# Reference Manual

Generated by Doxygen 1.6.2-20100208

Mon Jul 11 04:48:43 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

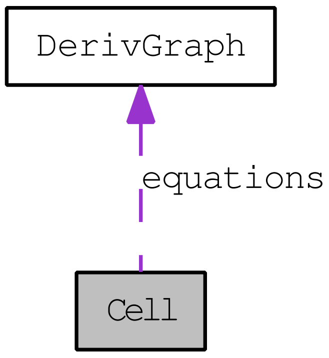
Here is a list of all files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 Cell Class Reference

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



### Public Member Functions

- Cell (int, int, int, int, float, float, float, float, float)
- ~Cell ()
- int mutate ()
- void outputDotImage ()
- void outputDataPlot ()
- int getScore ()
- int getID ()
- void rkTest ()

### 4.1.1 Detailed Description

Cell header file.

### 4.1.2 Constructor & Destructor Documentation

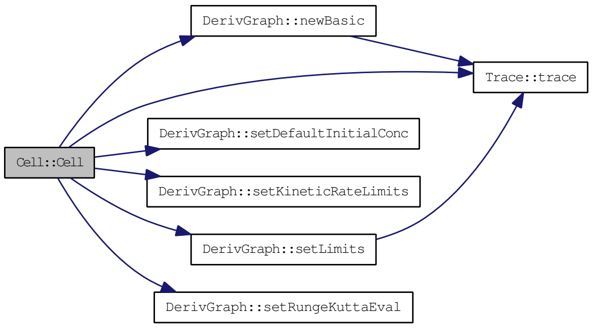
#### 4.1.2.1 Cell::Cell (int *max_basic*, int *max_ptm*, int *max_comp*, int *max_promoter*, float *min_kinetic_rate*, float *max_kinetic_rate*, float *rk_time_step*, float *rk_time_limit*, float *initial_conc*)

Cell::Cell()

Cell default constructor.

Allocates: 1 derivGraph object

Here is the call graph for this function:


Cell::Cell
DerivGraph::newBasic
Trace::trace
DerivGraph::setDefaultInitialConc
DerivGraph::setKineticRateLimits
DerivGraph::setLimits
DerivGraph::setRungeKuttaEval

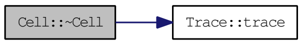
#### 4.1.2.2 Cell::∼Cell ()

Cell::∼Cell()

Cell default destructor.

Frees: 1 derivGraph object

Here is the call graph for this function:

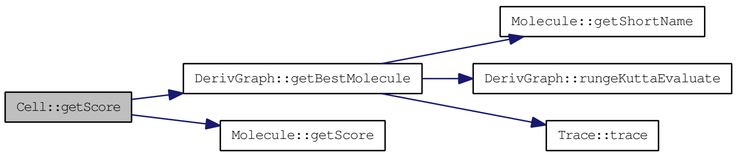


### 4.1.3 Member Function Documentation

#### 4.1.3.1 int Cell::getID () `[inline]`

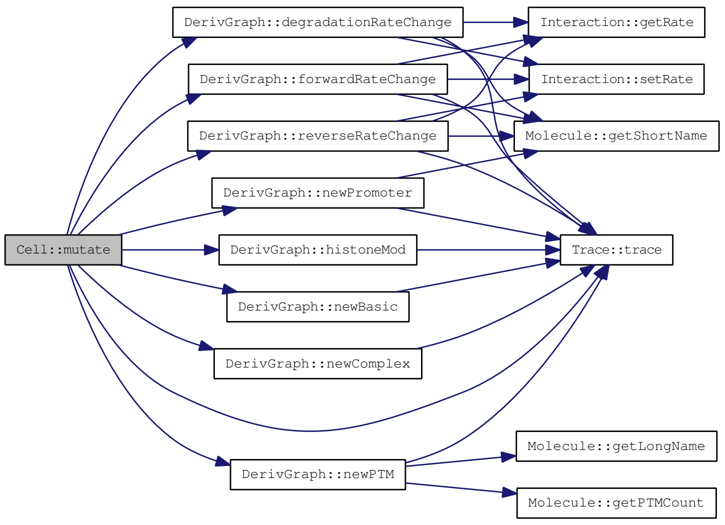
#### 4.1.3.2 int Cell::getScore ()

Here is the call graph for this function:



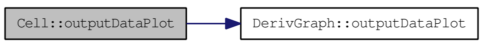
#### 4.1.3.3 int Cell::mutate ()

Here is the call graph for this function:



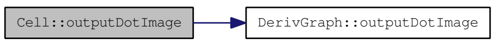
#### 4.1.3.4 void Cell::outputDataPlot ()

Here is the call graph for this function:



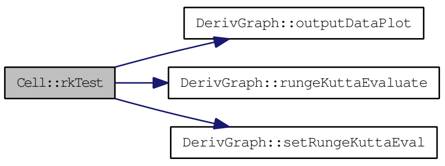
#### 4.1.3.5 void Cell::outputDotImage ()

Here is the call graph for this function:



#### 4.1.3.6 void Cell::rkTest ()

Here is the call graph for this function:


Cell::rkTest
DerivGraph::outputDataPlot
DerivGraph::rungeKuttaEvaluate
DerivGraph::setRungeKuttaEval

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

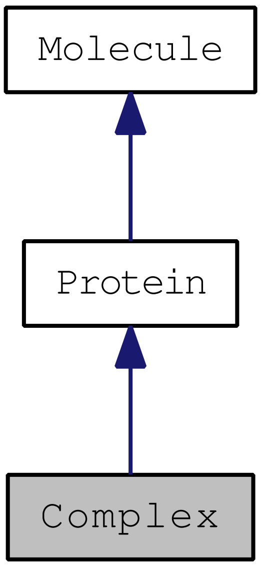
#### 4.2.1.1 bool cmp_str::operator() (const char $*$ *a*, const char $*$ *b*) `[inline]`

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

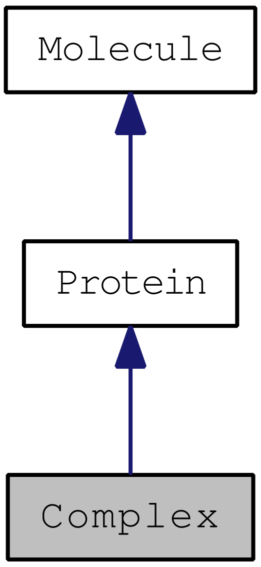
- Trace.h

## 4.3 Complex Class Reference

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



Collaboration diagram for Complex:



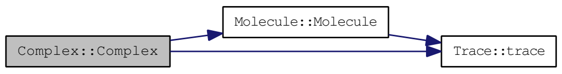
### Public Member Functions

- 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

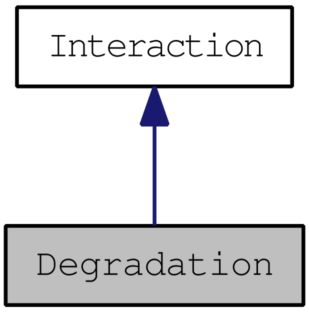
#### 4.3.2.1 int Complex::getComponentId (int *i*)

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

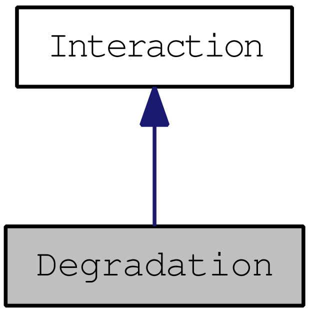
- CustomMolecules.h
- CustomMolecules.cpp

## 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.

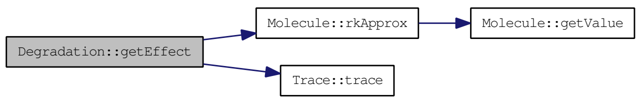
***a*** The Node to calculate the effect for

