My Vensin

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical 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:

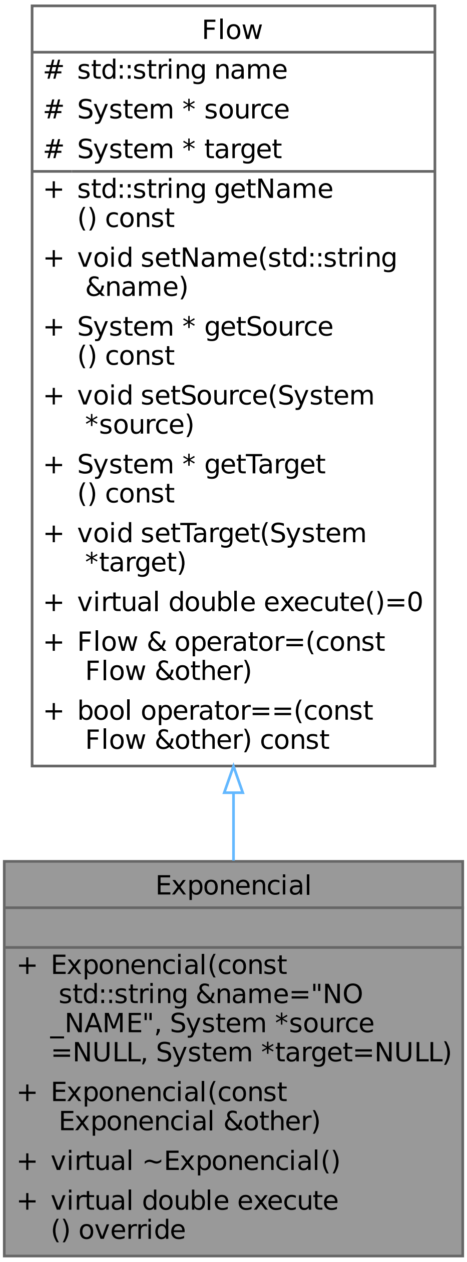
Chapter 4

Class Documentation

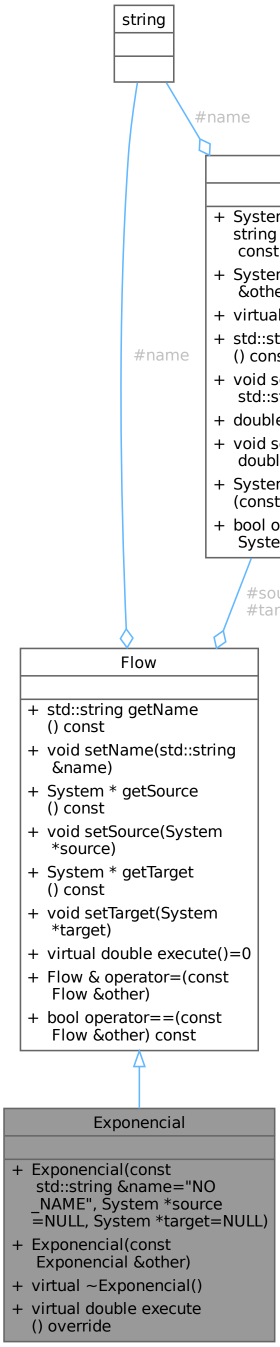
4.1 Exponencial Class Reference

```
#include <Exponencial.h>
```

Inheritance diagram for Exponencial:


Flow
# std::string name
# System * source
# System * target
+ std::string getName () const
+ void setName(std::string &name)
+ System * getSource () const
+ void setSource(System *source)
+ System * getTarget () const
+ void setTarget(System *target)
+ virtual double execute()=0
+ Flow & operator=(const Flow &other)
+ bool operator==(const Flow &other) const
Exponencial
+ Exponencial(const std::string &name="NO_NAME", System *source =NULL, System *target=NULL)
+ Exponencial(const Exponencial &other)
+ virtual ~Exponencial()
+ virtual double execute () override

Collaboration diagram for Exponencial:


string
double
#name
#value
System
+ System(const std::string &name="NO_NAME", const double &value=0.0)
+ System(const System &other)
+ virtual ~System()
+ std::string getName() const
+ void setName(const std::string &name)
+ double getValue() const
+ void setValue(const double &value)
+ System & operator=(const System &other)
+ bool operator==(const System &other) const
#name
#source
#target
Flow
+ std::string getName() const
+ void setName(std::string &name)
+ System * getSource() const
+ void setSource(System *source)
+ System * getTarget() const
+ void setTarget(System *target)
+ virtual double execute()=0
+ Flow & operator=(const Flow &other)
+ bool operator==(const Flow &other) const
Exponencial
+ Exponencial(const std::string &name="NO_NAME", System *source=NULL, System *target=NULL)
+ Exponencial(const Exponencial &other)
+ virtual ~Exponencial()
+ virtual double execute() override

Public Member Functions

- Exponencial (const std::string &name="NO_NAME", System ∗source=NULL, System ∗target=NULL)

  Construct a new Exponencial by name, source and target.
- Exponencial (const Exponencial &other)

  Construct a new Exponencial by a obj.
- virtual ∼Exponencial ()

This destructor is a virtual destructor of the Class.

- virtual double execute () override

  Pure virtual method that will contain an equation that will be executed in the flow by the model.

