

Production Networks

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Company Class Reference	5
3.1.1 Detailed Description	6
3.2 Good Class Reference	6
3.2.1 Detailed Description	6
3.3 ModelBuilder Class Reference	7
3.3.1 Detailed Description	7
3.3.2 Member Function Documentation	7
3.3.2.1 create_company()	7
3.3.2.2 create_good()	8
3.3.2.3 create_plant()	8
3.3.2.4 create_plant_randomly()	8
3.3.2.5 create_process()	9
3.3.2.6 create_process_randomly()	9
3.4 NetworkGenerator Class Reference	10
3.4.1 Member Function Documentation	10
3.4.1.1 generate_random_production_data()	10
3.5 Plant Class Reference	11
3.5.1 Detailed Description	12
3.5.2 Member Function Documentation	12
3.5.2.1 add_production_process()	12
3.5.2.2 get_production()	12
3.5.2.3 inc_production_capacity()	13
3.5.2.4 inc_production_quantity()	13
3.5.2.5 set_production_capacity()	13
3.5.2.6 set_production_quantity()	13
3.6 Process Class Reference	14
3.6.1 Detailed Description	15
3.6.2 Member Function Documentation	15
3.6.2.1 add_input()	15
3.6.2.2 get_quantity()	15
3.7 ProductionData Class Reference	15
3.7.1 Detailed Description	16
3.8 ProductionNetwork Class Reference	16
3.8.1 Member Typedef Documentation	17
3.8.1.1 ChangedPriceData	17
3.8.2 Member Function Documentation	17

3.8.2.1 change_price()	17
3.8.2.2 to_dot()	18
3.9 Rng Class Reference	18
3.9.1 Detailed Description	19
3.9.2 Member Function Documentation	19
3.9.2.1 flip_coin()	19
3.9.2.2 random_choice()	19
3.9.2.3 random_exponential()	20
3.9.2.4 random_normal()	20
3.9.2.5 random_rif()	21
3.9.2.6 random_uniform_int()	21
3.9.2.7 random_uniform_real()	21
3.10 TestResults Struct Reference	22
4 File Documentation	23
4.1 student_distribution/lib/include/Generator.hpp File Reference	23
4.1.1 Detailed Description	23
4.2 student_distribution/lib/include/IOUtils.hpp File Reference	23
4.2.1 Detailed Description	24
4.3 student_distribution/lib/include/Models.hpp File Reference	25
4.3.1 Detailed Description	26
4.4 student_distribution/lib/include/ProductionNetwork.hpp File Reference	26
4.4.1 Detailed Description	26
4.5 student_distribution/lib/include/Rng.hpp File Reference	27
4.5.1 Detailed Description	27

Chapter 1

Class Index

1.1 Class List

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

Company		
	Model for a company	5
Good		
	Model for goods or services	6
ModelBuilder	7
NetworkGenerator	10
Plant	11
Process		
	Represents a formula to generate a good	14
ProductionData	15
ProductionNetwork	16
Rng		
	Singleton for random number generation	18
TestResults	22

Chapter 2

File Index

2.1 File List

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

student_distribution/lib/include/ Generator.hpp	23
student_distribution/lib/include/ IOUtils.hpp	23
student_distribution/lib/include/ Models.hpp	25
student_distribution/lib/include/ ProductionNetwork.hpp	26
student_distribution/lib/include/ Rng.hpp	27
student_distribution/lib/include/ TestUtils.hpp	??

Chapter 3

Class Documentation

3.1 Company Class Reference

Model for a company.

```
#include <Models.hpp>
```

Public Member Functions

- `Company ()` noexcept
Default constructor. Don't use it.
- `Company (const std::string &r)` noexcept
Constructor with a R.I.F as l-value.
- `Company (std::string &&r)` noexcept
Constructor with a R.I.F as l-value.
- `Company (const Company &c)` noexcept
Copy constructor.
- `Company (Company &&c)` noexcept
Move constructor.
- `void swap (Company &c)` noexcept
Swap elements.
- `Company & operator= (Company c)` noexcept
Assignment operator.
- `const std::string & get_rif ()` const noexcept
Gets the R.I.F.
- `const std::string & get_name ()` const noexcept
Gets the name.
- `void set_name (const std::string &n)` noexcept
Sets name as l-value.
- `void set_name (std::string &&n)` noexcept
Sets name as r-value.
- `void add_plant (std::shared_ptr< Plant > p)` noexcept
Adds the plant p.
- `std::vector< std::shared_ptr< Plant > > & get_plants ()` noexcept
Returns the list of plants.