***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>
```

### Public Member Functions

- DerivGraph ()
- ∼DerivGraph ()
- void test ()
- void rungeKuttaEvaluate (float, float)
- void outputDotImage (int, int)
- void outputDataPlot (int, int, float)
- Molecule ∗ getBestMolecule (int)
- void setLimits (int, int, int, int)
- void setKineticRateLimits (float, float)
- void setRungeKuttaEval (float, float)
- void setDefaultInitialConc (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.

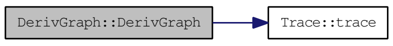
The DerivGraph holds LEMON objects such as ListDigraph, NodeMap, and ArcMap. It also holds the data produced by Runge-Kutta and facilitates plotting using Gnuplot.

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

Here is the call graph for this function:



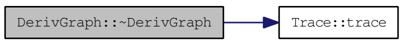
#### 4.5.1.2 DerivGraph::∼DerivGraph ()

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

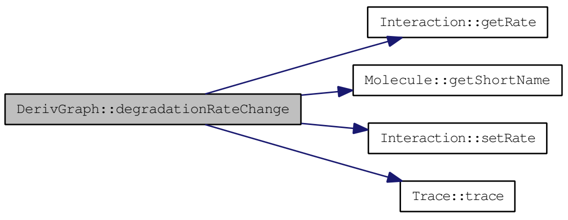
#### 4.5.2.1 void DerivGraph::degradationRateChange ()

void DerivGraph::degradationRateChange()

Randomly select a degradation interaction and modify its rate.

Degradation interactions are of type Degradation

Here is the call graph for this function:



#### 4.5.2.2 void DerivGraph::forwardRateChange ()

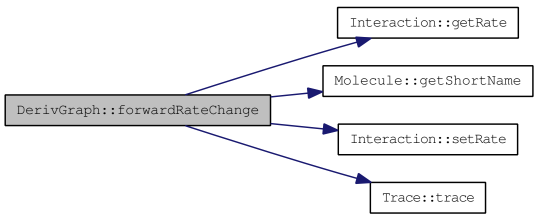
void DerivGraph::forwardRateChange()

Randomly select a forward interaction and modify its rate.

Forward interactions are of type Translation, ForwardComplex

TODO: add ForwardPTMs

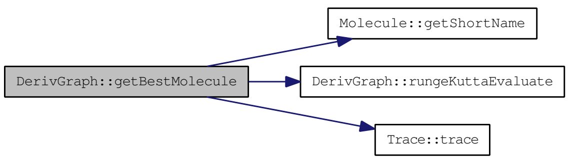
Here is the call graph for this function:



#### 4.5.2.3 ListDigraph::ArcMap<Interaction∗>∗ DerivGraph::getArcMap ()

#### 4.5.2.4 Molecule ∗ DerivGraph::getBestMolecule (int *CellID*)

Here is the call graph for this function:



#### 4.5.2.5 ListDigraph∗ DerivGraph::getListDigraph ()

#### 4.5.2.6 ListDigraph::NodeMap<Molecule∗>∗ DerivGraph::getNodeMap ()

#### 4.5.2.7 DNA ∗ DerivGraph::histoneMod ()

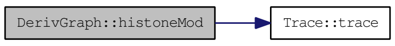
void DerivGraph::histoneMod()

Randomly select a DNA molecule, and set the Histone factor to a random value [0,2].

The histone value is a constant multiplied factor applied to the rate of mRNA production by a DNA molecule. It is initialized at 1.0, and is randomly assigned a value between [0,2].

A value [0,1) results in repression of mRNA production. A value (1,2] results in activation of mRNA production.

Here is the call graph for this function:



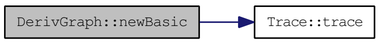
#### 4.5.2.8 void DerivGraph::newBasic ()

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:



#### 4.5.2.9 void DerivGraph::newComplex ()

void DerivGraph::newComplex()

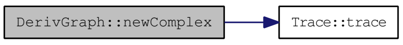
Randomly select two Protein molecules to be complexed together.

If the two selected proteins already exist in a complex reaction together, the mutation will fail.

Molecules which can complex together are Protein, and ComplexProteins

TODO: add PTM's to the possible complex

Here is the call graph for this function:



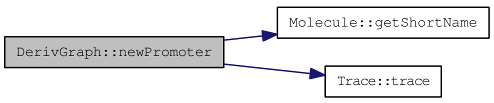
#### 4.5.2.10 void DerivGraph::newPromoter ()

void DerivGraph::newPromoter()

Select a random protein and DNA to add a Protein-Promoter interaction to.

The Protein-Promoter interaction is used in conjunction with the Hill model of cooperativity, and affects the Goodwin term used by DNA to calculate the rate of production.

Here is the call graph for this function:



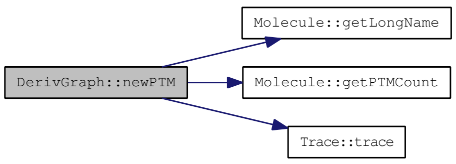
#### 4.5.2.11 void DerivGraph::newPTM ()

void DerivGraph::newPTM()

Randomly select a Protein molecule to which a PTM should be applied. The new PTM Protein has the same counts of modifications, with a random index incremented by one to reflect the new value. A PTM can be applied to a basic protein or a previously existing PTM.

TODO: check if a PTM already exists on a protein and prevent duplicate PTM's

Here is the call graph for this function:



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

void DerivGraph::outputDataPlot(int, int, float)

Output a png image of the concentration data of molecules plotted by Gnuplot.

For each molecule in the MoleculeList, a process running gnuplot is forked to which data from Runge-Kutta is fed to produce a plot.

**Parameters:**

***cellNum*** the cell number to put in the filename

***gen*** the generation number to put in the filename

***step*** the stepSize used between the rungeKuttaSolution data points

#### 4.5.2.13 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

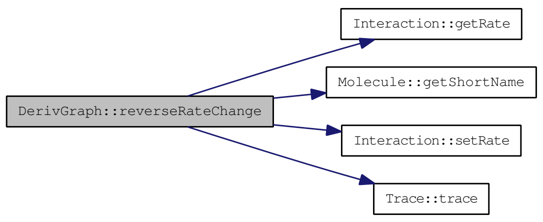
#### 4.5.2.14 void DerivGraph::reverseRateChange ()

void DerivGraph::reverseRateChange()

Randomly select a reverse interaction and modify its rate.

Reverse interactions are of type ReverseComplexation, ReversePTM

Here is the call graph for this function:



#### 4.5.2.15 void DerivGraph::rungeKuttaEvaluate (float *rkStep*, float *rkLimit*)

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

#### 4.5.2.16 void DerivGraph::setDefaultInitialConc (float *initial_conc*)

DerivGraph::setDefaultInitialConcentration(float)

Set the default initial concentration for molecules

**Parameters:**

***initial_conc*** The initial concentration for new molecules

#### 4.5.2.17 void DerivGraph::setKineticRateLimits (float *min_kinetic_rate*, float *max_kinetic_rate*)

DerivGraph::setKineticRateLimits(float, float)