Public Member Functions inherited from Flow

- std::string getName () const

  This method returns the name of a flow.
- void setName (std::string &name)

  This method assigns a string to the name of a flow obj.
- System ∗ getSource () const

  This method returns the source system poiter.
- void setSource (System ∗source)

  This method assigns a system poiter to the source of a flow obj.
- System ∗ getTarget () const

  This method returns the target system poiter.
- void setTarget (System ∗target)

  This method assigns a system poiter to the target of a flow obj.
- Flow & operator= (const Flow &other)

  This method is overloading the '=' operator, "cloning" from one flow to another.
- bool operator== (const Flow &other) const

  This method is overloading the '==' operator, compare two flows objs.

Additional Inherited Members

Protected Attributes inherited from Flow

- std::string name
- System ∗ source
- System ∗ target

4.1.1 Constructor & Destructor Documentation

4.1.1.1 Exponencial() [1/2]

```
Exponencial::Exponencial (
             const std::string & name = "NO_NAME",
             System * source = NULL,
             System * target = NULL )
```

Construct a new Exponencial by name, source and target.

Parameters

name	string with default value "NO_NAME"
source	System pointer with default value NULL
target	System pointer with default value NULL

```
                                                                                  {
    this->name = name;
    this->source = source;
    this->target = target;
}
```

References Flow::name, Flow::source, and Flow::target.

4.1.1.2 Exponencial() [2/2]

```
Exponencial::Exponencial (
            const Exponencial & other )
```

Construct a new Exponencial by a obj.

Parameters

other	Exponencial obj

```
                                            {
    this->name = other.name;
    this->source = other.source;
    this->target = other.target;
}
```

References Flow::name, Flow::source, and Flow::target.

4.1.1.3 ∼Exponencial()

```
Exponencial::∼Exponencial ( )  [virtual]
```

This destructor is a virtual destructor of the Class.

```
{}
```

4.1.2 Member Function Documentation

4.1.2.1 execute()

```
double Exponencial::execute ( )  [override], [virtual]
```

Pure virtual method that will contain an equation that will be executed in the flow by the model.

Returns

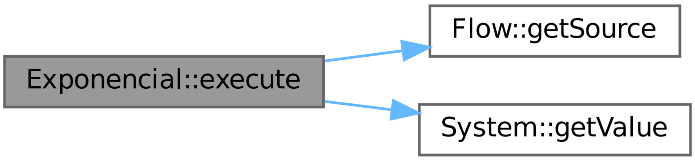
double

Implements Flow.

```
                        {
    return getSource()->getValue() * 0.01;
}
```

References Flow::getSource(), and System::getValue().

Here is the call graph for this function:



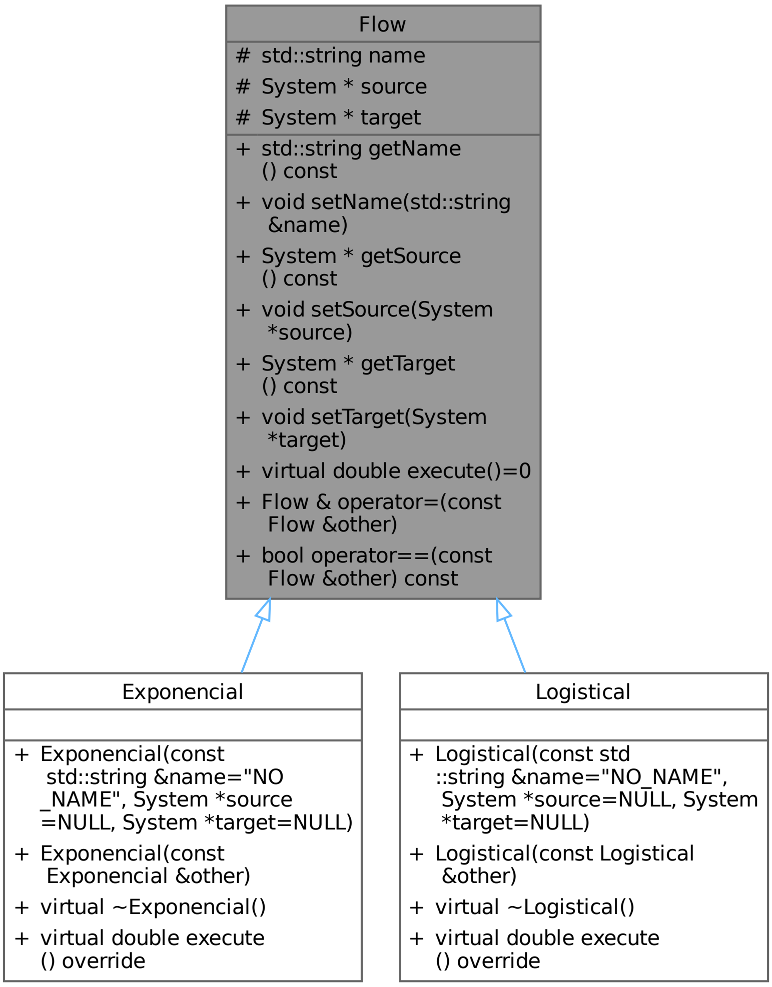
The documentation for this class was generated from the following files:

- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↩ tests/src/Exponencial.h
- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↩ tests/src/Exponencial.cpp

