , ArcMap<Interaction∗>∗ , Node , int, float)

Get the effect this interaction has on a particular node.

This method defines the behavior of an interaction which connects two molecules. The effect on Node a can be dependent on any other molecule, which can be accessed using the ListDigraph, NodeMap, and ArcMap parameters.

Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary to call Molecule::rkApprox(stepsize, iteration) rather than Molecule::getValue() to get the current Iteration's approximated concentration.

Parameters:

g The graph object containing Node-Node relationships.

m The NodeMap object containing Node-Molecule mappings.

i The ArcMap object containing Arc-Interaction mappings.

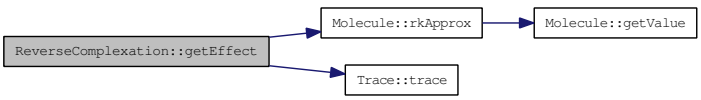
a The Node to calculate the effect for

rkIter The current iteration of Runge-Kutta [0,3]

rkStep The stepsize of Runge-Kutta

Reimplemented from Interaction.

Here is the call graph for this function:



4.19.3 Member Data Documentation

4.19.3.1 int ReverseComplexation::firstNodeID

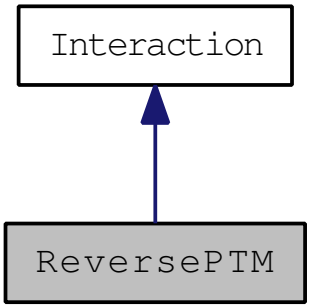
4.19.3.2 int ReverseComplexation::secondNodeID

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

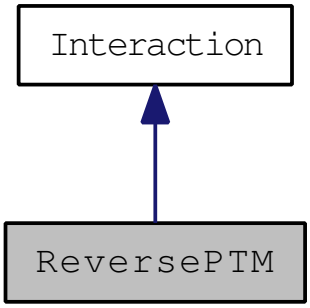
- CustomInteractions.h
- CustomInteractions.cpp

4.20 ReversePTM Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for ReversePTM:



Collaboration diagram for ReversePTM:



Public Member Functions

- ReversePTM ()
- ~ReversePTM ()

4.20.1 Constructor & Destructor Documentation

4.20.1.1 ReversePTM::ReversePTM ()

Here is the call graph for this function:

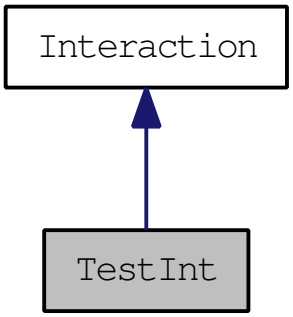
4.20.1.2 ReversePTM::~ReversePTM ()

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

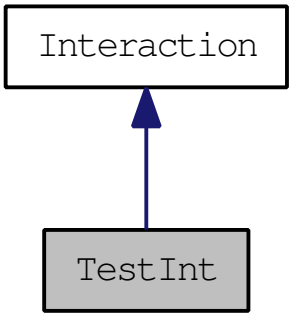
- CustomInteractions.h
- CustomInteractions.cpp

4.21 TestInt Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for TestInt:



Collaboration diagram for TestInt:



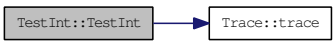
Public Member Functions

- TestInt ()
- ~TestInt ()
- virtual float getEffect (ListDigraph *, ListDigraph::NodeMap< Molecule * > *, ListDigraph::ArcMap< Interaction * > *, ListDigraph::Node, int, float)

4.21.1 Constructor & Destructor Documentation

4.21.1.1 TestInt::TestInt ()

Here is the call graph for this function:



4.21.1.2 TestInt::~TestInt ()

4.21.2 Member Function Documentation

4.21.2.1 float TestInt::getEffect (ListDigraph * *g*, ListDigraph::NodeMap< Molecule * > * *m*, ListDigraph::ArcMap< Interaction * > * *i*, ListDigraph::Node *a*, int *rkIter*, float *rkStep*) [virtual]

float Interaction::getEffect(ListDigraph* , NodeMap<Molecule*>* , ArcMap<Interaction*>* , Node , int, float)

Get the effect this interaction has on a particular node.

This method defines the behavior of an interaction which connects two molecules. The effect on Node a can be dependent on any other molecule, which can be accessed using the ListDigraph, NodeMap, and ArcMap parameters.

Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary to call Molecule::rkApprox(stepsize, iteration) rather than Molecule::getValue() to get the current Iteration's approximated concentration.

Parameters:

g The graph object containing Node-Node relationships.

m The NodeMap object containing Node-Molecule mappings.

i The ArcMap object containing Arc-Interaction mappings.

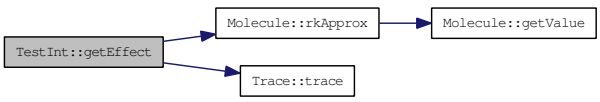
a The Node to calculate the effect for

rkIter The current iteration of Runge-Kutta [0,3]

rkStep The stepsize of Runge-Kutta

Reimplemented from Interaction.

Here is the call graph for this function:



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

- CustomInteractions.h
- CustomInteractions.cpp

4.22 Trace Class Reference

`#include <Trace.h>`

Public Member Functions

- Trace ()
- Trace (const char ∗)
- ∼Trace ()
- void addTraceType (const char ∗, int)
- void trace (const char ∗, const char ∗,...)
- FILE ∗ getTraceFile ()
- FILE ∗ setTraceFile (FILE ∗)
- void enableTraceType (const char ∗)
- void disableTraceType (const char ∗)

Public Attributes

- FILE ∗ traceFile
- map< const char ∗, int, cmp_str > traceTypes

4.22.1 Constructor & Destructor Documentation

4.22.1.1 Trace::Trace ()

Trace.cpp

Trace implementation.

Trace messages are called with a tag that can be optionally turned on or off with a simple call.

This allows trace messages to be given context and turned on or off very flexibly. Trace::Trace()

Default Constructor.