Assign lower and upper bounds to the randomly generated kinetic rates

**Parameters:**

***min_kinetic_rate*** The lower bound on random kinetic rates

***max_kinetic_rate*** The upper bound on random kinetic rates

#### 4.5.2.18 void DerivGraph::setLimits (int *max_basic*, int *max_ptm*, int *max_comp*, int *max_promoter*)

DerivGraph::setLimits(int, int, int, int)

Set the occurrence limits for mutation types

**Parameters:**

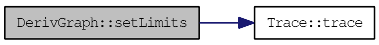
***max_basic*** Maximum basic proteins allowed

***max_ptm*** Maximum number of post translationally modified proteins allowed

***max_comp*** Maximum number of complexed proteins allowed

***max_promoter*** Maximum number of protein-promoter interactions allowed

Here is the call graph for this function:



#### 4.5.2.19 void DerivGraph::setRungeKuttaEval (float *rk_time_step*, float *rk_time_limit*)

DerivGraph::setRungeKuttaEval(float, float)

Set the parameters for runge-kutta evalutation

**Parameters:**

***rk_time_step*** The timestep between points (t, conc) calculated by Runge-Kutta

***rk_time_limit*** The upper time limit for Runge-Kutta calculation (time = x axis)

#### 4.5.2.20 void DerivGraph::test ()

Here is the call graph for this function:

DerivGraph::test
DerivGraph::rungeKuttaEvaluate

### 4.5.3 Member Data Documentation

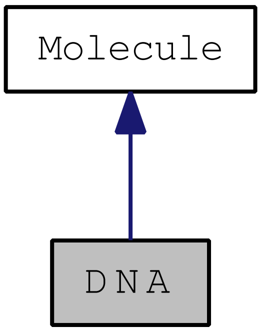
#### 4.5.3.1 MTRand DerivGraph::r

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

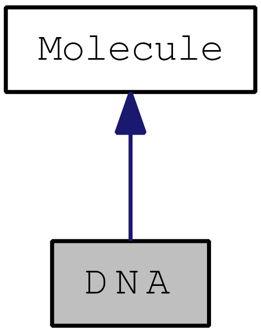
- DerivGraph.h
- DerivGraph.cpp

## 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.

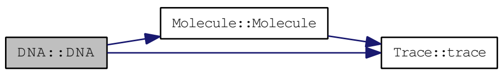
Custom Molecules allow modification of the default behavior of molecules.

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) (Virtual Function)

Returns the next approximate value of this molecule for the next timestep for the specified stage of Runge-Kutta. Runge-Kutta uses successive iterations to make more accurate approximations of a solution.

rkApprox should be used in Interaction::getEffect, to provide the Runge-Kutta corrected concentrations of molecules during runge-kutta calculation instead of the base value for all iterations.

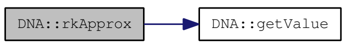
**Parameters:**

***rkIteration*** the current iteration of Runge-Kutta

***rkStepSize*** the timestep being used by Runge-Kutta

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 ncells, int generations, int max_basic, int max_ptm, int max_comp, int max_prom, float min_kinetic_rate, float max_kinetic_rate, float rk_time_limit, float rk_time_step, float initial_-conc)
- ~Experiment ()
- void start ()
- void setOutputOptions (int, int, int, int)

### 4.7.1 Constructor & Destructor Documentation

#### 4.7.1.1 Experiment::Experiment (int *ncells*, int *generations*, int *max_basic*, int *max_ptm*, int *max_comp*, int *max_prom*, float *min_kinetic_rate*, float *max_kinetic_rate*, float *rk_time_limit*, float *rk_time_step*, float *initial_conc*)

Experiment::Experiment(int, int)

Experiment constructor.

**Parameters:**

*ncells* number of Cell objects to be created.

*generations* number of Generations the Experiment will run for.

*max_basic* maximum number of basic proteins allowed in each Cell

*max_ptm* maximum number of PTM proteins allowed in each Cell

*max_comp* maximum number of complexed proteins allowed in each Cell

*max_prom* maximum number of protein-promoter interactions allowed in each cell

*min_kinetic_rate* the lower bound on randomly generated kinetic rates

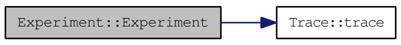
*max_kinetic_rate* the upper bound on randomly generated kinetic rates

*rk_time_limit* the stopping condition for runge-kutta iteration

*rk_time_step* how much time to advance each iteration

*initial_conc* the initial concentration for molecules

Here is the call graph for this function:



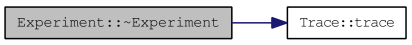
#### 4.7.1.2 Experiment::~Experiment ()

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:

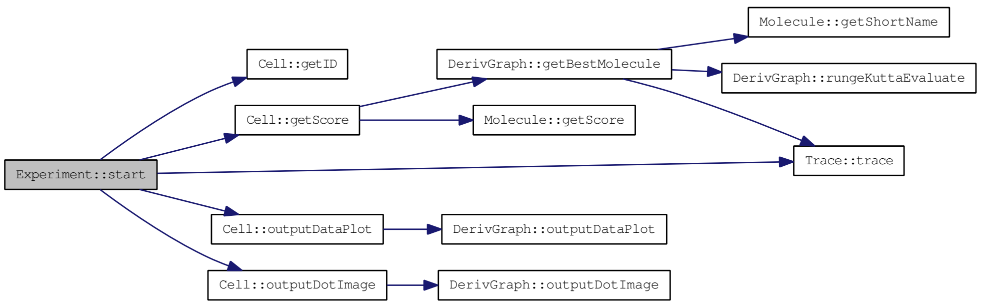


### 4.7.2 Member Function Documentation

#### 4.7.2.1 void Experiment::setOutputOptions (int *gv_flag*, int *gp_flag*, int *eachgen_flag*, int *scoring_interval*)

#### 4.7.2.2 void Experiment::start ()

Here is the call graph for this function:

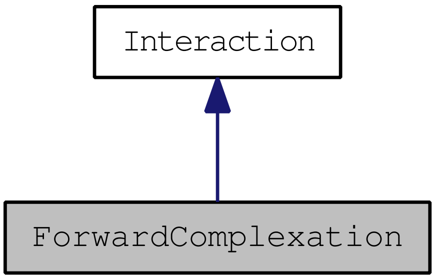


The documentation for this class was generated from the following files:

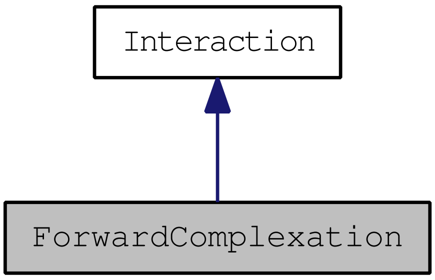
- Experiment.h
- Experiment.cpp

## 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)

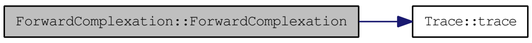
### Public Attributes

- 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.

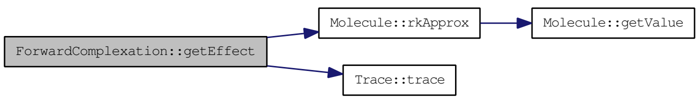
*a* The Node to calculate the effect for

*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

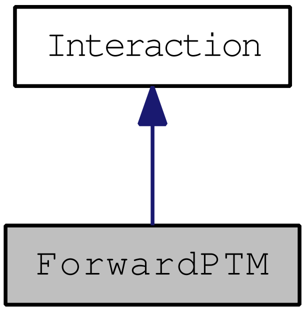
#### 4.8.3.2 int ForwardComplexation::secondNodeID

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

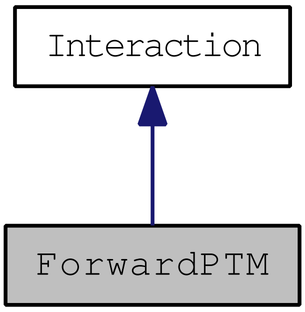
- CustomInteractions.h
- CustomInteractions.cpp

## 4.9 ForwardPTM Class Reference

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



Collaboration diagram for ForwardPTM:



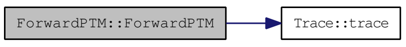
### Public Member Functions

- ForwardPTM ()
- ∼ForwardPTM ()