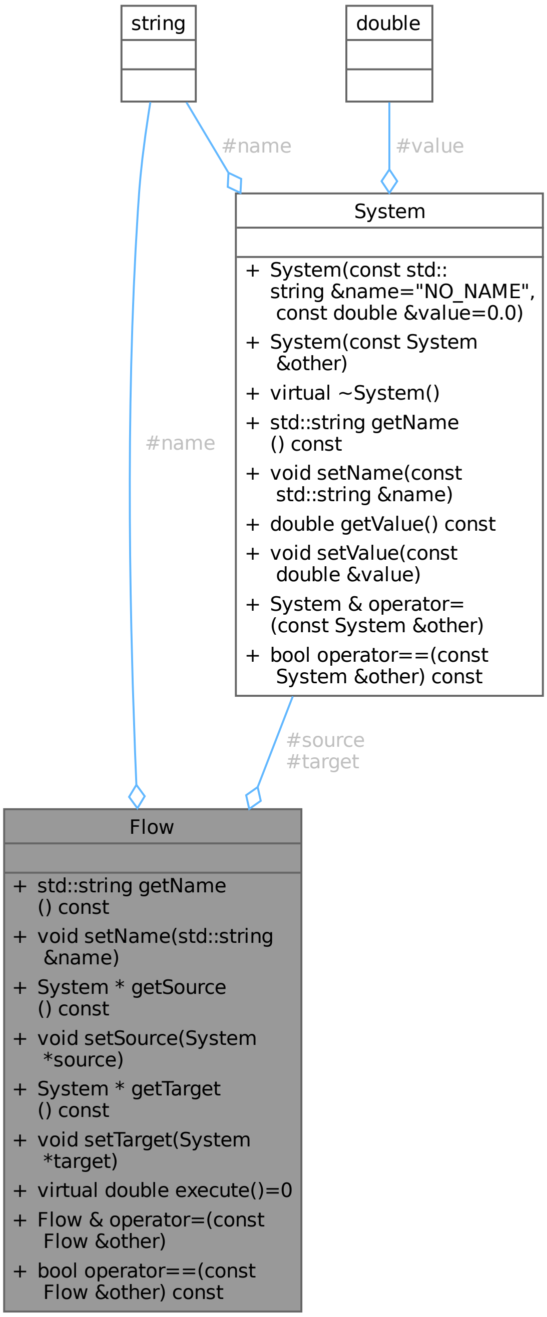
4.2 Flow Class Reference

```
#include <Flow.h>
```

Inheritance diagram for Flow:


Flow
# std::string name
# System * source
# System * target
+ std::string getName () const
+ void setName(std::string &name)
+ System * getSource () const
+ void setSource(System *source)
+ System * getTarget () const
+ void setTarget(System *target)
+ virtual double execute()=0
+ Flow & operator=(const Flow &other)
+ bool operator==(const Flow &other) const
Exponencial
+ Exponencial(const std::string &name="NO_NAME", System *source =NULL, System *target=NULL)
+ Exponencial(const Exponencial &other)
+ virtual ~Exponencial()
+ virtual double execute () override
Logistical
+ Logistical(const std ::string &name="NO_NAME", System *source=NULL, System *target=NULL)
+ Logistical(const Logistical &other)
+ virtual ~Logistical()
+ virtual double execute () override

Collaboration diagram for Flow:



Public Member Functions

- std::string getName () const

  This method returns the name of a flow.
- void setName (std::string &name)

  This method assigns a string to the name of a flow obj.
- System ∗ getSource () const

This method returns the source system poiter.

- void setSource (System ∗source)

  This method assigns a system poiter to the source of a flow obj.
- System ∗ getTarget () const

  This method returns the target system poiter.
- void setTarget (System ∗target)

  This method assigns a system poiter to the target of a flow obj.
- virtual double execute ()=0

  Pure virtual method that will be inherited by subclasses created by the user, this one will contain an equation that will be executed in the flow by the model.
- Flow & operator= (const Flow &other)

  This method is overloading the '=' operator, "cloning" from one flow to another.
- bool operator== (const Flow &other) const

  This method is overloading the '==' operator, compare two flows objs.

Protected Attributes

- std::string name
- System ∗ source
- System ∗ target

Friends

- std::ostream & operator<< (std::ostream &out, const Flow &obj)

  This method is overloading the '<<' operator, print the flow obj info.

4.2.1 Member Function Documentation

4.2.1.1 execute()

```
virtual double Flow::execute ( )  [pure virtual]
```

Pure virtual method that will be inherited by subclasses created by the user, this one will contain an equation that will be executed in the flow by the model.

Returns

double

Implemented in Exponencial, and Logistical.

4.2.1.2 getName()

```
std::string Flow::getName ( ) const
```

This method returns the name of a flow.

Returns

a string containing the name is returned

```
00005 { return name; }
```

References name.

4.2.1.3 getSource()

```
System * Flow::getSource ( ) const
```

This method returns the source system poiter.

Returns

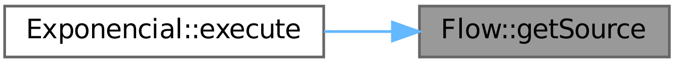
a system poiter containing the source memory address is returned

```
00008 { return source; }
```

References source.

Referenced by Exponencial::execute().

Here is the caller graph for this function:



4.2.1.4 getTarget()

```
System * Flow::getTarget ( ) const
```

This method returns the target system poiter.

Returns

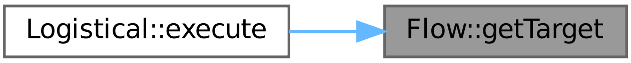
a system poiter containing the target memory address is returned

```
00011 { return target; }
```

References target.

Referenced by Logistical::execute().

Here is the caller graph for this function:



4.2.1.5 operator=()

```
Flow & Flow::operator= (
            const Flow & other )
```

This method is overloading the '=' operator, "cloning" from one flow to another.

Parameters

other	flow obj to be cloned must be passed

Returns

A flow is returned that is a clone of what was passed to the method

```
                                                  {
    if(other == *this) return *this;
    name = other.name;
    source = other.source;
    target = other.target;
    return *this;
}
```

References name, source, and target.

4.2.1.6 operator==()

```
bool Flow::operator== (
              const Flow & other ) const
```

This method is overloading the '==' operator, compare two flows objs.