3.1.1 Detailed Description

Model for a company.

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

- [student_distribution/lib/include/Models.hpp](#)

3.2 Good Class Reference

Model for goods or services.

```
#include <Models.hpp>
```

Public Member Functions

- [Good](#) () noexcept
Default constructor. Don't use it.
- [Good](#) (const std::string &d) noexcept
Constructor that receives a description as l-value.
- [Good](#) (std::string &&d) noexcept
Constructor that receives a description as r-value.
- [Good](#) (const [Good](#) &g) noexcept
Copy constructor.
- [Good](#) ([Good](#) &&g) noexcept
Move constructor.
- void [swap](#) ([Good](#) &g) noexcept
Swap elements.
- [Good](#) & [operator=](#) ([Good](#) g) noexcept
Assignment operator.
- const std::string & [get_description](#) () const noexcept
Gets the description.
- void [add_producer_process](#) (std::shared_ptr< [Process](#) > p) noexcept
Add a process as a producer of this good.
- void [add_consumer_process](#) (std::shared_ptr< [Process](#) > p) noexcept
Add a process as a consumer of this good.
- std::vector< std::shared_ptr< [Process](#) > > & [get_producer_processes](#) () noexcept
Returns the list of producer processes.
- std::vector< std::shared_ptr< [Process](#) > > & [get_consumer_processes](#) () noexcept
Returns the list of consumer processes.

3.2.1 Detailed Description

Model for goods or services.

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

- [student_distribution/lib/include/Models.hpp](#)

3.3 ModelBuilder Class Reference

```
#include <Models.hpp>
```

Public Member Functions

- `std::shared_ptr< Good > create_good` (const std::string &description) noexcept
Creates a new good.
- `std::shared_ptr< Process > create_process` (std::shared_ptr< [Good](#) > output, std::vector< std::tuple< std::shared_ptr< [Good](#) >, float >> &inputs) noexcept
Creates a process.
- `std::shared_ptr< Process > create_process_randomly` (std::shared_ptr< [Good](#) > output, std::unordered_set< std::shared_ptr< [Good](#) >> &possible_inputs) noexcept
Creates a process with random inputs.
- `std::shared_ptr< Plant > create_plant` (std::shared_ptr< [Company](#) > owner, std::vector< std::tuple< std::shared_ptr< [Process](#) >, float, float >> &production) noexcept
Creates a plant.
- `std::shared_ptr< Plant > create_plant_randomly` (std::shared_ptr< [Company](#) > owner, size_t num_processes, std::vector< std::shared_ptr< [Process](#) >> &process_set) noexcept
Creates a plant with random processes.
- `std::shared_ptr< Company > create_company` (const std::string &rif) noexcept
Creates a company.

3.3.1 Detailed Description

This class is intended to wrap the complex construction of some of the objects by granting that they are constructed with the right references in both directions.

3.3.2 Member Function Documentation

3.3.2.1 create_company()

```
std::shared_ptr<Company> ModelBuilder::create_company (
    const std::string & rif ) [noexcept]
```

Creates a company.

Parameters

<i>rif</i>	The R.I.F.
------------	------------

Returns

A shared pointer to the company.

3.3.2.2 create_good()

```
std::shared_ptr<Good> ModelBuilder::create_good (
    const std::string & description ) [noexcept]
```

Creates a new good.

Parameters

<i>description</i>	The description (name) of the good.
--------------------	-------------------------------------

Returns

a shared pointer to the good.

3.3.2.3 create_plant()

```
std::shared_ptr<Plant> ModelBuilder::create_plant (
    std::shared_ptr< Company > owner,
    std::vector< std::tuple< std::shared_ptr< Process >, float, float >> & production
) [noexcept]
```

Creates a plant.