### 4.9.1 Constructor & Destructor Documentation

#### 4.9.1.1 ForwardPTM::ForwardPTM ()

Here is the call graph for this function:



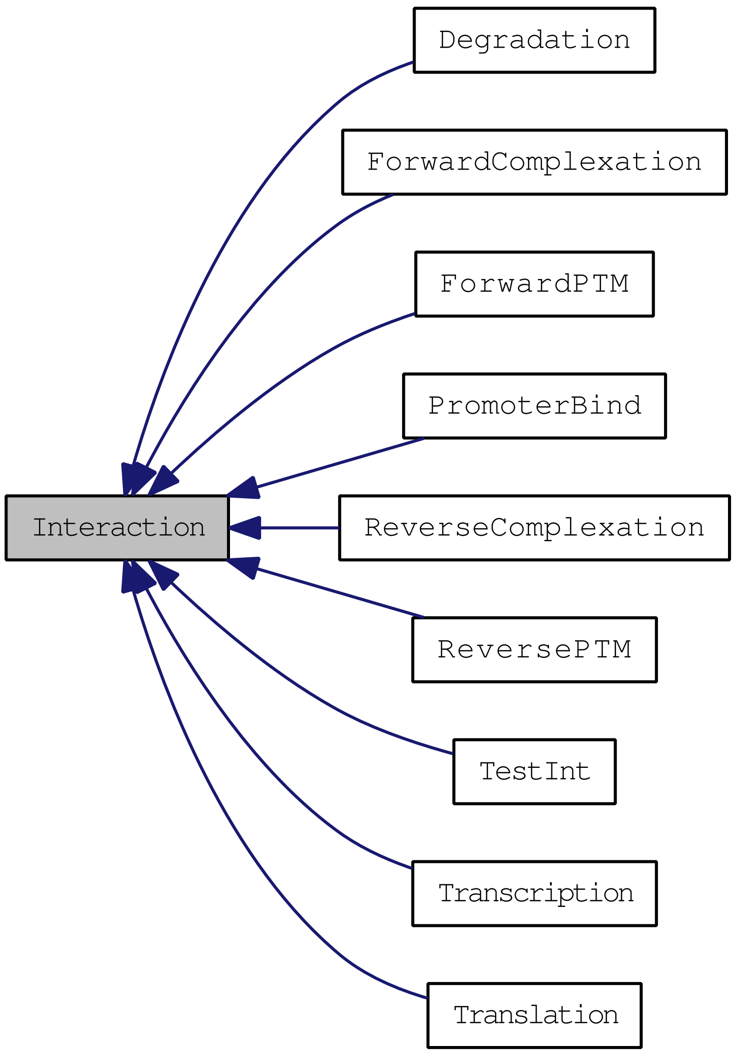
#### 4.9.1.2 ForwardPTM::∼ForwardPTM ()

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

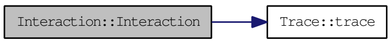
## 4.10.1 Constructor & Destructor Documentation

### 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:

Interaction::~Interaction
Trace::trace

## 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.

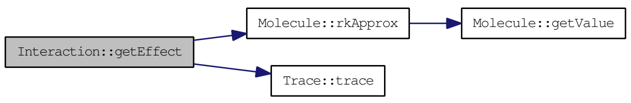
***a*** The Node to calculate the effect for

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

***rkStep*** The stepsize of Runge-Kutta

Reimplemented in TestInt, Transcription, Degradation, Translation, 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

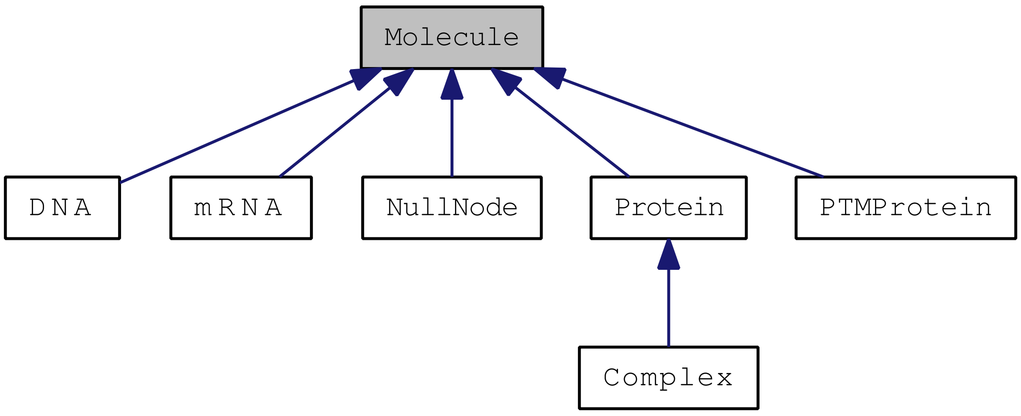
#### 4.10.3.3 float Interaction::rate `[protected]`

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:



### 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

- float initialConcentration
- float currentConcentration
- float rkVal [4]
- int numChanges
- int prevDir
- int currentDir
- char buf [200]
- const char ∗ longName
- const char ∗ shortName
- int moleculeID
- vector< float > rungeKuttaSolution

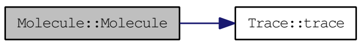
## 4.11.1 Constructor & Destructor Documentation

### 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 ()

int Molecule::getID()

Get the ID of the current molecule.

**Returns:**

the current Molecule's ID

### 4.11.2.2 char ∗ Molecule::getLongName () `[virtual]`

char∗ Molecule::getLongName() (Virtual function)

Return the "long" name of a molecule.

The long name consists of the long prefix set in the constructor appended to the moleculeID with a space in between. Ex. DNA 1, Protein 3, Complex 8

**Returns:**

the long name of the current molecule

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]`

char∗ Molecule::getShortName() (Virtual function)

Return the "short" name of a molecule.

The short name consists of the short prefix set in the constructor appended to the moleculeID with no space in between. Ex. g1, p4, ptm2

**Returns:**

the short name of the current molecule

#### 4.11.2.9 float Molecule::getValue () `[virtual]`

float Molecule::getValue() (Virtual Function)

Get the current value of this molecule.

**Returns:**

the current value of the concentration

Reimplemented in DNA, and NullNode.

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

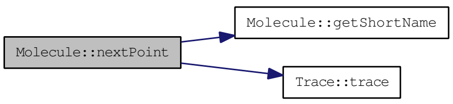
void Molecule::nextPoint(float)

Adds a data point to the rungeKuttaSolution based on the rkVals calculated by Runge-Kutta

**Parameters:**

***step*** The stepsize used to calculate the rkVals

Here is the call graph for this function:

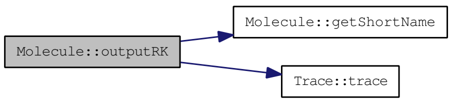


#### 4.11.2.11 void Molecule::outputRK ()

void Molecule::outputRK()

TEST METHOD

Here is the call graph for this function:



#### 4.11.2.12 void Molecule::reset ()

void Molecule::reset()

Reset the molecule between Runge-Kutta runs. The vector containing runge-kutta data points is erased, the initial concentration is added as the first element, and the rkVals are all reset to 0.

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

float Molecule::rkApprox(int, float) (Virtual Function)

Returns the next approximate value of this molecule for the next timestep for the specified stage of Runge-Kutta. Runge-Kutta uses successive iterations to make more accurate approximations of a solution.

rkApprox should be used in Interaction::getEffect, to provide the Runge-Kutta corrected concentrations of molecules during runge-kutta calculation instead of the base value for all iterations.

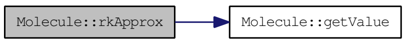
**Parameters:**

***rkIteration*** the current iteration of Runge-Kutta

***rkStepSize*** the timestep being used by Runge-Kutta

Reimplemented in DNA.

Here is the call graph for this function:



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

void Molecule::setID(int)

Set the ID of a molecule, used when displaying molecule names. The ID is a number which is not necessarily unique, but should only be shared between strongly related molecules.

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

void Molecule::setValue(float)

**Parameters:**

*v* the new value to set as the concentration

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

void Molecule::updateRkVal(int, float)

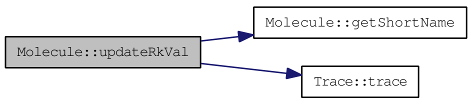
Adds some amount to the specified intermediate values used by Runge-Kutta.

**Parameters:**

*index* The index of the rkValue array to update

*amount* The amount to add to the rkValue array

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 float Molecule::currentConcentration `[protected]`

#### 4.11.3.3 int Molecule::currentDir `[protected]`

#### 4.11.3.4 float Molecule::initialConcentration `[protected]`

#### 4.11.3.5 const char* Molecule::longName `[protected]`

#### 4.11.3.6 int Molecule::moleculeID `[protected]`

#### 4.11.3.7 int Molecule::nodeID

#### 4.11.3.8 int Molecule::numChanges `[protected]`

#### 4.11.3.9 int Molecule::prevDir `[protected]`

#### 4.11.3.10 int Molecule::PTMArray[4]

#### 4.11.3.11 MTRand Molecule::r

#### 4.11.3.12 float Molecule::rkVal[4] `[protected]`

#### 4.11.3.13 vector<float> Molecule::rungeKuttaSolution `[protected]`

#### 4.11.3.14 const char* Molecule::shortName `[protected]`

#### 4.11.3.15 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 ()

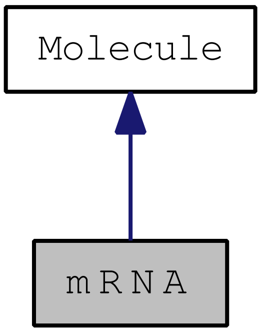