Parameters

other	flow obj to be compare must be passed

Returns

A bool is returned, true if they are equal and false if not

```
                                                        {
    return (name == other.name && source == other.source && target == other.target);
}
```

References name, source, and target.

4.2.1.7 setName()

```
void Flow::setName (
              std::string & name )
```

This method assigns a string to the name of a flow obj.

Parameters

name	string must be passed to the method

```
{ this->name = name; }
```

References name.

4.2.1.8 setSource()

```
void Flow::setSource (
            System * source )
```

This method assigns a system poiter to the source of a flow obj.

Parameters

source	system poiter must be passed to the method

```
{ this->source = source; }
```

References source.

4.2.1.9 setTarget()

```
void Flow::setTarget (
            System * target )
```

This method assigns a system poiter to the target of a flow obj.

Parameters

target	system poiter must be passed to the method

```
{ this->target = target; }
```

References target.

4.2.2 Friends And Related Symbol Documentation

4.2.2.1 operator<<

```
std::ostream & operator<< (
            std::ostream & out,
            const Flow & obj ) [friend]
```

This method is overloading the '<<' operator, print the flow obj info.

Parameters

out	is a ostream obj
obj	is a flow obj

Returns

a ostream obj to print the obj info

```
                                                        {
    out « "(Flow) Name: " « obj.name « " - "
        « obj.source->getName() « " -----> " « obj.target->getName();
    return out;
}
```

4.2.3 Member Data Documentation

4.2.3.1 name

```
std::string Flow::name  [protected]
```

Name string attribute.

Referenced by Exponencial::Exponencial(), Exponencial::Exponencial(), getName(), Logistical::Logistical(), Logistical::Logistical(), operator=(), operator==(), and setName().

4.2.3.2 source

```
System* Flow::source  [protected]
```

Source system pointer attribute.

Referenced by Exponencial::Exponencial(), Exponencial::Exponencial(), getSource(), Logistical::Logistical(), Logistical::Logistical(), operator=(), operator==(), and setSource().

4.2.3.3 target

```
System* Flow::target  [protected]
```

Target system pointer attribute.

Referenced by Exponencial::Exponencial(), Exponencial::Exponencial(), getTarget(), Logistical::Logistical(), Logistical::Logistical(), operator=(), operator==(), and setTarget().

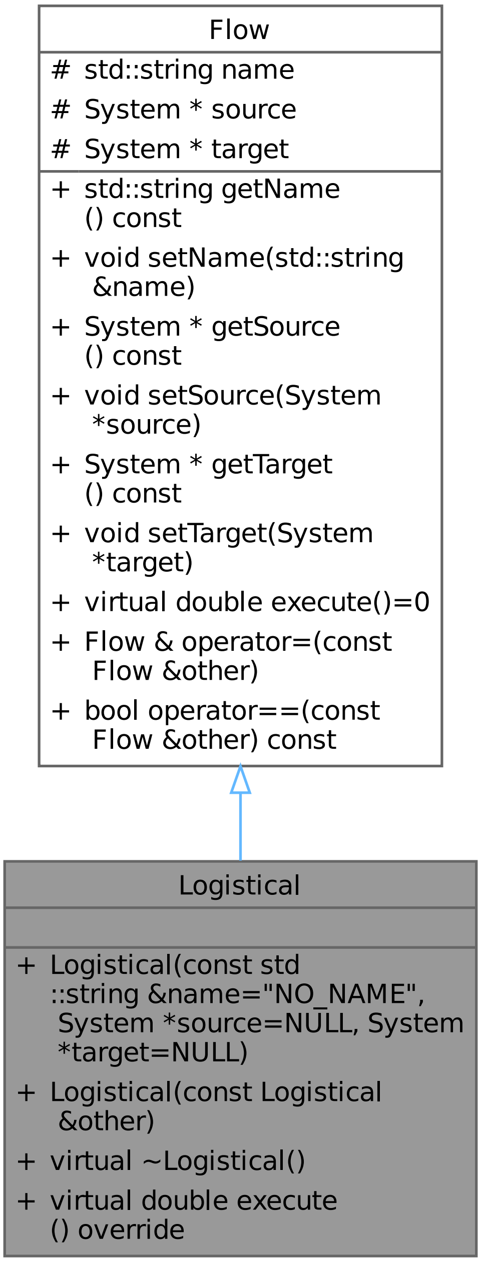
The documentation for this class was generated from the following files:

- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Flow.h
- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Flow.cpp

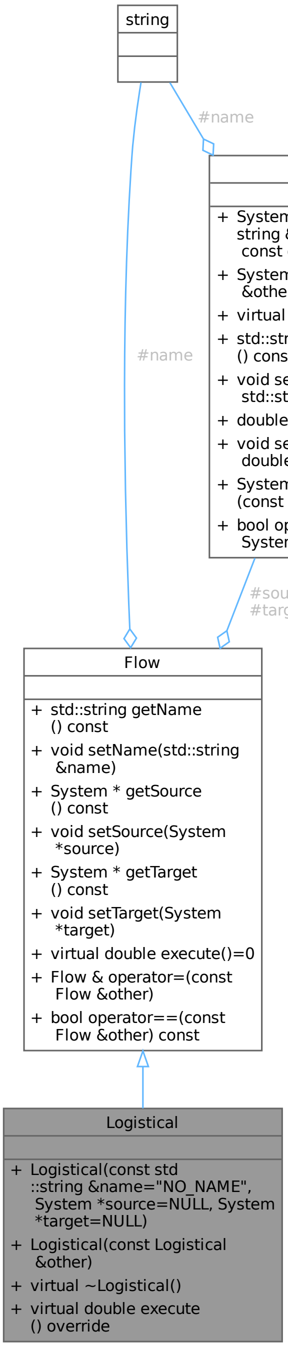
4.3 Logistical Class Reference

```
#include <Logistical.h>
```