Parameters

<i>owner</i>	The company to which the plant belongs.
<i>production</i>	A list of tuples (process, produced amount, used capacity) with all of the production processes.

Returns

a shared pointer to the plant.

3.3.2.4 create_plant_randomly()

```
std::shared_ptr<Plant> ModelBuilder::create_plant_randomly (
    std::shared_ptr< Company > owner,
    size_t num_processes,
    std::vector< std::shared_ptr< Process >> & process_set ) [noexcept]
```

Creates a plant with random processes.

Parameters

<i>owner</i>	The company to which the plant belongs.
<i>num_processes</i>	The maximum amount of processes that the plant will have.
<i>process_set</i>	A set of available processes.

Returns

a shared pointer to the plant.

3.3.2.5 create_process()

```
std::shared_ptr<Process> ModelBuilder::create_process (
    std::shared_ptr< Good > output,
    std::vector< std::tuple< std::shared_ptr< Good >, float >> & inputs ) [noexcept]
```

Creates a process.

Parameters

<i>output</i>	The produced good.
<i>inputs</i>	A list of tuples (good, amount) with the inputs.

Returns

a shared pointer to the process.

3.3.2.6 create_process_randomly()

```
std::shared_ptr<Process> ModelBuilder::create_process_randomly (
    std::shared_ptr< Good > output,
    std::unordered_set< std::shared_ptr< Good >> & possible_inputs ) [noexcept]
```

Creates a process with random inputs.

This method creates a process by selecting inputs and their required amounts randomly.

Parameters

<i>output</i>	The good that the process produces.
<i>possible_inputs</i>	A set with the possible inputs of the process.

Returns

a shared pointer to the process.

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

- [student_distribution/lib/include/Models.hpp](#)

3.4 NetworkGenerator Class Reference

Static Public Member Functions

- static [ProductionData](#) [generate_random_production_data](#) (unsigned int num_levels, unsigned int num_plants_per_level, unsigned int num_goods_per_plant, unsigned int num_inputs_per_process, float probability_to_repeat_good, float probability_to_repeat_company)

Generates a random production network.

3.4.1 Member Function Documentation

3.4.1.1 generate_random_production_data()

```
static ProductionData NetworkGenerator::generate_random_production_data (
    unsigned int num_levels,
    unsigned int num_plants_per_level,
    unsigned int num_goods_per_plant,
    unsigned int num_inputs_per_process,
    float probability_to_repeat_good,
    float probability_to_repeat_company ) [static]
```

Generates a random production network.

This function generates a very idealist-random-production network. The features that this function provides are:

- It is structured by production levels.
- Each plant is only in one of the levels (although many plants might belong to the same company).
- Each good is produced in an only level too.
- Each good has a unique producer process.
- The inputs of each process are produced in any of the previous levels.
- The processes in the first level do not have inputs.
- The production amounts in the last level were generated randomly.
- The production amounts of the non-last-level-goods are computed to satisfy the required production.
- The production prices in the first level were generated randomly.
- The production prices of the non-first-level-goods are computed according to the input prices.

Parameters

<i>num_levels</i>	The number of production levels in the network.
<i>num_plants_per_level</i>	Expected number of plants by level.
<i>num_goods_per_plant</i>	Expected number of goods produced by a plant.
<i>num_inputs_per_process</i>	Expected number of inputs for a process.
<i>probability_to_repeat_good</i>	Probability of many plants producing the same good.
<i>probability_to_repeat_company</i>	Probability of many plants belonging to the same company.

Returns

[ProductionData](#)

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

- [student_distribution/lib/include/Generator.hpp](#)