#### 4.12.2.2 MoleculeType::∼MoleculeType ()

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

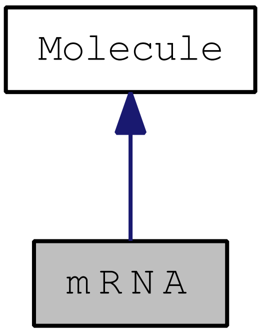
- MoleculeType.h

## 4.13 mRNA Class Reference

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



Collaboration diagram for mRNA:



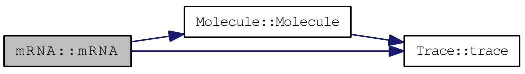
### Public Member Functions

- mRNA ()
- ~mRNA ()

### 4.13.1 Constructor & Destructor Documentation

#### 4.13.1.1 mRNA::mRNA ()

Here is the call graph for this function:



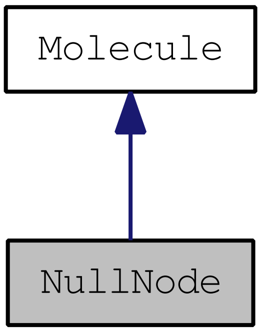
#### 4.13.1.2 mRNA::~mRNA ()

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

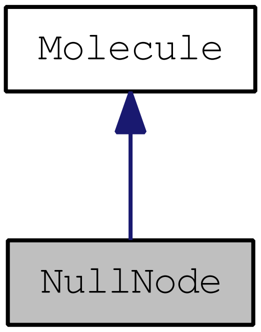
- CustomMolecules.h
- CustomMolecules.cpp

## 4.14 NullNode Class Reference

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



Collaboration diagram for NullNode:



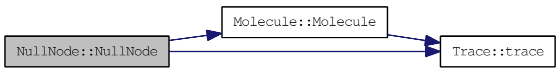
### Public Member Functions

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

### 4.14.1 Constructor & Destructor Documentation

#### 4.14.1.1 NullNode::NullNode ()

Here is the call graph for this function:



#### 4.14.1.2 NullNode::∼NullNode ()

### 4.14.2 Member Function Documentation

#### 4.14.2.1 float NullNode::getValue () `[virtual]`

float Molecule::getValue() (Virtual Function)

Get the current value of this molecule.

**Returns:**

the current value of the concentration

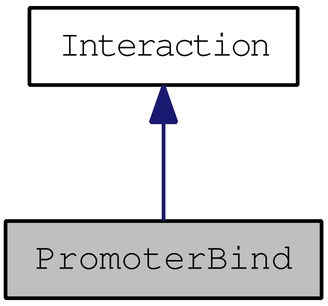
Reimplemented from Molecule.

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

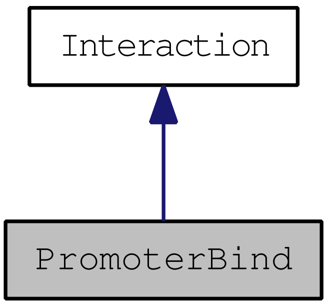
- CustomMolecules.h
- CustomMolecules.cpp

## 4.15 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

### 4.15.1 Constructor & Destructor Documentation

#### 4.15.1.1 PromoterBind::PromoterBind (float *fwdRate*, float *revRate*)

#### 4.15.1.2 PromoterBind::∼PromoterBind ()

### 4.15.2 Member Function Documentation

#### 4.15.2.1 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.

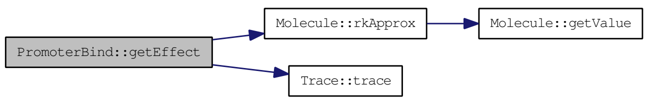
*a* The Node to calculate the effect for

*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:



### 4.15.3 Member Data Documentation

#### 4.15.3.1 float PromoterBind::kf

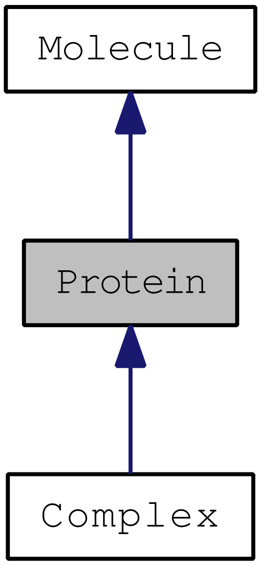
#### 4.15.3.2 float PromoterBind::kr

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

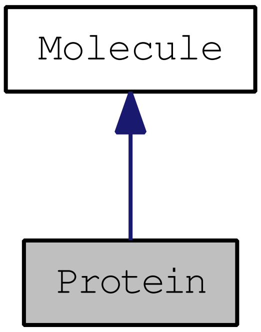
- CustomInteractions.h
- CustomInteractions.cpp

## 4.16 Protein Class Reference

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



Collaboration diagram for Protein:



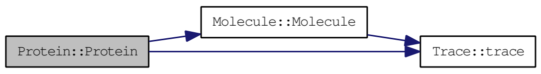
### Public Member Functions

- Protein ()
- ∼Protein ()

### 4.16.1 Constructor & Destructor Documentation

#### 4.16.1.1 Protein::Protein ()

Here is the call graph for this function:



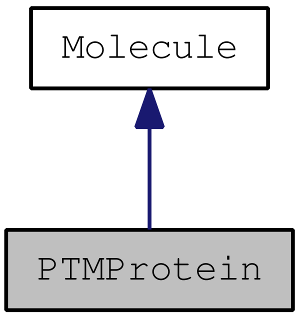
#### 4.16.1.2 Protein::∼Protein ()

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

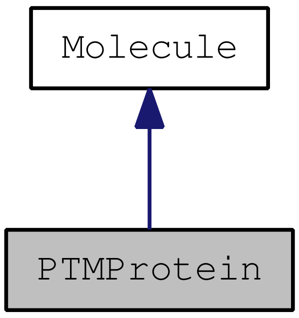
- CustomMolecules.h
- CustomMolecules.cpp

## 4.17 PTMProtein Class Reference

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



Collaboration diagram for PTMProtein:



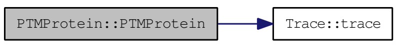
### Public Member Functions

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

### 4.17.1 Constructor & Destructor Documentation

#### 4.17.1.1 PTMProtein::PTMProtein ()

Here is the call graph for this function:



#### 4.17.1.2 PTMProtein::PTMProtein (PTMProtein ∗ *c*)

#### 4.17.1.3 PTMProtein::∼PTMProtein ()

### 4.17.2 Member Function Documentation

#### 4.17.2.1 void PTMProtein::addRandPTM (int *i*)

#### 4.17.2.2 char ∗ PTMProtein::getLongName () `[virtual]`

char∗ Molecule::getLongName() (Virtual function)

Return the "long" name of a molecule.

The long name consists of the long prefix set in the constructor appended to the moleculeID with a space in between. Ex. DNA 1, Protein 3, Complex 8

**Returns:**

the long name of the current molecule

Reimplemented from Molecule.

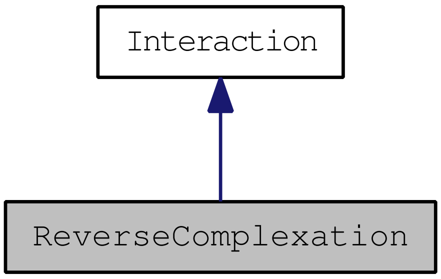
#### 4.17.2.3 void PTMProtein::setPTMCount (int *index*, int *count*)

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

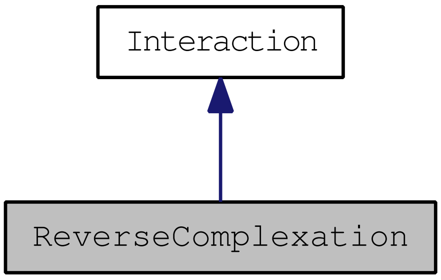
- CustomMolecules.h
- CustomMolecules.cpp

## 4.18 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)

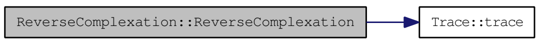
### Public Attributes

- int firstNodeID
- int secondNodeID

### 4.18.1 Constructor & Destructor Documentation

#### 4.18.1.1 ReverseComplexation::ReverseComplexation (int *n1*, int *n2*)

Here is the call graph for this function:



#### 4.18.1.2 ReverseComplexation::∼ReverseComplexation ()

## 4.18.2 Member Function Documentation

#### 4.18.2.1 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.

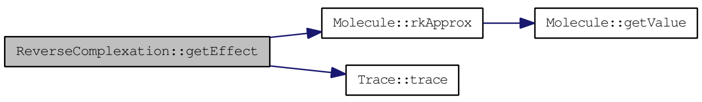
Runge-Kutta iteratively approximates the change in concentration during a given timestep. The first iteration is based soley on the current concentration, and each further iteration takes the result of the previous iteration into account. The Runge-Kutta data are stored in each molecule, and it is necessary 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:



## 4.18.3 Member Data Documentation

#### 4.18.3.1 int ReverseComplexation::firstNodeID

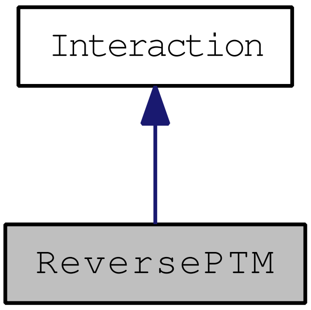