Inheritance diagram for Logistical:


Flow
# std::string name
# System * source
# System * target
+ std::string getName () const
+ void setName(std::string &name)
+ System * getSource () const
+ void setSource(System *source)
+ System * getTarget () const
+ void setTarget(System *target)
+ virtual double execute()=0
+ Flow & operator=(const Flow &other)
+ bool operator==(const Flow &other) const
Logistical
+ Logistical(const std ::string &name="NO_NAME", System *source=NULL, System *target=NULL)
+ Logistical(const Logistical &other)
+ virtual ~Logistical()
+ virtual double execute () override

Collaboration diagram for Logistical:



Public Member Functions

- Logistical (const std::string &name="NO_NAME", System ∗source=NULL, System ∗target=NULL)

    Construct a new Logistical by name, source and target.
- Logistical (const Logistical &other)

    Construct a new Logistical by a obj.
- virtual ∼Logistical ()

This destructor is a virtual destructor of the Class.

- virtual double execute () override

  Pure virtual method that will contain an equation that will be executed in the flow by the model.

Public Member Functions inherited from Flow

- std::string getName () const

  This method returns the name of a flow.
- void setName (std::string &name)

  This method assigns a string to the name of a flow obj.
- System ∗ getSource () const

  This method returns the source system poiter.
- void setSource (System ∗source)

  This method assigns a system poiter to the source of a flow obj.
- System ∗ getTarget () const

  This method returns the target system poiter.
- void setTarget (System ∗target)

  This method assigns a system poiter to the target of a flow obj.
- Flow & operator= (const Flow &other)

  This method is overloading the '=' operator, "cloning" from one flow to another.
- bool operator== (const Flow &other) const

  This method is overloading the '==' operator, compare two flows objs.

Additional Inherited Members

Protected Attributes inherited from Flow

- std::string name
- System ∗ source
- System ∗ target

4.3.1 Constructor & Destructor Documentation

4.3.1.1 Logistical() [1/2]

```
Logistical::Logistical (
             const std::string & name = "NO_NAME",
             System * source = NULL,
             System * target = NULL )
```

Construct a new Logistical by name, source and target.

Parameters

name	string with default value "NO_NAME"
source	System pointer with default value NULL
target	System pointer with default value NULL

```
                                                                            {
    this->name = name;
    this->source = source;
    this->target = target;
}
```

References Flow::name, Flow::source, and Flow::target.

4.3.1.2 Logistical() [2/2]

```
Logistical::Logistical (
              const Logistical & other )
```

Construct a new Logistical by a obj.

Parameters

other	Logistical obj

```
                                            {
    this->name = other.name;
    this->source = other.source;
    this->target = other.target;
}
```

References Flow::name, Flow::source, and Flow::target.

4.3.1.3 ∼Logistical()

```
Logistical::∼Logistical ( ) [virtual]
```

This destructor is a virtual destructor of the Class.

```
{}
```

4.3.2 Member Function Documentation

4.3.2.1 execute()

```
double Logistical::execute ( ) [override], [virtual]
```

Pure virtual method that will contain an equation that will be executed in the flow by the model.

Returns

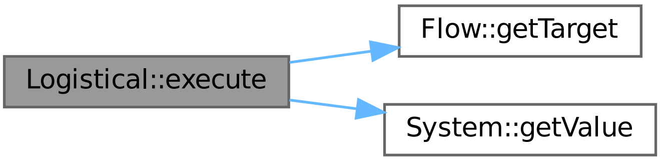
double

Implements Flow.

```
                        {
    return 0.01 * getTarget()->getValue() * (1.0 - getTarget()->getValue() / 70.0);
}
```

References Flow::getTarget(), and System::getValue().

Here is the call graph for this function:



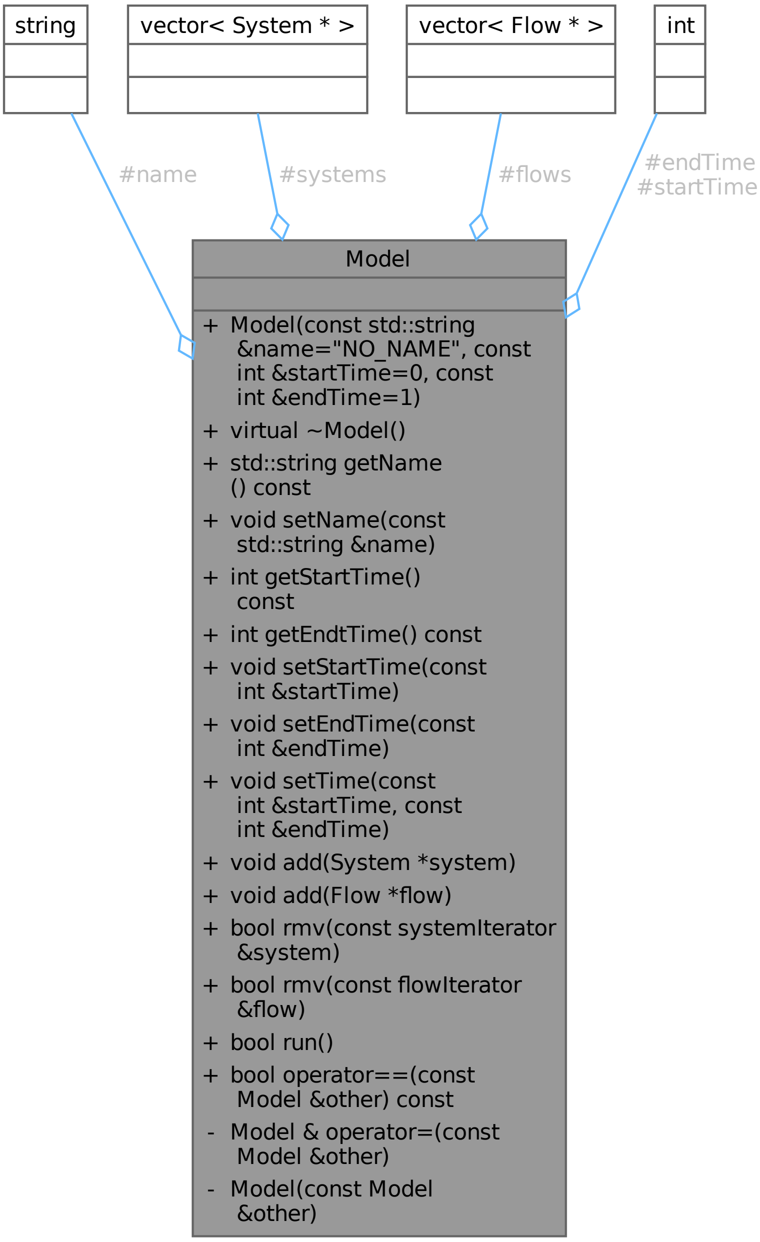
The documentation for this class was generated from the following files:

- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↩ tests/src/Logistical.h
- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↩ tests/src/Logistical.cpp

4.4 Model Class Reference

```
#include <Model.h>
```

Collaboration diagram for Model:


string
vector< System * >
vector< Flow * >
int
#name
#systems
#flows
#endTime
#startTime
Model
+ Model(const std::string &name="NO_NAME", const int &startTime=0, const int &endTime=1)
+ virtual ~Model()
+ std::string getName () const
+ void setName(const std::string &name)
+ int getStartTime() const
+ int getEndtTime() const
+ void setStartTime(const int &startTime)
+ void setEndTime(const int &endTime)
+ void setTime(const int &startTime, const int &endTime)
+ void add(System *system)
+ void add(Flow *flow)
+ bool rmv(const systemIterator &system)
+ bool rmv(const flowIterator &flow)
+ bool run()
+ bool operator==(const Model &other) const
- Model & operator=(const Model &other)
- Model(const Model &other)

Public Types

- typedef std::vector< System ∗ >::iterator systemIterator

  typedef vetors iterators
- typedef std::vector< Flow ∗ >::iterator flowIterator

Public Member Functions

- Model (const std::string &name="NO_NAME", const int &startTime=0, const int &endTime=1)

Construct a new Model by name and sart and end time.

- virtual ∼Model ()

    This destructor is a virtual destructor of the class.

- std::string getName () const

    This method returns the name of a Model.

- void setName (const std::string &name)

    This method assigns a string to the name of a Model.

- int getStartTime () const

    This method returns the startTime of a Model.

- int getEndtTime () const

    This method returns the end of a Model.

- void setStartTime (const int &startTime)

    This method assigns a int to the startTime of a Model.

- void setEndTime (const int &endTime)

    This method assigns a int to the endTime of a Model.

- void setTime (const int &startTime, const int &endTime)

    This method assigns a int to the startTime and endTime of a Model.

- void add (System ∗system)

    This method add a System pointer to the vector of a Model.

- void add (Flow ∗flow)

    This method add a Flow pointer to the vector of a Model.

- bool rmv (const systemIterator &system)

    This method remove a System pointer of the vector of a Model.

- bool rmv (const flowIterator &flow)

    This method remove a Flow pointer of the vector of a Model.

- bool run ()

    This method run all model.

- bool operator== (const Model &other) const

    This method is overloading the '==' operator, compare two models objs.

Protected Attributes

- std::string name
- std::vector< System ∗ > systems
- std::vector< Flow ∗ > flows
- int startTime
- int endTime

Private Member Functions

- Model & operator= (const Model &other)

    This method is overloading the '=' operator, "cloning" from one Model to another.

- Model (const Model &other)

    Construct a new Model by a obj.

Friends

- std::ostream & operator<< (std::ostream &out, const Model &obj)

    This method is overloading the '<<' operator, print the model obj info.

4.4.1 Member Typedef Documentation

4.4.1.1 flowIterator

```
typedef std::vector<Flow*>::iterator Model::flowIterator
```

4.4.1.2 systemIterator

```
typedef std::vector<System*>::iterator Model::systemIterator
```

typedef vetors iterators

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Model() [1/2]

```
Model::Model (
            const Model & other )  [private]
```

Construct a new Model by a obj.

Parameters

other	Model obj

```
                                : name(other.name), startTime(other.startTime), endTime(other.endTime)
{
    flows.clear();
    systems.clear();
    for (auto i : other.flows) flows.push_back(i);
    for (auto i : other.systems) systems.push_back(i);
}
```

References flows, and systems.

4.4.2.2 Model() [2/2]

```
Model::Model (
            const std::string & name = "NO_NAME",
            const int & startTime = 0,
            const int & endTime = 1 )
```

Construct a new Model by name and sart and end time.

Parameters

name	string with default value "NO_NAME"
startTime	int with default value 0
endTime	int with default value 1

```
: name(name), startTime(startTime), endTime(endTime) {}
```

4.4.2.3 ~Model()

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

This destructor is a virtual destructor of the class.

```
{systems.clear(); flows.clear();}
```

References flows, and systems.