4.22.1.2 Trace::Trace (const char ∗ *c*)

4.22.1.3 Trace::∼Trace ()

Trace::∼Trace()

Default Destructor.

4.22.2 Member Function Documentation

4.22.2.1 void Trace::addTraceType (const char ∗ *tag*, int *enabled*)

void Trace::addTraceType(const char∗, int)

Adds a new trace tag and enables it.

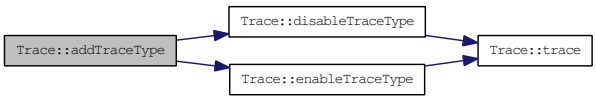
e.g. Trace::addTraceType("output") //output related t.trace("output", "Output complete");

Parameters:

tag The tag to be used for tracing

enabled Initial state of the trace type. Nonzero is enabled.

Here is the call graph for this function:



4.22.2.2 void Trace::disableTraceType (const char * *tag*)

Trace::disableTraceType(const char*)

Disable the trace type, causing future trace messages tagged with this type to be surpressed.

Parameters:

tag the trace tag to disable.

Here is the call graph for this function:

4.22.2.3 void Trace::enableTraceType (const char * *tag*)

Trace::enableTraceType(const char*)

Enable the trace type, causing future trace messages tagged with this type to be output.

Parameters:

tag the trace tag to enable.

Here is the call graph for this function:

4.22.2.4 FILE * Trace::getTraceFile ()

4.22.2.5 FILE * Trace::setTraceFile (FILE * *tf*)

4.22.2.6 void Trace::trace (const char * *tag*, const char * *format*, ...)

void Trace::trace(const char*, const char*, ...)

Outputs a trace message with the given format if the trace tag is enabled. Output is not automatically terminated with a newline character.

Parameters:

tag Trace type

format string

... variable arguments corresponding to format string

4.22.3 Member Data Documentation

4.22.3.1 FILE* Trace::traceFile

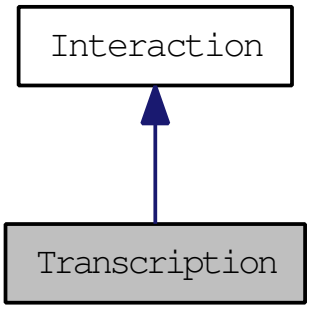
4.22.3.2 map<const char*, int, cmp_str> Trace::traceTypes

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

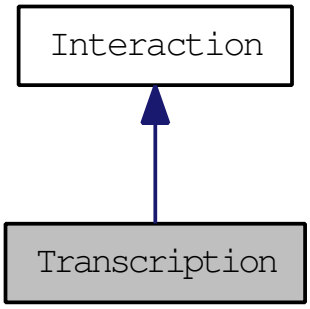
- Trace.h
- Trace.cpp

4.23 Transcription Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for Transcription:



Collaboration diagram for Transcription:



Public Member Functions

- Transcription ()
- ~Transcription ()
- virtual float getEffect (ListDigraph *, ListDigraph::NodeMap< Molecule * > *, ListDigraph::ArcMap< Interaction * > *, ListDigraph::Node, int, float)

4.23.1 Constructor & Destructor Documentation

4.23.1.1 Transcription::Transcription ()

Implementation file for Custom Interactions.

Interactions may overload the virtual method getEffect() to create a custom effect between Molecules

4.23.1.2 Transcription::~Transcription ()

4.23.2 Member Function Documentation

4.23.2.1 float Transcription::getEffect (ListDigraph * *g*, ListDigraph::NodeMap< Molecule * > * *m*, ListDigraph::ArcMap< Interaction * > * *i*, ListDigraph::Node *a*, int *rkIter*, float *rkStep*) [virtual]

float Interaction::getEffect(ListDigraph* , NodeMap<Molecule*>* , ArcMap<Interaction*>* , Node , int, float)

Get the effect this interaction has on a particular node.

This method defines the behavior of an interaction which connects two molecules. The effect on Node a can be dependent on any other molecule, which can be accessed using the ListDigraph, NodeMap, and ArcMap parameters.

Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary to call Molecule::rkApprox(stepsize, iteration) rather than Molecule::getValue() to get the current Iteration's approximated concentration.

Parameters:

g The graph object containing Node-Node relationships.

m The NodeMap object containing Node-Molecule mappings.

i The ArcMap object containing Arc-Interaction mappings.

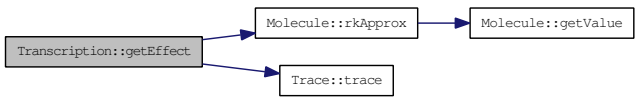
a The Node to calculate the effect for

rkIter The current iteration of Runge-Kutta [0,3]

rkStep The stepsize of Runge-Kutta

Reimplemented from Interaction.

Here is the call graph for this function:

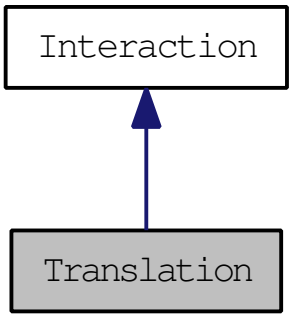


The documentation for this class was generated from the following files:

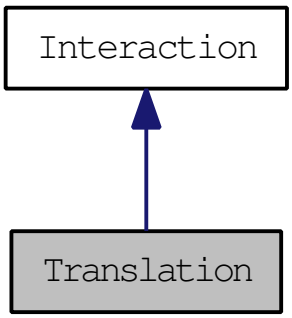
- CustomInteractions.h
- CustomInteractions.cpp

4.24 Translation Class Reference

`#include <CustomInteractions.h>`Inheritance diagram for Translation:



Collaboration diagram for Translation:



Public Member Functions

- Translation ()
- ~Translation ()

4.24.1 Constructor & Destructor Documentation

4.24.1.1 Translation::Translation ()

Here is the call graph for this function:

4.24.1.2 Translation::~Translation ()

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

- CustomInteractions.h
- CustomInteractions.cpp

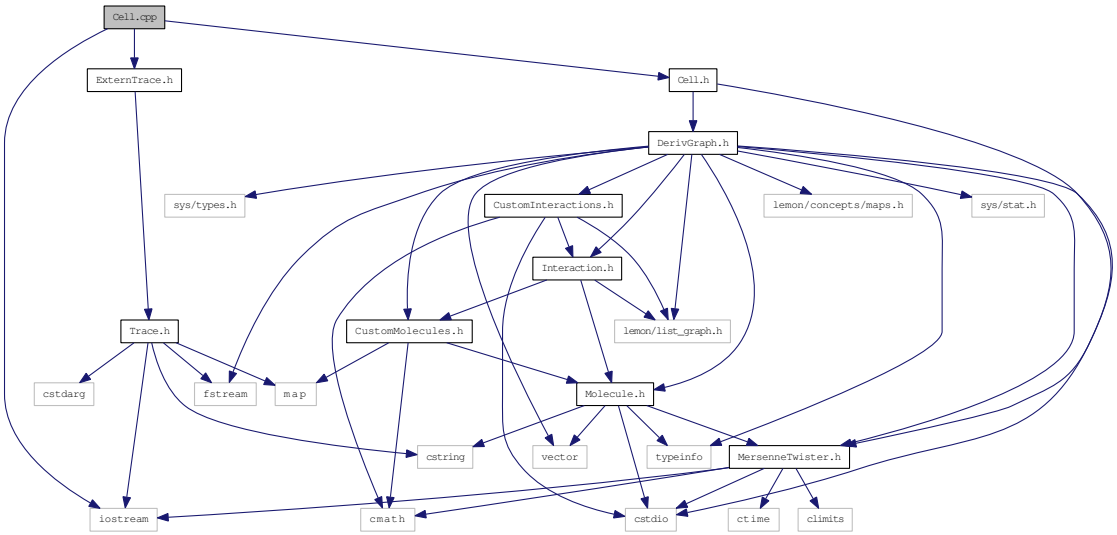
Chapter 5

File Documentation

5.1 Cell.cpp File Reference

```
#include <iostream>
#include "Cell.h"
#include "ExternTrace.h"
```

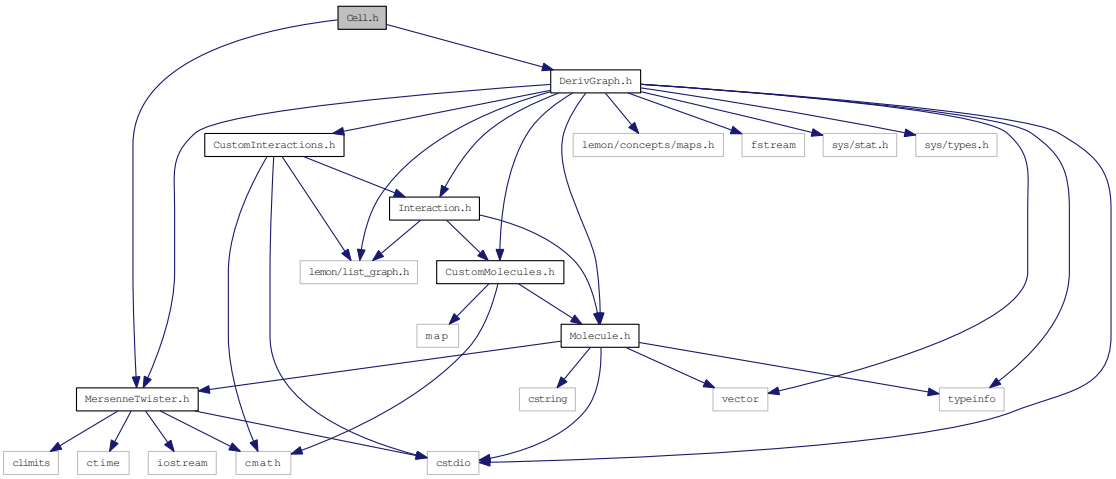
Include dependency graph for Cell.cpp:



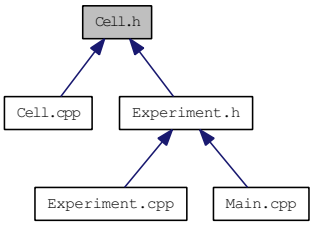
5.2 Cell.h File Reference

```
#include "MersenneTwister.h"
#include "DerivGraph.h"
```

Include dependency graph for Cell.h:



This graph shows which files directly or indirectly include this file:



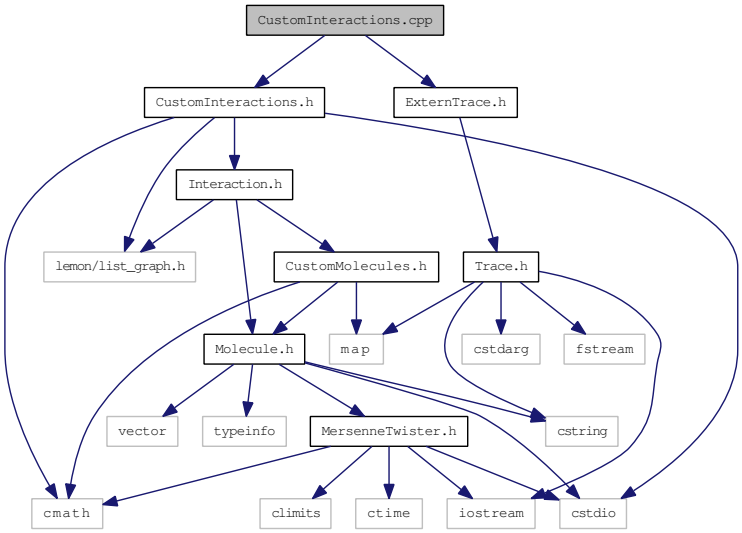
Classes

- class Cell

5.3 CustomInteractions.cpp File Reference

```
#include "CustomInteractions.h"
#include "ExternTrace.h"
```

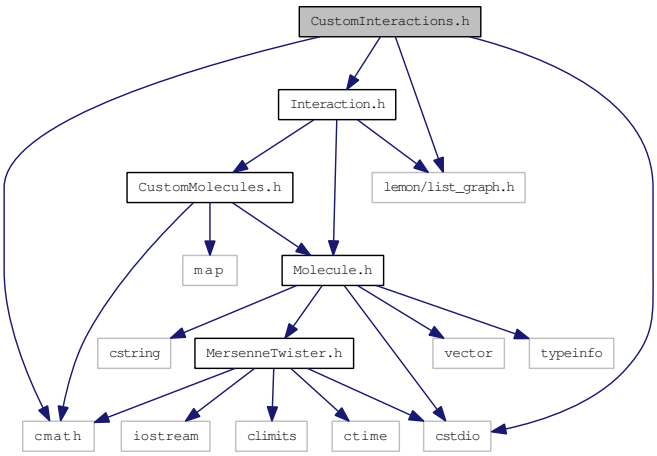
Include dependency graph for CustomInteractions.cpp:



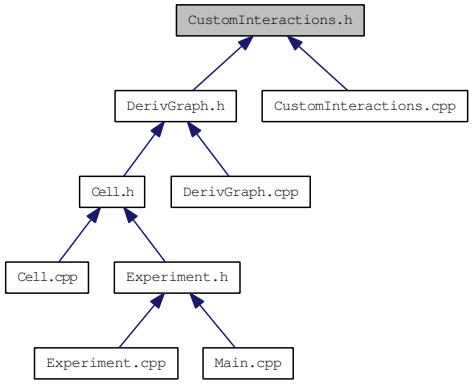
5.4 CustomInteractions.h File Reference

```
#include <cmath>
#include "Interaction.h"
#include <cstdio>
#include "lemon/list_graph.h"
```

Include dependency graph for CustomInteractions.h:



This graph shows which files directly or indirectly include this file:



Classes

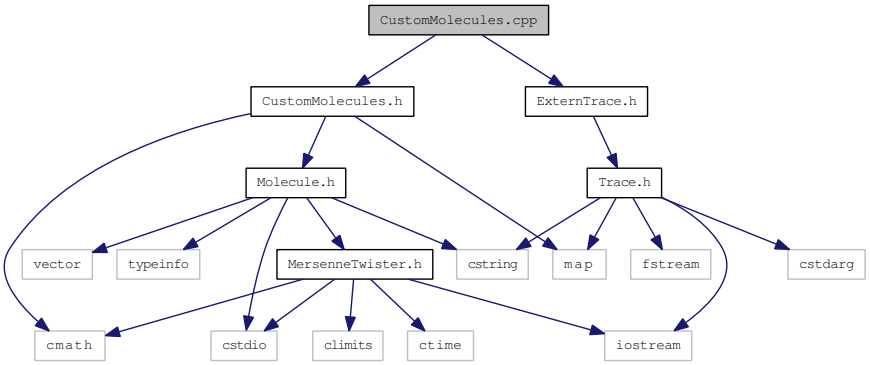
- class TestInt
- class Transcription
- class Degradation
- class Translation
- class ForwardComplexation
- class ReverseComplexation
- class ForwardPTM

- class ReversePTM
- class PromoterBind

5.5 CustomMolecules.cpp File Reference

```
#include "CustomMolecules.h"
#include "ExternTrace.h"
```

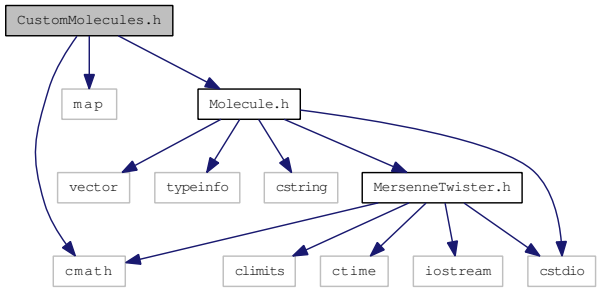
Include dependency graph for CustomMolecules.cpp:



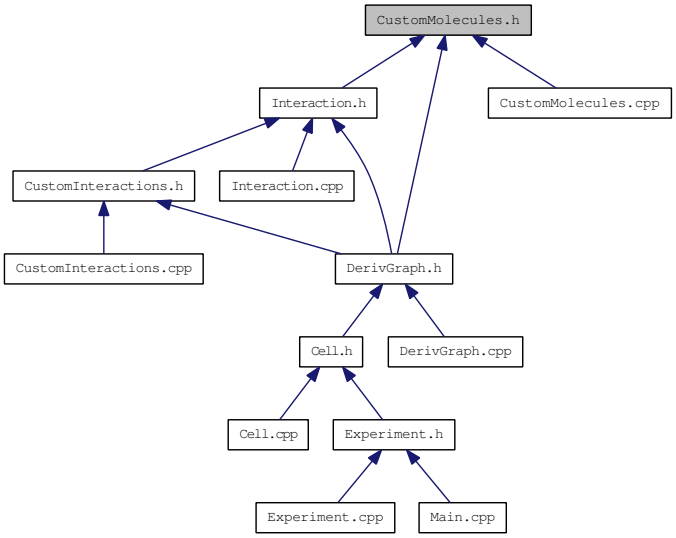
5.6 CustomMolecules.h File Reference

```
#include <cmath>
#include <map>
#include "Molecule.h"
```

Include dependency graph for CustomMolecules.h:



This graph shows which files directly or indirectly include this file:



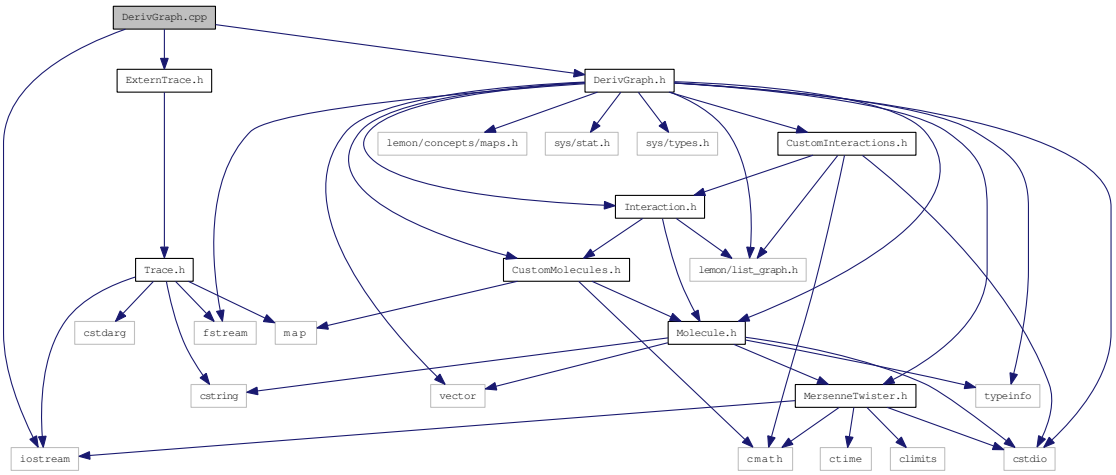
Classes

- class PTMProtein
- class DNA
- class NullNode
- class mRNA
- class Protein
- class Complex

5.7 DerivGraph.cpp File Reference

```
#include <iostream>
#include "DerivGraph.h"
#include "ExternTrace.h"
```

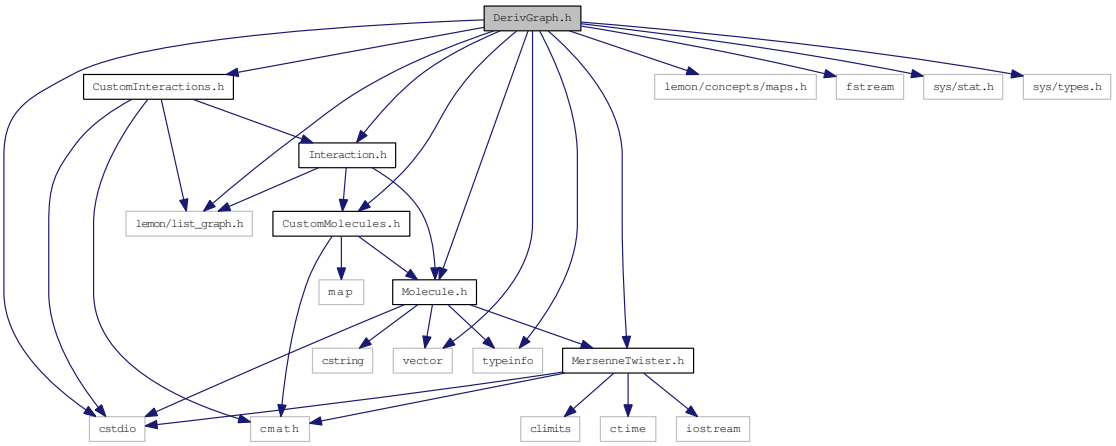
Include dependency graph for DerivGraph.cpp:



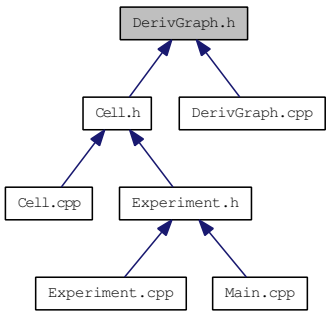
5.8 DerivGraph.h File Reference

```
#include "lemon/list_graph.h"
#include "lemon/concepts/maps.h"
#include <cstdio>
#include <vector>
#include <fstream>
#include <sys/stat.h>
#include <sys/types.h>
#include <typeinfo>
#include "MersenneTwister.h"
#include "Molecule.h"
#include "Interaction.h"
#include "CustomInteractions.h"
#include "CustomMolecules.h"
```

Include dependency graph for DerivGraph.h:



This graph shows which files directly or indirectly include this file:



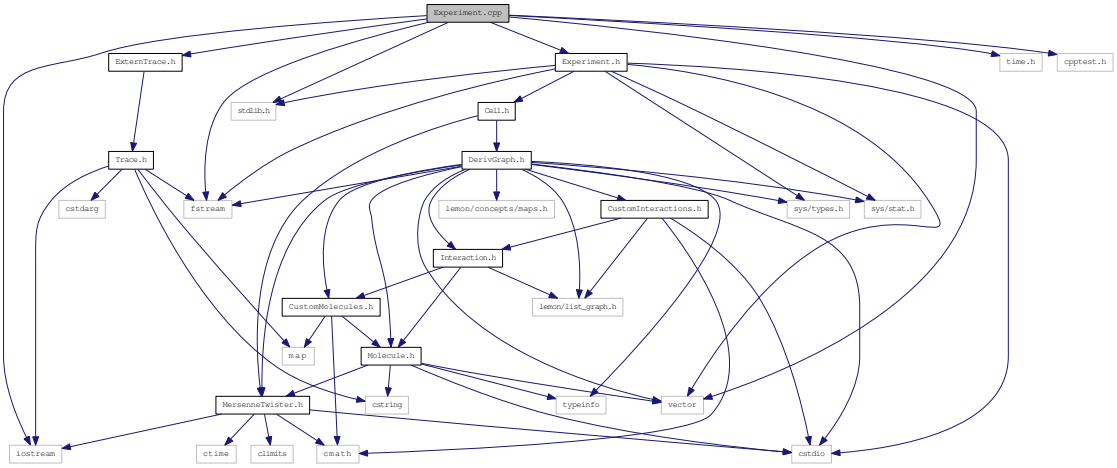
Classes

- class DerivGraph

5.9 Experiment.cpp File Reference

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <vector>
#include "Experiment.h"
#include "cpptest.h"
#include "ExternTrace.h"
```

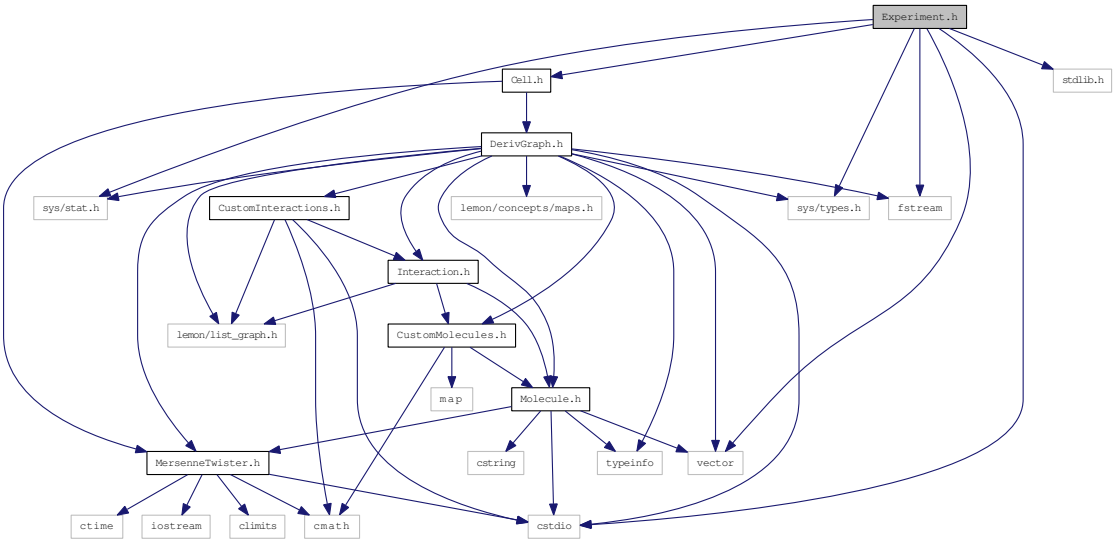
Include dependency graph for Experiment.cpp:



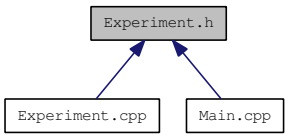
5.10 Experiment.h File Reference

```
#include <sys/stat.h>
#include <sys/types.h>
#include <cstdio>
#include <stdlib.h>
#include <vector>
#include <fstream>
#include "Cell.h"
```

Include dependency graph for Experiment.h:



This graph shows which files directly or indirectly include this file:



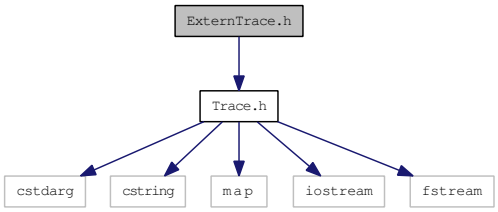
Classes

- class Experiment

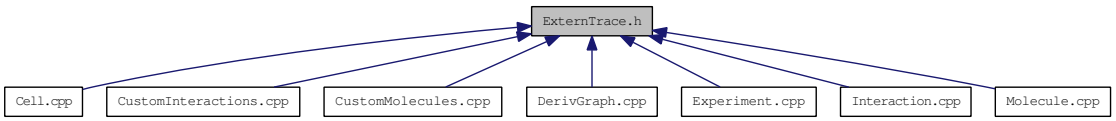
5.11 ExternTrace.h File Reference

```
#include "Trace.h"
```

Include dependency graph for ExternTrace.h:



This graph shows which files directly or indirectly include this file:



Variables

- Trace t

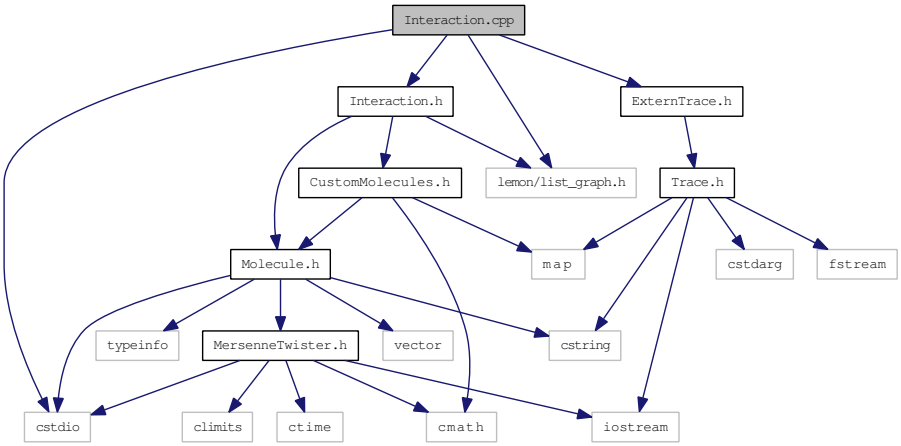
5.11.1 Variable Documentation

5.11.1.1 Trace t

5.12 Interaction.cpp File Reference