#### 4.18.3.2 int ReverseComplexation::secondNodeID

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

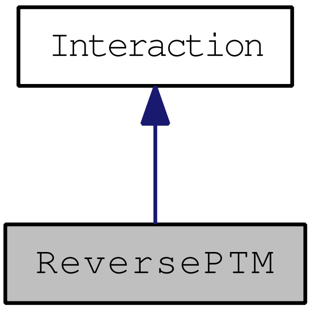
- CustomInteractions.h
- CustomInteractions.cpp

## 4.19 ReversePTM Class Reference

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



Collaboration diagram for ReversePTM:



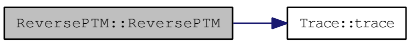
### Public Member Functions

- ReversePTM ()
- ∼ReversePTM ()

### 4.19.1 Constructor & Destructor Documentation

#### 4.19.1.1 ReversePTM::ReversePTM ()

Here is the call graph for this function:



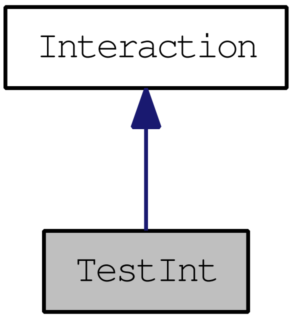
#### 4.19.1.2 ReversePTM::∼ReversePTM ()

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

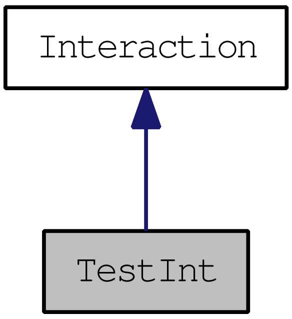
- CustomInteractions.h
- CustomInteractions.cpp

## 4.20 TestInt Class Reference

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



Collaboration diagram for TestInt:



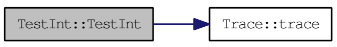
### Public Member Functions

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

### 4.20.1 Constructor & Destructor Documentation

#### 4.20.1.1 TestInt::TestInt ()

Here is the call graph for this function:



#### 4.20.1.2 TestInt::∼TestInt ()

### 4.20.2 Member Function Documentation

#### 4.20.2.1 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.

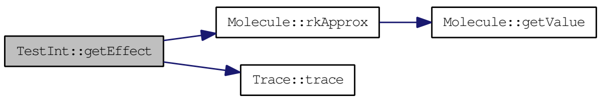
*a* The Node to calculate the effect for

*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

## 4.21 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

### 4.21.1 Constructor & Destructor Documentation

#### 4.21.1.1 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.

#### 4.21.1.2 Trace::Trace (const char ∗ *c*)

#### 4.21.1.3 Trace::∼Trace ()

Trace::∼Trace()

Default Destructor.

### 4.21.2 Member Function Documentation

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

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

Adds a new trace tag and enables it.

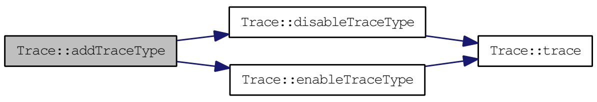
e.g. Trace::addTraceType("output") //output related t.trace("output", "Output complete");

**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:



### 4.21.2.2 void Trace::disableTraceType (const char ∗ *tag*)

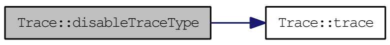
Trace::disableTraceType(const char∗)

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:



### 4.21.2.3 void Trace::enableTraceType (const char ∗ *tag*)

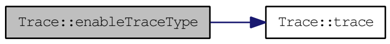
Trace::enableTraceType(const char∗)

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:



### 4.21.2.4 FILE ∗ Trace::getTraceFile ()

### 4.21.2.5 FILE ∗ Trace::setTraceFile (FILE ∗ *tf*)

### 4.21.2.6 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 |

## 4.21.3 Member Data Documentation

#### 4.21.3.1 FILE∗ Trace::traceFile

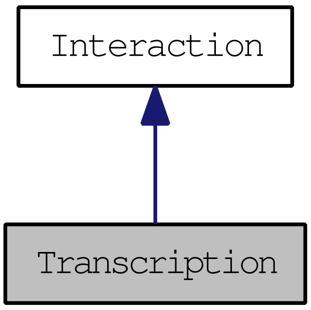
#### 4.21.3.2 map<const char∗, int, cmp_str> Trace::traceTypes

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

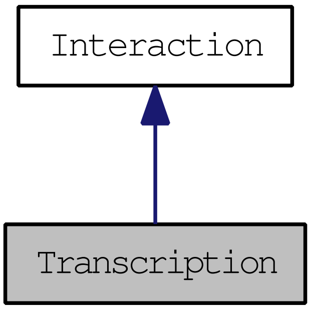
- Trace.h
- Trace.cpp

## 4.22 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)

### 4.22.1 Constructor & Destructor Documentation

#### 4.22.1.1 Transcription::Transcription ()

Implementation file for Custom Interactions.

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

#### 4.22.1.2 Transcription::∼Transcription ()

## 4.22.2 Member Function Documentation

#### 4.22.2.1 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.

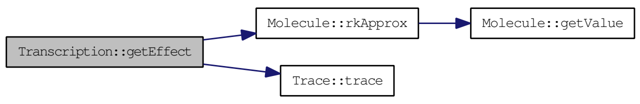
***a*** The Node to calculate the effect for

***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:

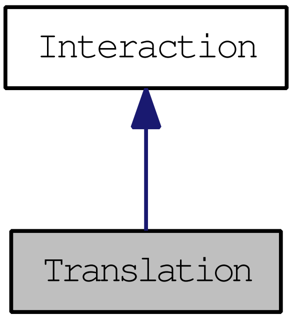


The documentation for this class was generated from the following files:

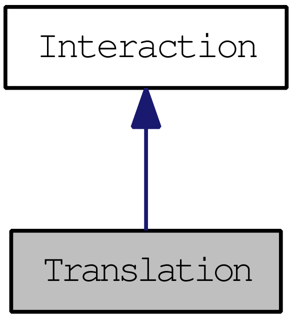
- CustomInteractions.h
- CustomInteractions.cpp

## 4.23 Translation Class Reference

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



Collaboration diagram for Translation:



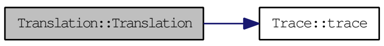
### Public Member Functions

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

### 4.23.1 Constructor & Destructor Documentation

#### 4.23.1.1 Translation::Translation ()

Here is the call graph for this function:



#### 4.23.1.2 Translation::∼Translation ()

## 4.23.2 Member Function Documentation

#### 4.23.2.1 float Translation::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.

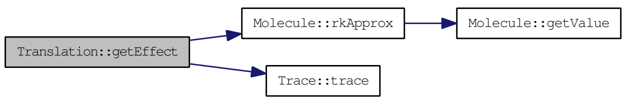
*a* The Node to calculate the effect for

*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

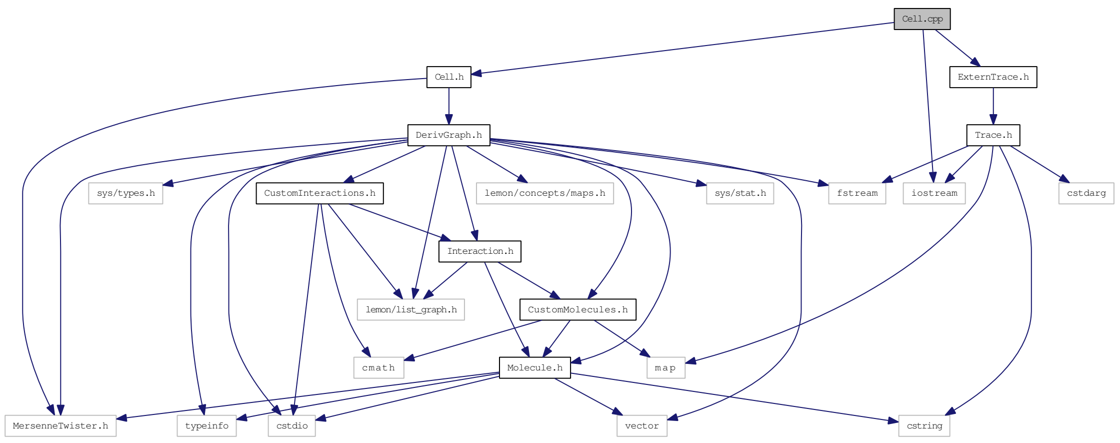
# Chapter 5

# File Documentation

## 5.1 Cell.cpp File Reference

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

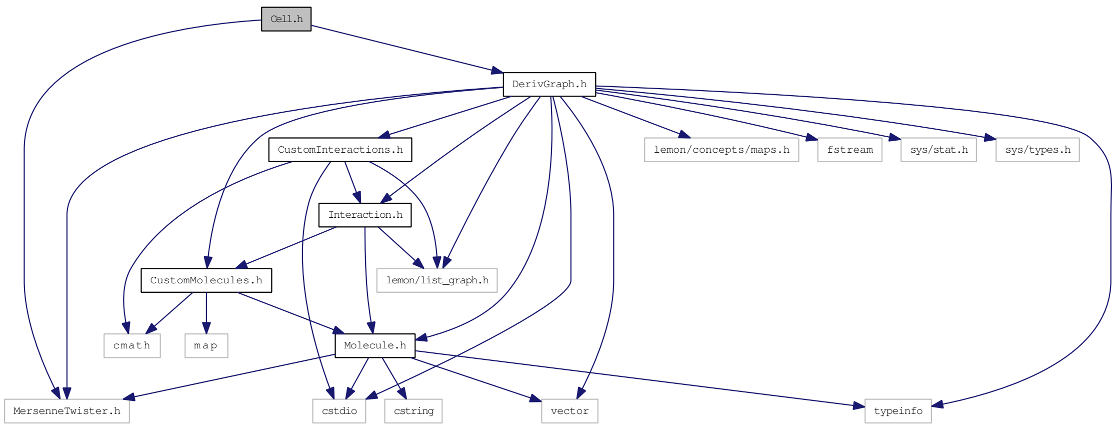
Include dependency graph for Cell.cpp:



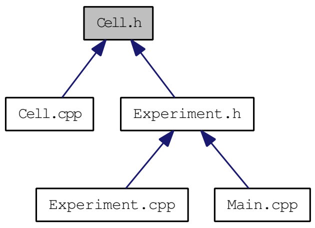
## 5.2 Cell.h File Reference

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

Include dependency graph for Cell.h:



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



### Classes

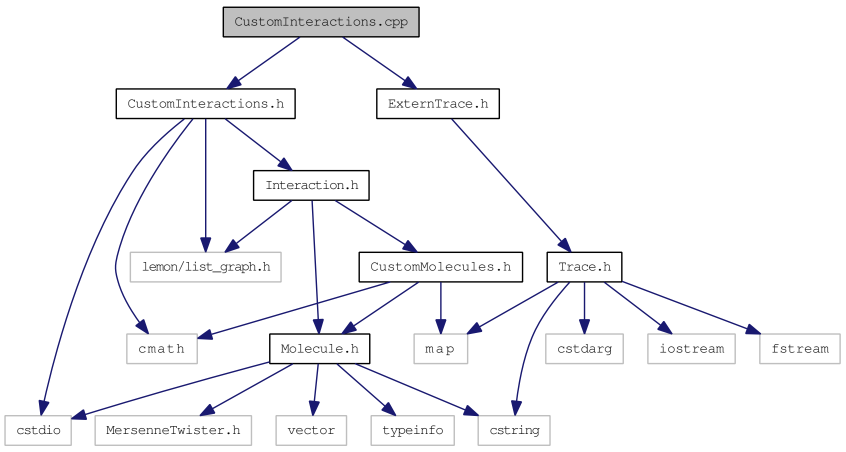
- class Cell

## 5.3 CustomInteractions.cpp File Reference

`#include "CustomInteractions.h"`

`#include "ExternTrace.h"`

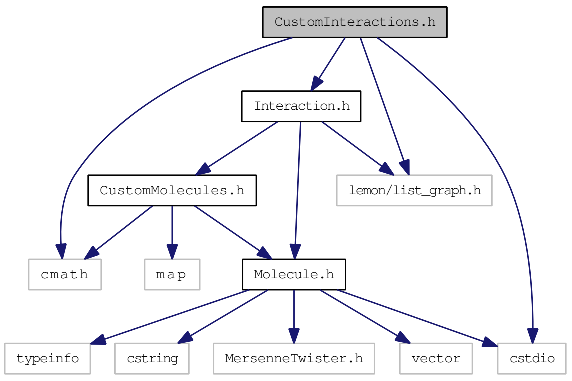
Include dependency graph for CustomInteractions.cpp:



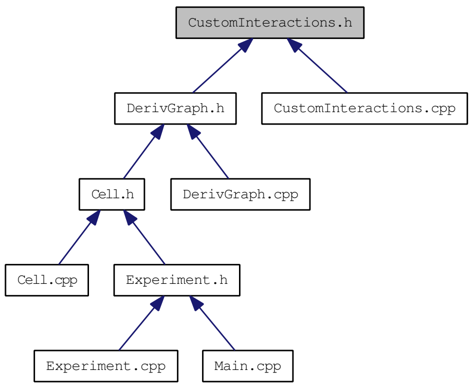
## 5.4 CustomInteractions.h File Reference