3.5 Plant Class Reference

```
#include <Models.hpp>
```

Public Member Functions

- [Plant](#) () noexcept
Default constructor. Don't use it.
- [Plant](#) (std::shared_ptr< [Company](#) > ow) noexcept
Constructor with the owner company.
- [Plant](#) (const [Plant](#) &p) noexcept
Copy constructor.
- [Plant](#) ([Plant](#) &&p) noexcept
Move constructor.
- void [swap](#) ([Plant](#) &p) noexcept
Swap elements.
- [Plant](#) & [operator=](#) ([Plant](#) p) noexcept
Assignment operator.
- std::shared_ptr< [Company](#) > [get_owner](#) () const noexcept
Gets the owner company.
- void [add_production_process](#) (std::shared_ptr< [Process](#) > p, float q, float cap)
- void [set_production_quantity](#) (std::shared_ptr< [Process](#) > p, float q)
- void [inc_production_quantity](#) (std::shared_ptr< [Process](#) > p, float q)
- void [set_production_capacity](#) (std::shared_ptr< [Process](#) > p, float c)
- void [inc_production_capacity](#) (std::shared_ptr< [Process](#) > p, float c)
- std::tuple< float, float > [get_production](#) (std::shared_ptr< [Process](#) > p) const
- std::unordered_map< std::shared_ptr< [Process](#) >, std::pair< float, float > > & [get_production_table](#) () noexcept
Gets the table of production.
- std::vector< std::tuple< std::shared_ptr< [Process](#) >, float, float > > [get_production](#) () const noexcept
Gets a list of tuples (process, quantity, capacity) de los procesos.

3.5.1 Detailed Description

Represents a plant or a company branch office that produces goods.

The goods that produces are defined by each process that the plant owns.

In this model, the production of the goods is represented by a table of processes, each of them associated with the actual production and its capacity of production represented as a percentage.

For instance: Lets P1 be a process owned by a plant. Suppose that P1 has a capacity of production of 100 units by time unit and in a time point it is producing 80 units, then P1 will be associated with the pair (80, 80). That means that P1 is producing 80 units of its output and it is using the 80% of its capacity of production.

3.5.2 Member Function Documentation

3.5.2.1 add_production_process()

```
void Plant::add_production_process (
    std::shared_ptr< Process > p,
    float q,
    float cap )
```

Adds a process with its actual production and its capacity of production.

If the process already exists, then the production and the capacity will be updated.

Exceptions

<code>std::invalid_argument</code>	if the quantity or the capacity are less than zero or the capacity is greater than 100.
------------------------------------	---

3.5.2.2 get_production()

```
std::tuple<float, float> Plant::get_production (
    std::shared_ptr< Process > p ) const
```

Gets a tuple (q, c) with the quantity and capacity of production of a given process.

Exceptions

<code>std::domain_error</code>	if the process does not exist.
--------------------------------	--------------------------------

3.5.2.3 inc_production_capacity()

```
void Plant::inc_production_capacity (
    std::shared_ptr< Process > p,
    float c )
```

Increments the capacity of production of a given process. A negative value will decrement the capacity of production. The value will be truncated to be in [0, 100].

Exceptions

<i>std::domain_error</i>	if the process does not exist.
--------------------------	--------------------------------

3.5.2.4 inc_production_quantity()

```
void Plant::inc_production_quantity (
    std::shared_ptr< Process > p,
    float q )
```

Increments the quantity of production of a given process. A negative value will decrement the quantity of production. The value will be truncated to zero.

Exceptions

<i>std::domain_error</i>	if the process does not exist.
--------------------------	--------------------------------

3.5.2.5 set_production_capacity()

```
void Plant::set_production_capacity (
    std::shared_ptr< Process > p,
    float c )
```

Sets a new capacity of production to a process.

Exceptions

<i>std::domain_error</i>	if the process does not exist.
<i>std::invalid_argument</i>	if the quantity is not in [0, 100].

3.5.2.6 set_production_quantity()

```
void Plant::set_production_quantity (
```

```
std::shared_ptr< Process > p,
float q )
```

Sets a new quantity of production to a process.

Exceptions

<code>std::domain_error</code>	if the process does not exist.
<code>std::invalid_argument</code>	if the quantity is less than zero.