4.4.3 Member Function Documentation

4.4.3.1 add() [1/2]

```
void Model::add (
            Flow * flow )
```

This method add a Flow pointer to the vector of a Model.

Parameters

flow	Flow pointer must be passed to the method

```
{ flows.push_back(flow); }
```

References flows.

4.4.3.2 add() [2/2]

```
void Model::add (
            System * system )
```

This method add a System pointer to the vector of a Model.

Parameters

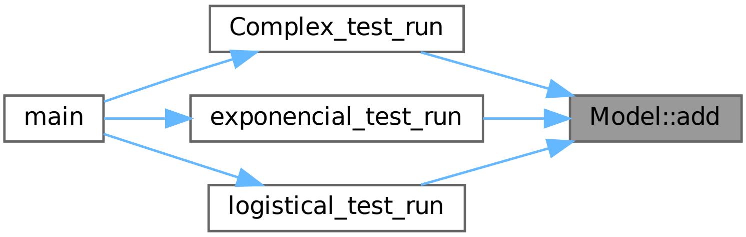
system	System pointer must be passed to the method

```
{ systems.push_back(system); }
```

References systems.

Referenced by Complex_test_run(), exponencial_test_run(), and logistical_test_run().

Here is the caller graph for this function:



4.4.3.3 getEndtTime()

```
int Model::getEndtTime ( ) const
```

This method returns the end of a Model.

Returns

a int containing the end is returned

```
00022 { return endTime; }
```

References endTime.

4.4.3.4 getName()

```
std::string Model::getName ( ) const
```

This method returns the name of a Model.

Returns

a string containing the name is returned

```
00018 { return name; }
```

References name.

4.4.3.5 getStartTime()

```
int Model::getStartTime ( ) const
```

This method returns the startTime of a Model.

Returns

a int containing the startTime is returned

```
00021 { return startTime; }
```

References startTime.

4.4.3.6 operator=()

```
Model & Model::operator= (
             const Model & other )   [private]
```

This method is overloading the '=' operator, "cloning" from one Model to another.

Parameters

other	Model obj to be cloned must be passed

Returns

A Model is returned that is a clone of what was passed to the method

```
                                        {
    if(other == *this) return *this;
    name = other.name;
    systems = other.systems;
    flows.clear();
    systems.clear();
    for (auto i : other.flows) flows.push_back(i);
    for (auto i : other.systems) systems.push_back(i);
    startTime = other.startTime;
    endTime = other.endTime;
    return *this;
}
```

References endTime, flows, name, startTime, and systems.

4.4.3.7 operator==()

```
bool Model::operator== (
              const Model & other ) const
```

This method is overloading the '==' operator, compare two models objs.

Parameters

other	model obj to be compare must be passed

Returns

A bool is returned, true if they are equal and false if not

```
                                             {
    return (name == other.name && systems == other.systems && flows == other.flows && startTime ==
other.startTime && endTime == other.endTime);
}
```

References endTime, flows, name, startTime, and systems.

4.4.3.8 rmv() [1/2]

```
bool Model::rmv (
              const flowIterator & flow )
```

This method remove a Flow pointer of the vector of a Model.

Parameters

flow	Flow pointer iterator must be passed to the method

Returns

a bool value, true if can remove, false if not

```
{ return (flows.erase(flow) != flows.end()); }
```

References flows.

4.4.3.9 rmv() [2/2]

```
bool Model::rmv (
                const systemIterator & system )
```

This method remove a System pointer of the vector of a Model.

Parameters

system	System pointer iterator must be passed to the method

Returns

a bool value, true if can remove, false if not

```
{ return (systems.erase(system) != systems.end()); }
```

References systems.

4.4.3.10 run()

```
bool Model::run ( )
```

This method run all model.

Returns

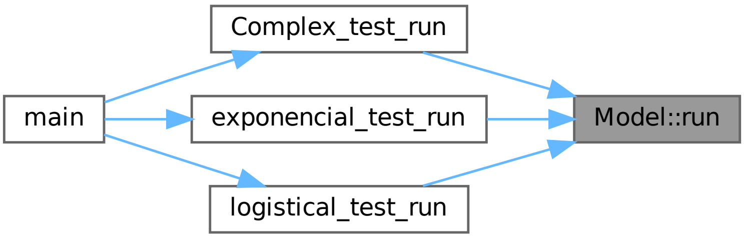
a bool value, true if can run, false if not

```
           {
std::vector<double> flowValue;
flowIterator f;
std::vector<double>::iterator d;
double calcValue;

for(int i = startTime; i < endTime; i++){

    f = flows.begin();

    while (f != flows.end()) {
        flowValue.push_back((*f)->execute());
        f++;
    }

    f = flows.begin();
    d = flowValue.begin();

    while(f != flows.end()){
        calcValue = (*f)->getSource()->getValue() - (*d);
        (*f)->getSource()->setValue(calcValue);
        calcValue = (*f)->getTarget()->getValue() + (*d);
        (*f)->getTarget()->setValue(calcValue);
        f++;
        d++;
    }

    flowValue.clear();
00064
00065     }
00066
00067     return true;
00068 }
```

References endTime, flows, and startTime.

Referenced by Complex_test_run(), exponencial_test_run(), and logistical_test_run().

Here is the caller graph for this function:



4.4.3.11 setEndTime()

```
void Model::setEndTime (
            const int & endTime )
```

This method assigns a int to the endTime of a Model.

Parameters

endTime	int must be passed to the method

```
00024 { this->endTime = endTime; }
```

References endTime.

4.4.3.12 setName()

```
void Model::setName (
            const std::string & name )
```

This method assigns a string to the name of a Model.