```
#include "Interaction.h"
#include <cstdio>
#include "lemon/list_graph.h"
#include "ExternTrace.h"
```

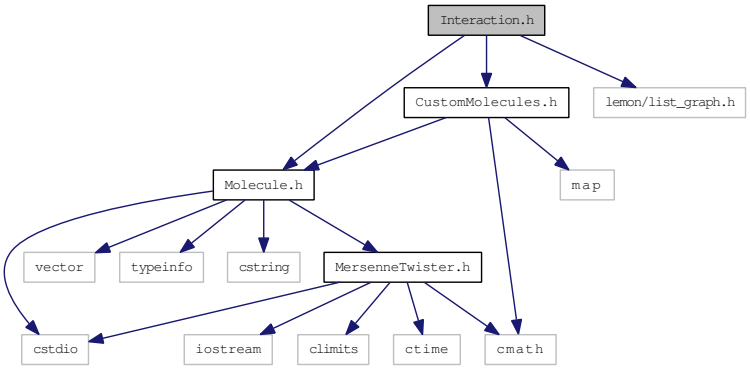
Include dependency graph for Interaction.cpp:



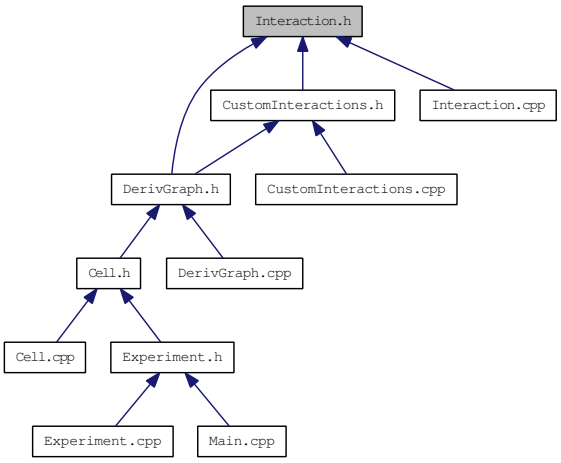
5.13 Interaction.h File Reference

```
#include "Molecule.h"
#include "CustomMolecules.h"
#include "lemon/list_graph.h"
```

Include dependency graph for Interaction.h:



This graph shows which files directly or indirectly include this file:



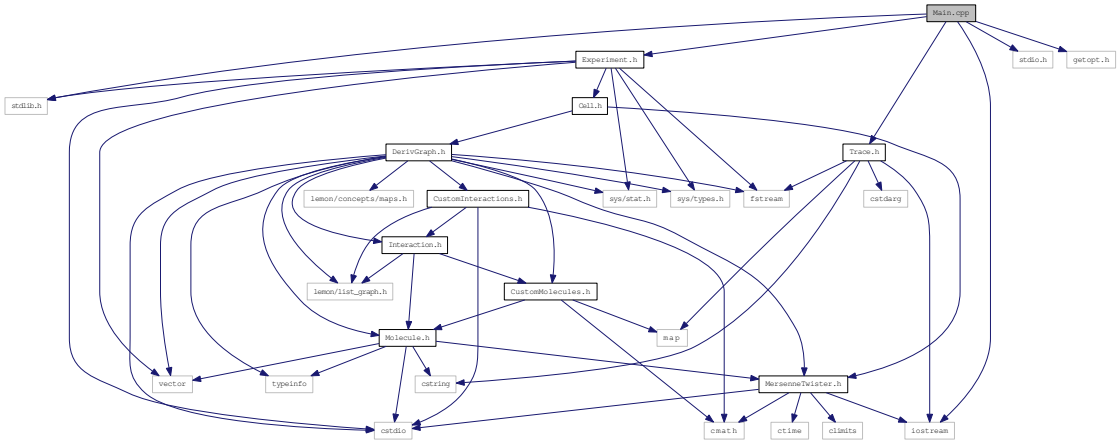
Classes

- class Interaction

5.14 Main.cpp File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <iostream>
#include "Experiment.h"
#include "Trace.h"
```

Include dependency graph for Main.cpp:



Functions

- int main (int argc, char **argv)

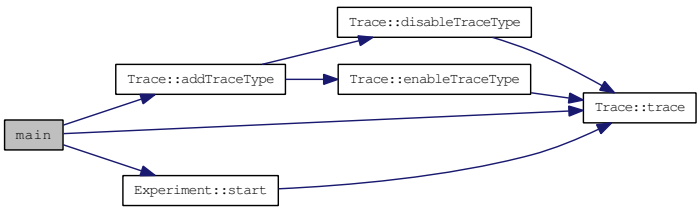
Variables

- Trace t

5.14.1 Function Documentation

5.14.1.1 int main (int *argc*, char ** *argv*)

Here is the call graph for this function:



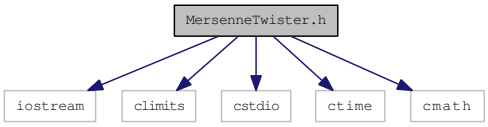
5.14.2 Variable Documentation

5.14.2.1 Trace t

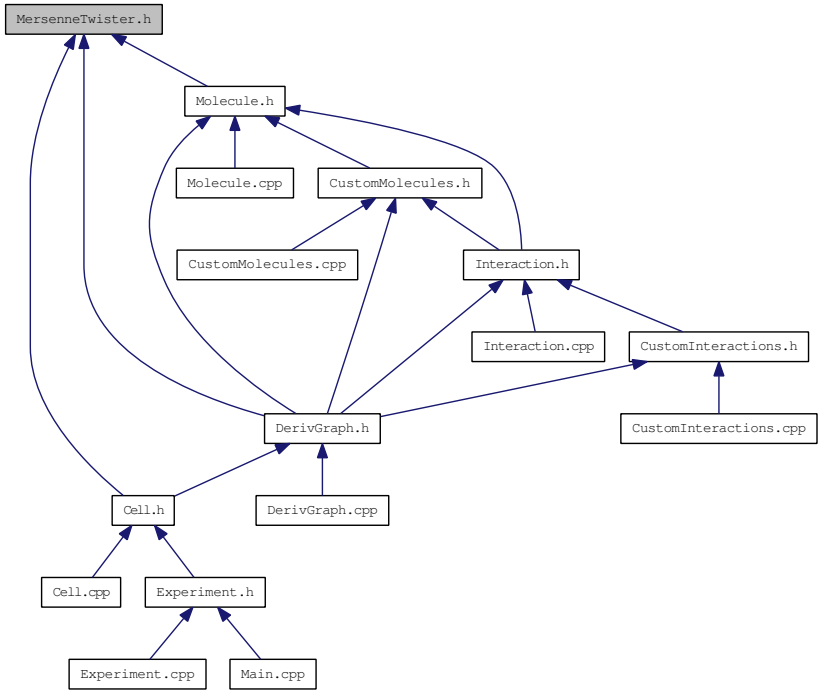
5.15 MersenneTwister.h File Reference

```
#include <iostream>
#include <climits>
#include <cstdio>
#include <ctime>
#include <cmath>
```