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

- `student_distribution/lib/include/Models.hpp`

3.6 Process Class Reference

Represents a formula to generate a good.

```
#include <Models.hpp>
```

Public Member Functions

- `Process ()` noexcept
Default constructor, Don't use it.
- `Process (std::shared_ptr< Good > out)` noexcept
Constructor that receives the produced good.
- `Process (const Process &p)` noexcept
Copy constructor.
- `Process (Process &&p)` noexcept
Move constructor.
- `void swap (Process &p)` noexcept
Swap elements.
- `Process & operator= (Process p)` noexcept
Assignment operator.
- `float get_quantity (std::shared_ptr< Good > input)` const
- `void add_input (std::shared_ptr< Good > input, float quantity)`
- `void add_plant (std::shared_ptr< Plant > plant)` noexcept
Add a new plant that owns this process.
- `std::shared_ptr< Good > get_output ()` const noexcept
Gets the produced good.
- `std::unordered_map< std::shared_ptr< Good >, float > & get_inputs_table ()` noexcept
Gets the input table.
- `std::vector< std::tuple< std::shared_ptr< Good >, float > > get_inputs ()` const noexcept
Returns a list of tuples (good, quantity) of the inputs.
- `std::vector< std::shared_ptr< Plant > > & get_plants ()` noexcept
Returns the list of plants that own this process.

3.6.1 Detailed Description

Represents a formula to generate a good.

3.6.2 Member Function Documentation

3.6.2.1 add_input()

```
void Process::add_input (
    std::shared_ptr< Good > input,
    float quantity )
```

Add a new input associated with the required amount. If the input already exists, then the required amount is updated.

Exceptions

<code>std::invalid_argument</code>	if quantity less or equal to zero.
------------------------------------	------------------------------------

3.6.2.2 get_quantity()

```
float Process::get_quantity (
    std::shared_ptr< Good > input ) const
```

Returns the required amount of an input to produce one unit of the output.

Exceptions

<code>std::domain_error</code>	if the input does not belong to the process.
--------------------------------	--

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

- [student_distribution/lib/include/Models.hpp](#)

3.7 ProductionData Class Reference

```
#include <Models.hpp>
```

Public Types

- using `Relationship` = `std::tuple< std::shared_ptr< Plant >, std::shared_ptr< Plant >, std::shared_ptr< Good >, float, float >`
Represents a relationship (sale - purchase) between two plants.

Public Member Functions

- [ProductionData](#) ()=default
Default constructor.
- [ProductionData](#) (const [ProductionData](#) &)=delete
Deleted copy constructor.
- [ProductionData](#) ([ProductionData](#) &&pd) noexcept
Move constructor.
- [ProductionData](#) & [operator=](#) ([ProductionData](#) pd) noexcept
Assignment operator.
- void [swap](#) ([ProductionData](#) &pd) noexcept
Swap elements.

Public Attributes

- std::unordered_set< std::shared_ptr< [Good](#) > > **goods**
- std::unordered_set< std::shared_ptr< [Process](#) > > **processes**
- std::unordered_set< std::shared_ptr< [Plant](#) > > **plants**
- std::unordered_set< std::shared_ptr< [Company](#) > > **companies**
- std::vector< [Relationship](#) > **relationships**
List of relationships.

3.7.1 Detailed Description

This class contains all of the goods, processes, plants, and companies of a production network.

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

- student_distribution/lib/include/[Models.hpp](#)

3.8 ProductionNetwork Class Reference

Public Types

- using [NodeInfoType](#) = std::shared_ptr< [Plant](#) >
Alias to the data type that should be stored in each node.
- using [ArcInfoType](#) = std::tuple< std::shared_ptr< [Plant](#) >, std::shared_ptr< [Plant](#) >, std::shared_ptr< [Good](#) >, float, float >
Alias to the data type that should be stored in each arc.
- using [NetworkType](#) = void
Alias for the type of Graph. (optional usage).
- using [NodeData](#) = std::shared_ptr< [Plant](#) >
Aliases to retrieve data from the graph.
- using [ArcData](#) = std::tuple< std::shared_ptr< [Plant](#) >, std::shared_ptr< [Plant](#) >, std::shared_ptr< [Good](#) >, float, float >
- using [ChangedPriceData](#) = std::tuple< std::shared_ptr< [Company](#) >, std::shared_ptr< [Company](#) >, std::shared_ptr< [Good](#) >, float, float, float >