```
#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:



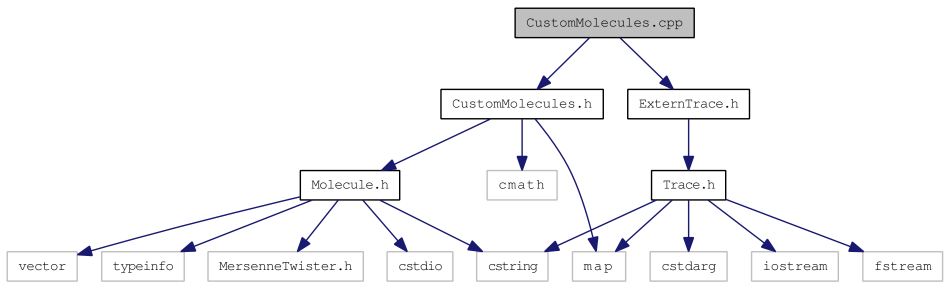
### Classes

- 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"
```

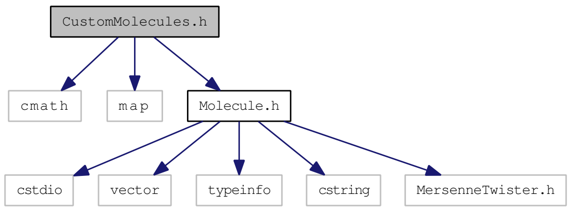
Include dependency graph for CustomMolecules.cpp:



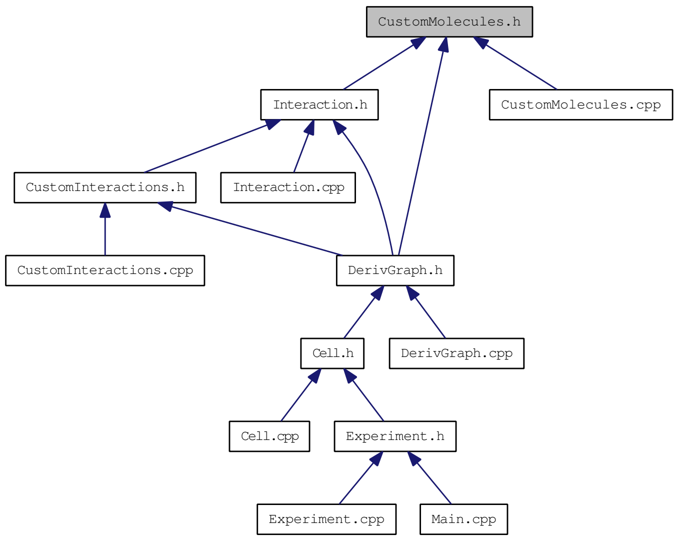
## 5.6 CustomMolecules.h File Reference

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

Include dependency graph for CustomMolecules.h:



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



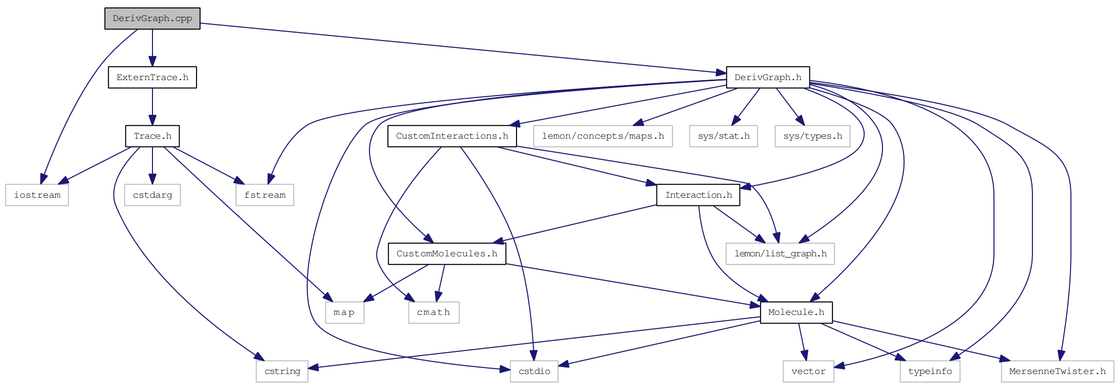
### Classes

- 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"
```

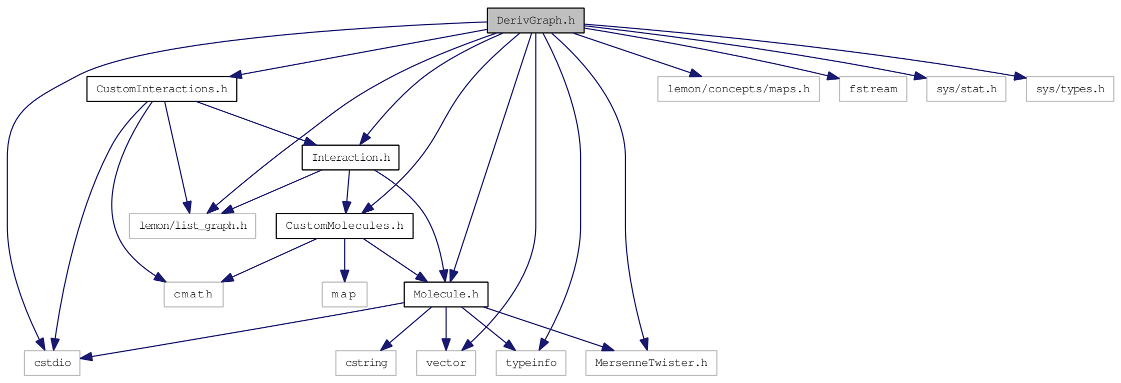
Include dependency graph for DerivGraph.cpp:



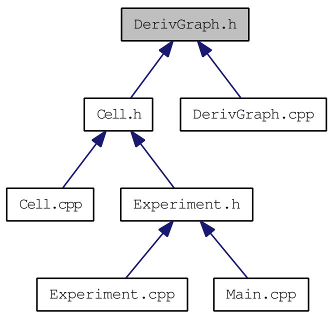
## 5.8 DerivGraph.h File Reference

```
#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:



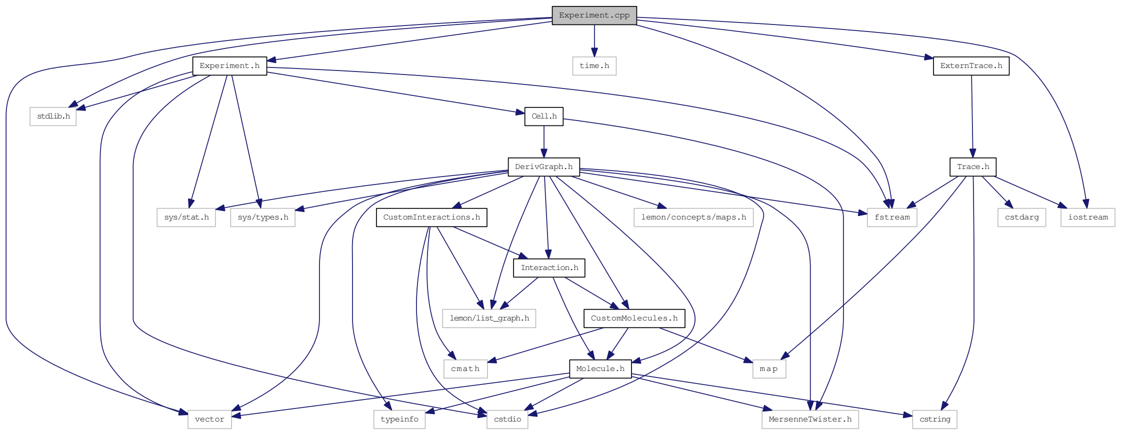
## Classes

- class DerivGraph

## 5.9 Experiment.cpp File Reference

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

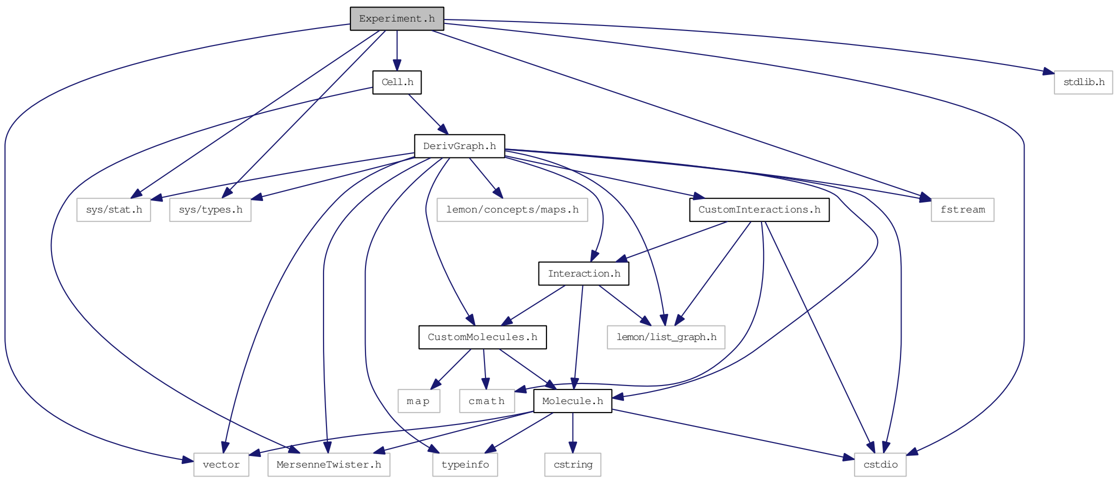
Include dependency graph for Experiment.cpp:



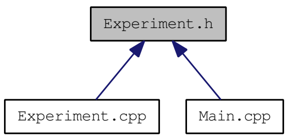
## 5.10 Experiment.h File Reference

```
#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:



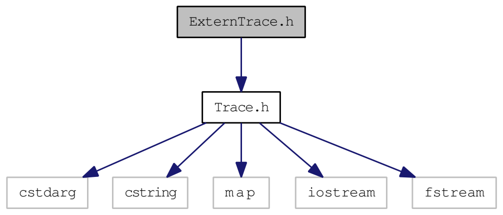
## Classes

- class Experiment

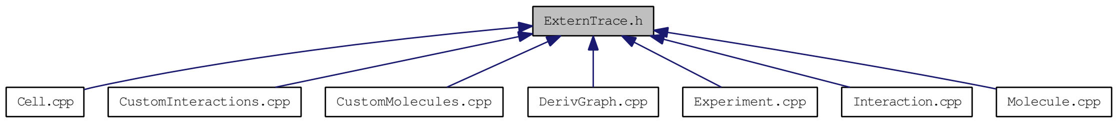
## 5.11 ExternTrace.h File Reference

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

Include dependency graph for ExternTrace.h:



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



### Variables

- Trace t

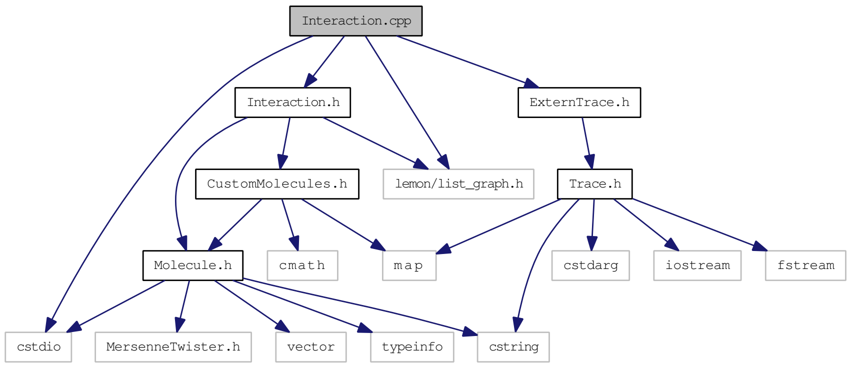
### 5.11.1 Variable Documentation

#### 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"
```

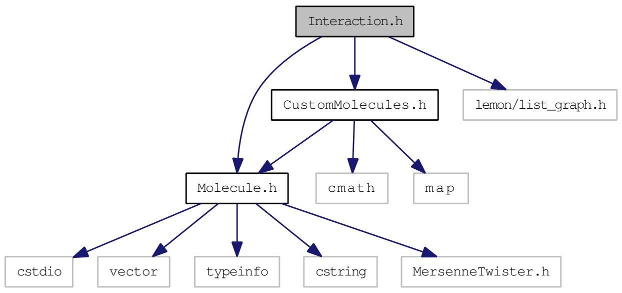
Include dependency graph for Interaction.cpp:



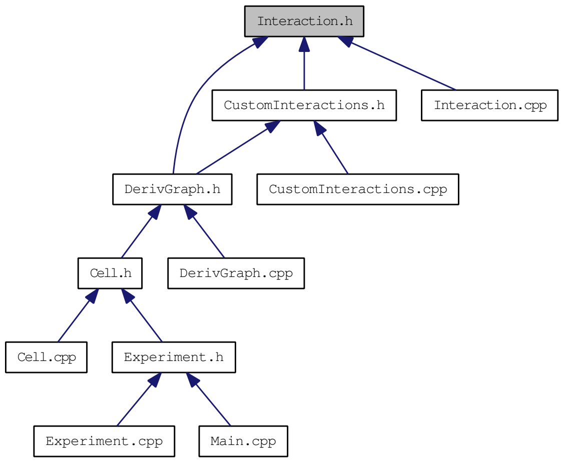
## 5.13 Interaction.h File Reference

```
#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:



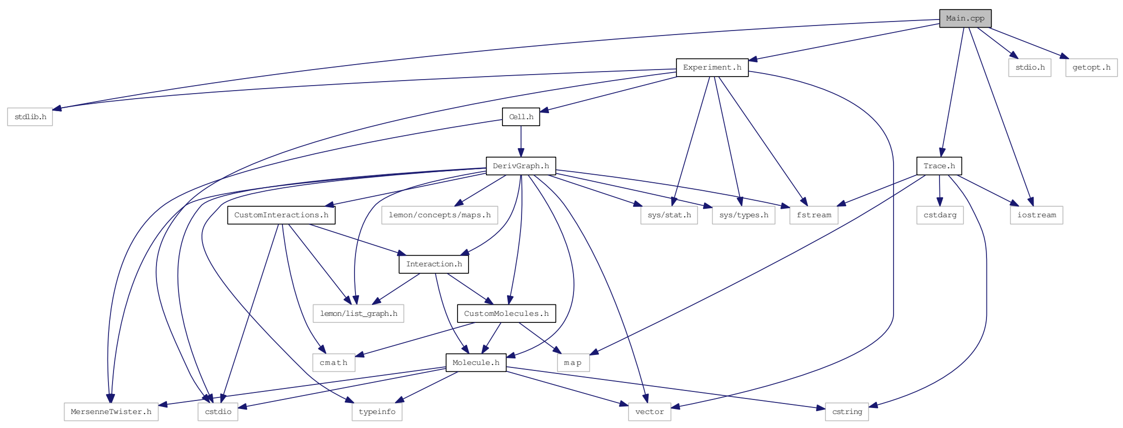
### Classes

- 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)

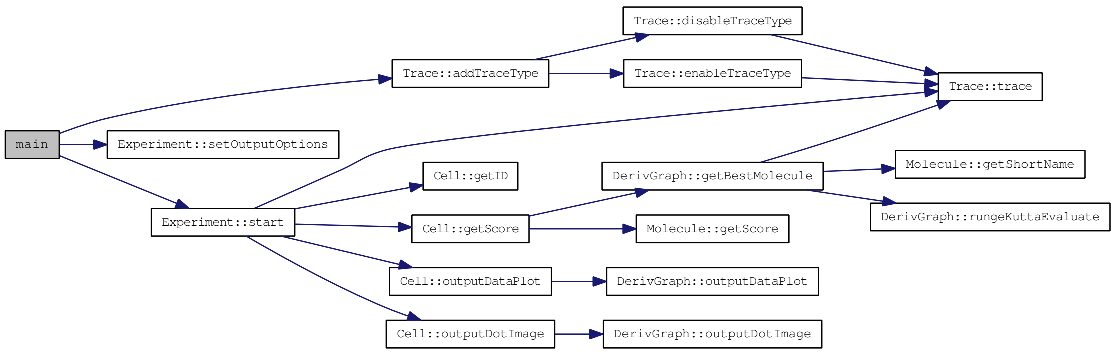
### Variables

- 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:



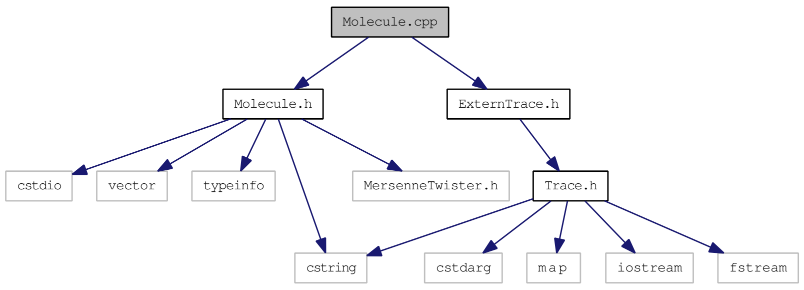
## 5.14.2 Variable Documentation

### 5.14.2.1 Trace t

## 5.15 Molecule.cpp File Reference

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

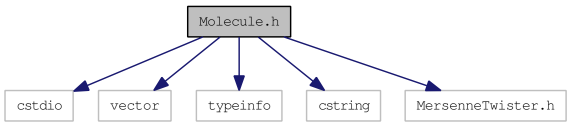
Include dependency graph for Molecule.cpp:



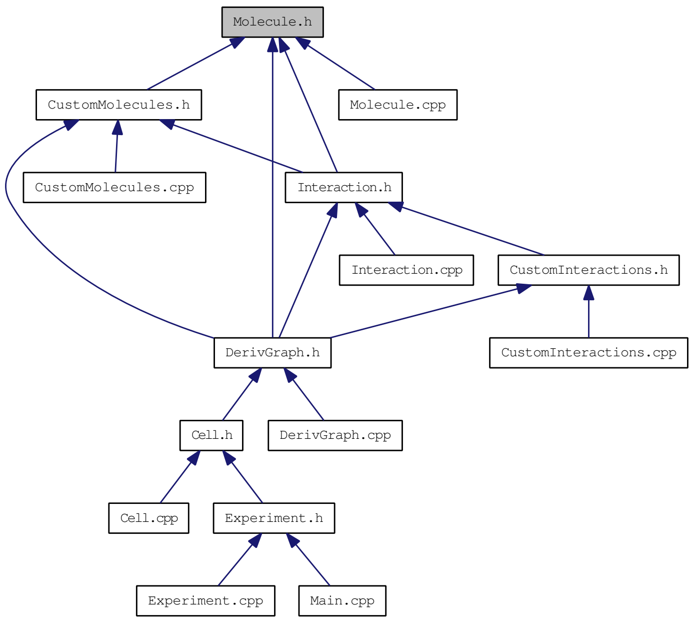
## 5.16 Molecule.h File Reference

```
#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

## 5.17 MoleculeType.h File Reference

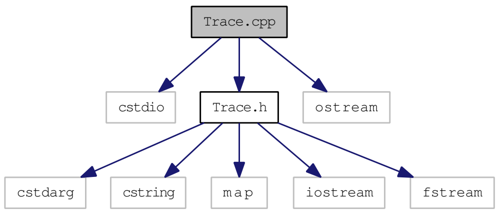
### Classes

- class MoleculeType

## 5.18 Trace.cpp File Reference

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

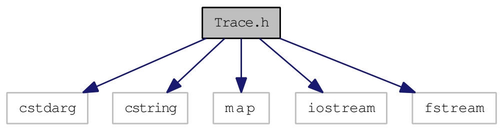
Include dependency graph for Trace.cpp:



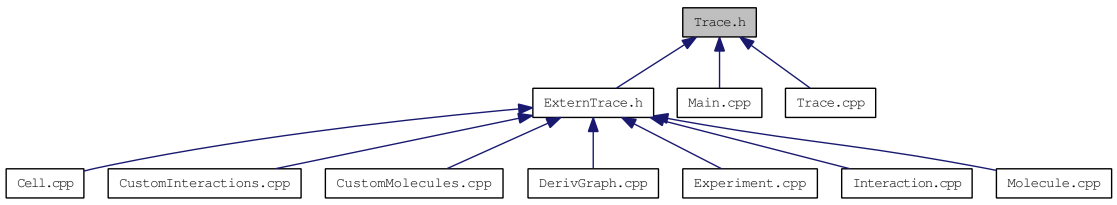
## 5.19 Trace.h File Reference

```
#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