Include dependency graph for MersenneTwister.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MTRand

Functions

- std::ostream & operator<< (std::ostream &os, const MTRand &mtrand)
- std::istream & operator>> (std::istream &is, MTRand &mtrand)

5.15.1 Function Documentation

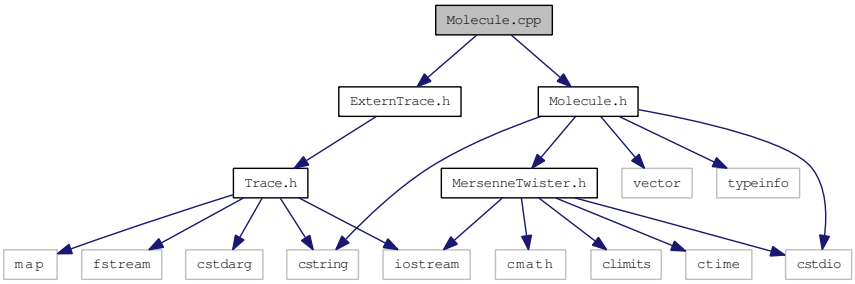
5.15.1.1 std::ostream& operator<< (std::ostream & *os*, const MTRand & *mtrand*) [inline]

5.15.1.2 std::istream& operator>> (std::istream & *is*, MTRand & *mtrand*) [inline]

5.16 Molecule.cpp File Reference

```
#include "Molecule.h"
#include "ExternTrace.h"
```

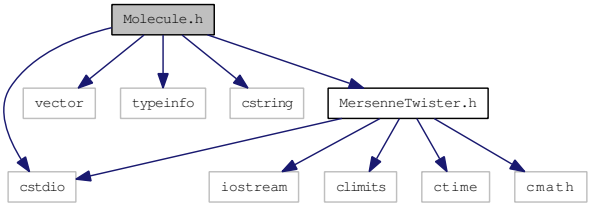
Include dependency graph for Molecule.cpp:



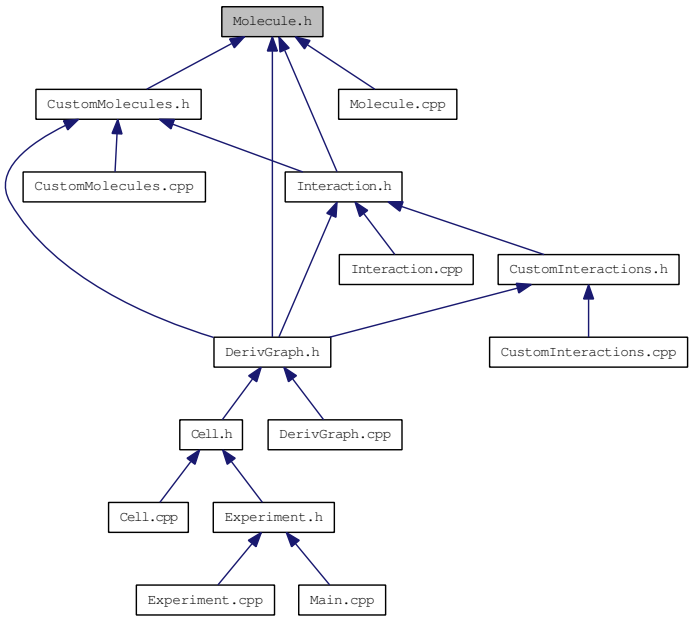
5.17 Molecule.h File Reference

```
#include <cstdio>
#include <vector>
#include <typeinfo>
#include <cstring>
#include "MersenneTwister.h"
```

Include dependency graph for Molecule.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Molecule

5.18 MoleculeType.h File Reference

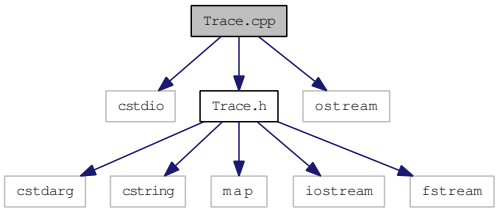
Classes

- class MoleculeType

5.19 Trace.cpp File Reference

```
#include <cstdio>
#include "Trace.h"
#include <ostream>
```

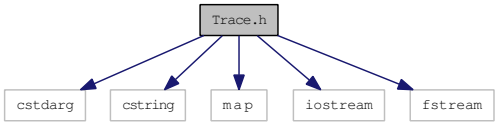
Include dependency graph for Trace.cpp:



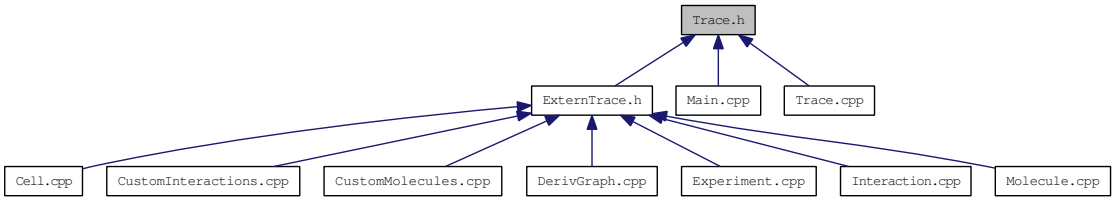
5.20 Trace.h File Reference

```
#include <cstdarg>
#include <cstring>
#include <map>
#include <iostream>
#include <fstream>
```

Include dependency graph for Trace.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cmp_str
- class Trace

Index