Public Member Functions

- **ProductionNetwork** (const [ProductionData](#) &data) noexcept
- **ProductionNetwork** (const [ProductionNetwork](#) &)=delete
- **ProductionNetwork** (const [ProductionNetwork](#) &&)=delete
- [ProductionNetwork](#) & **operator=** ([ProductionNetwork](#))=delete
- std::vector< [NodeData](#) > [get_node_info_list](#) () const noexcept
Returns the list of the plants stored in nodes.
- std::vector< [ArcData](#) > [get_arc_info_list](#) () const noexcept
Returns the list of tuples (seller, buyer, good, quantity, price) represented by the arcs.
- std::vector< [ChangedPriceData](#) > [change_price](#) (std::shared_ptr< [Good](#) > good, float change)
changes the sale price of a good.
- std::ostream & [to_dot](#) (std::ostream &output) const noexcept

3.8.1 Member Typedef Documentation

3.8.1.1 ChangedPriceData

```
using ProductionNetwork::ChangedPriceData = std::tuple<std::shared_ptr<Company>, std::shared_ptr<Company>, std::shared_ptr<Good>, float, float, float>
```

Alias for changed price data

Components: The first element is the seller company The second element is the buyer company The third element is the sold good. The fourth element is the old price. The fifth element is the new price. The sixth element is the percentage of change of the price.

3.8.2 Member Function Documentation

3.8.2.1 change_price()

```
std::vector<ChangedPriceData> ProductionNetwork::change\_price (
    std::shared_ptr< Good > good,
    float change )
```

changes the sale price of a good.

Changes the sales price of a good and computes the effect of this change, this means that all of the companies that use the good as input should change their products' sales prices. Then, there will be more companies affected by the change in the prices of their inputs.

Parameters

<i>good</i>	The good whose price is changed.
<i>change</i>	The amount of price change. This could be positive (increment), negative (decrement), or zero (no change).

Returns

a list with the relationship data affected by the change. Each element in the list is a tuple as follows: (seller company, buyer company, good, old_price, new_price, percentage_of_change).

Exceptions

<code>std::invalid_argument</code>	with the message "The good should be not null" if good is nullptr.
<code>std::domain_error</code>	with the message "Good does not exist" if the good is not in the production network.
<code>std::logic_error</code>	with the message "The good is not a first-level-good" if the good is not in level 0.

3.8.2.2 to_dot()

```
std::ostream& ProductionNetwork::to_dot (
    std::ostream & output ) const [noexcept]
```

Writes on the stream the DOT representation of the production network. See the documentation of Graphviz here: <https://graphviz.org/documentation/>

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

- student_distribution/lib/include/[ProductionNetwork.hpp](#)

3.9 Rng Class Reference

Singleton for random number generation.

```
#include <Rng.hpp>
```

Static Public Member Functions

- static float [random_uniform](#) () noexcept
Generates a random real number in [0, 1).
- template<typename T >
static T [random_uniform_int](#) (T l, T r) noexcept
Generates a random integer number in [l, r].
- template<typename T >
static T [random_uniform_real](#) (T l, T r) noexcept
Generates a random real number in [l, r].
- template<typename T >
static T [random_normal](#) (T mu, T sigma) noexcept
Generates a random number normally distributed.
- template<typename T >
static T [random_exponential](#) (T mu) noexcept
Generates a random number exponentially distributed.
- static bool [flip_coin](#) (float p=0.5) noexcept
Simulates a coin toss.
- template<typename It >
static It [random_choice](#) (It begin, It end) noexcept
Selects an item randomly between two iterators.
- static std::string [random_rif](#) () noexcept
Generates a random string with the R.I.F.-format.
- static void **set_seed** (std::mt19937::result_type s) noexcept

3.9.1 Detailed Description

Singleton for random number generation.

3.9.2 Member Function Documentation

3.9.2.1 flip_coin()

```
static bool Rng::flip_coin (
    float p = 0.5 ) [static], [noexcept]
```

Simulates a coin toss.

This uses the Bernoulli distribution to simulate a coin toss.

Parameters

<i>p</i>	The probability of getting true as the result. By default is 0.5 (fair coin).
----------	---

Returns

true for p.

false for 1 - p.

3.9.2.2 random_choice()

```
template<typename It >
It Rng::random_choice (
    It begin,
    It end ) [static], [noexcept]
```

Selects an item randomly between two iterators.

Template Parameters

<i>It</i>	The iterator type.
-----------	--------------------

Parameters

<i>begin</i>	The iterator on the start position.
<i>end</i>	The iterator on the final position.

Returns

The iterator on the selected item.

3.9.2.3 random_exponential()

```
template<typename T >
T Rng::random_exponential (
    T mu ) [static], [noexcept]
```

Generates a random number exponentially distributed.

Template Parameters

<i>T</i>	The floating-point type.
----------	--------------------------

Parameters

<i>mu</i>	Mean.
-----------	-------

Returns

the generated random number.

3.9.2.4 random_normal()

```
template<typename T >
T Rng::random_normal (
    T mu,
    T sigma ) [static], [noexcept]
```

Generates a random number normally distributed.

Template Parameters

<i>T</i>	The floating-point type.
----------	--------------------------

Parameters

<i>mu</i>	Mean.
<i>sigma</i>	Standard deviation.

Returns

the generated random number.

3.9.2.5 random_rif()

```
static std::string Rng::random_rif ( ) [static], [noexcept]
```

Generates a random string with the R.I.F.-format.

Returns

The generates R.I.F.

3.9.2.6 random_uniform_int()

```
template<typename T >  
T Rng::random_uniform_int (   
    T l,  
    T r ) [static], [noexcept]
```

Generates a random integer number in [l, r].

Template Parameters

<i>T</i>	The integral type.
----------	--------------------

Parameters

<i>l</i>	The lowerbound.
<i>r</i>	The upperbound.

Returns

the generated random number.

3.9.2.7 random_uniform_real()

```
template<typename T >  
T Rng::random_uniform_real (   
    T l,  
    T r ) [static], [noexcept]
```

Generates a random real number in [l, r].

Template Parameters

<i>T</i>	The floating-point type.
----------	--------------------------

Parameters

<i>l</i>	The lowerbound.
<i>r</i>	The upperbound.

Returns

the generated random number.

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

- `student_distribution/lib/include/Rng.hpp`

3.10 TestResults Struct Reference

Static Public Attributes

- static ReportList **results**

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

- `student_distribution/lib/include/TestUtils.hpp`

Chapter 4

File Documentation

4.1 student_distribution/lib/include/Generator.hpp File Reference

Classes

- class [NetworkGenerator](#)

4.1.1 Detailed Description

Author

Alejandro Mujica (aledrums@gmail.com)

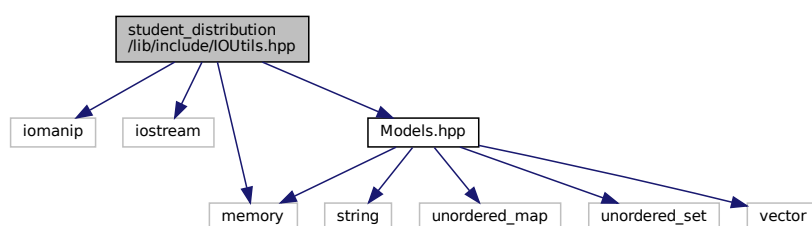
Copyright

Copyright (c) 2022

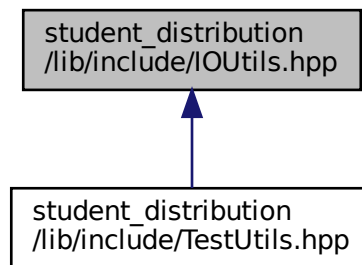
4.2 student_distribution/lib/include/IOUtils.hpp File Reference

```
#include <iomanip>
#include <iostream>
#include <memory>
#include <Models.hpp>
```

Include dependency graph for IOUtils.hpp:



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



Macros

- `#define LOG(...)`

Functions

- `std::ostream & operator<< (std::ostream &out, const std::shared_ptr< Company > &c)`
- `std::ostream & operator<< (std::ostream &out, const std::shared_ptr< Plant > &p)`
- `std::ostream & operator<< (std::ostream &out, const std::shared_ptr< Good > &g)`
- `std::ostream & operator<< (std::ostream &out, const std::tuple< std::shared_ptr< Plant >, std::shared_ptr< Plant >, std::shared_ptr< Good >, float, float > &t)`
- `std::ostream & operator<< (std::ostream &out, const std::tuple< std::shared_ptr< Company >, std::shared_ptr< Company >, std::shared_ptr< Good >, float, float, float > &t)`
- `template<typename T >`
`std::ostream & operator<< (std::ostream &out, const std::vector< T > &v)`

4.2.1 Detailed Description

Author

Alejandro Mujica (aledrums@gmail.com)

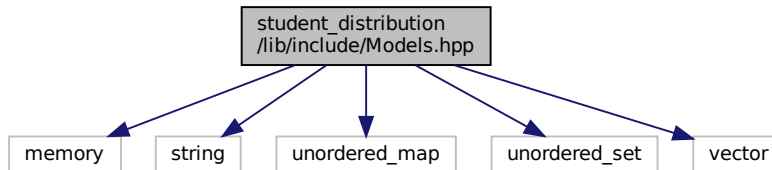
Copyright

Copyright (c) 2022

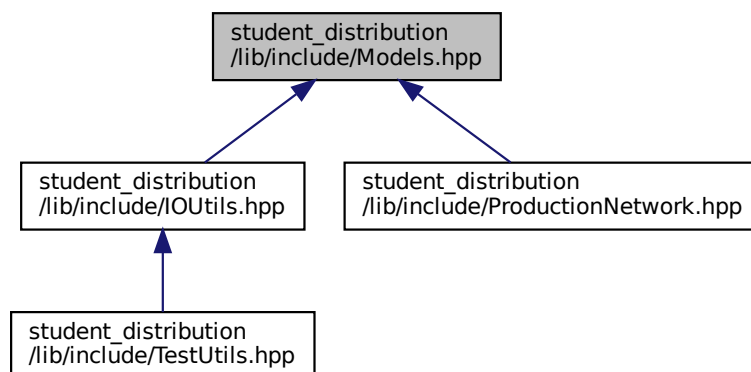
4.3 student_distribution/lib/include/Models.hpp File Reference

```
#include <memory>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <vector>
```

Include dependency graph for Models.hpp:



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



Classes

- class [Good](#)
Model for goods or services.
- class [Process](#)
Represents a formula to generate a good.
- class [Plant](#)
- class [Company](#)
Model for a company.
- class [ModelBuilder](#)
- class [ProductionData](#)

4.3.1 Detailed Description

Author

Alejandro Mujica (aledrums@gmail.com)

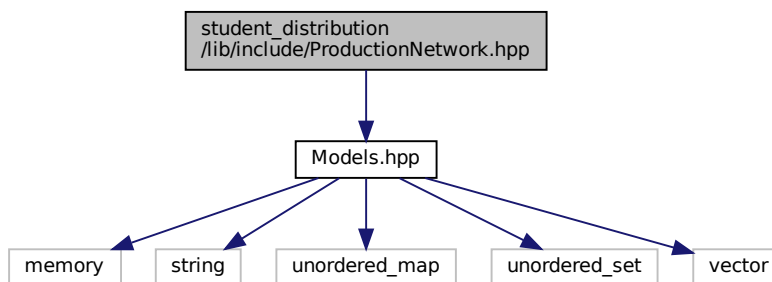
Copyright

Copyright (c) 2022

4.4 student_distribution/lib/include/ProductionNetwork.hpp File Reference

```
#include <Models.hpp>
```

Include dependency graph for ProductionNetwork.hpp:



Classes

- class [ProductionNetwork](#)

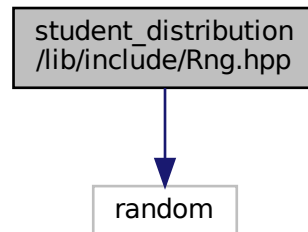
4.4.1 Detailed Description

Authors

4.5 student_distribution/lib/include/Rng.hpp File Reference

```
#include <random>
```

Include dependency graph for Rng.hpp:



Classes

- class [Rng](#)
Singleton for random number generation.

4.5.1 Detailed Description

Author

Alejandro Mujica (aledrums@gmail.com)

Copyright

Copyright (c) 2022