Parameters

name	string must be passed to the method

```
00019 { this->name = name; }
```

References name.

4.4.3.13 setStartTime()

```
void Model::setStartTime (
             const int & startTime )
```

This method assigns a int to the startTime of a Model.

Parameters

startTime	int must be passed to the method

```
00023 { this->startTime = startTime; }
```

References startTime.

4.4.3.14 setTime()

```
void Model::setTime (
             const int & startTime,
             const int & endTime )
```

This method assigns a int to the startTime and endTime of a Model.

Parameters

startTime	int must be passed to the method
endTime	int must be passed to the method

```
00025 { this->startTime = startTime; this->endTime = endTime; }
```

References endTime, and startTime.

4.4.4 Friends And Related Symbol Documentation

4.4.4.1 operator<<

```
std::ostream & operator<< (
             std::ostream & out,
             const Model & obj )  [friend]
```

This method is overloading the '<<' operator, print the model obj info.

Parameters

out	is a ostream obj
obj	is a model obj

Returns

a ostream obj to print the obj info

```
                                                          {
    out « "Name: " « obj.name « ";\n"
        « "Systems:\n";
    for (auto item : obj.systems) out « item « "\n";
    out « "Flows:\n";
    for (auto item : obj.flows) out « item « "\n";
    return out;
}
```

4.4.5 Member Data Documentation

4.4.5.1 endTime

`int Model::endTime [protected]`

End time simulation integer attribute.

Referenced by getEndtTime(), operator=(), operator==(), run(), setEndTime(), and setTime().

4.4.5.2 flows

`std::vector<Flow*> Model::flows [protected]`

Flow pointers vector.

Referenced by add(), Model(), operator=(), operator==(), rmv(), run(), and ∼Model().

4.4.5.3 name

`std::string Model::name [protected]`

Name string attribute.

Referenced by getName(), operator=(), operator==(), and setName().

4.4.5.4 startTime

`int Model::startTime [protected]`

Start time simulation integer attribute.

Referenced by getStartTime(), operator=(), operator==(), run(), setStartTime(), and setTime().

4.4.5.5 systems

```
std::vector<System*> Model::systems  [protected]
```

System pointers vector.

Referenced by add(), Model(), operator=(), operator==(), rmv(), and ∼Model().

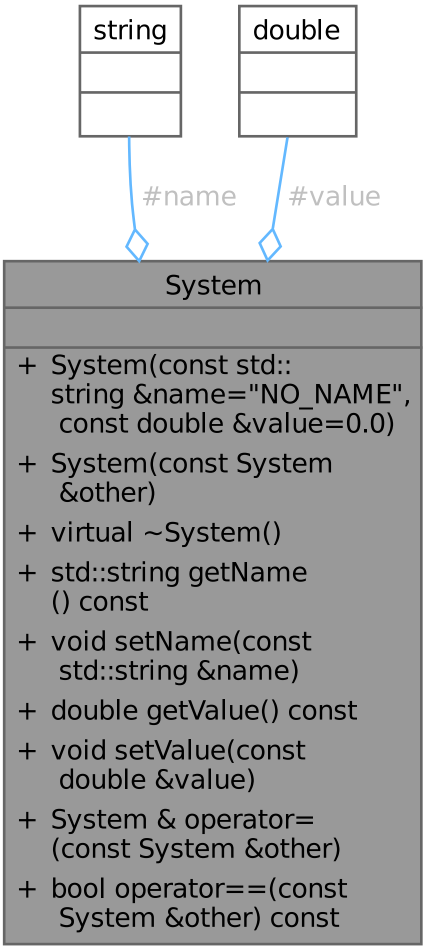
The documentation for this class was generated from the following files:

- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.h
- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.cpp

4.5 System Class Reference

```
#include <System.h>
```

Collaboration diagram for System:



Public Member Functions

- System (const std::string &name="NO_NAME", const double &value=0.0)

  Construct a new System by name and value.
- System (const System &other)

  Construct a new System by a obj.
- virtual ∼System ()

  This destructor is a virtual destructor of the Class.
- std::string getName () const

  This method returns the name of a system.
- void setName (const std::string &name)

  This method assigns a string to the name of a system.
- double getValue () const

  This method returns the value of a system.
- void setValue (const double &value)

  This method assigns a double to the value of a system.
- System & operator= (const System &other)

  This method is overloading the '=' operator, "cloning" from one system to another.
- bool operator== (const System &other) const

  This method is overloading the '==' operator, compare two systems objs.

Protected Attributes

- std::string name
- double value

Friends

- std::ostream & operator<< (std::ostream &out, const System &obj)

  This method is overloading the '<<' operator, print the system obj info.

4.5.1 Constructor & Destructor Documentation

4.5.1.1 System() [1/2]

```
System::System (
              const std::string & name = "NO_NAME",
              const double & value = 0.0 )
```

Construct a new System by name and value.

Parameters

name	string with default value "NO_NAME"
value	double with default value 0.0

```
00004 : name(name), value(value) {}
```

4.5.1.2 System() [2/2]

```
System::System (
            const System & other )
```

Construct a new System by a obj.

Parameters

other	System obj

```
00006 : name(other.name), value(other.value) {}
```

4.5.1.3 ∼System()

```
System::∼System ( )  [virtual]
```

This destructor is a virtual destructor of the Class.

```
00009 {};
```

4.5.2 Member Function Documentation

4.5.2.1 getName()

```
std::string System::getName ( ) const
```

This method returns the name of a system.

Returns

a string containing the name is returned

```
00013 { return name; }
```

References name.

4.5.2.2 getValue()

```
double System::getValue ( ) const
```

This method returns the value of a system.

Returns

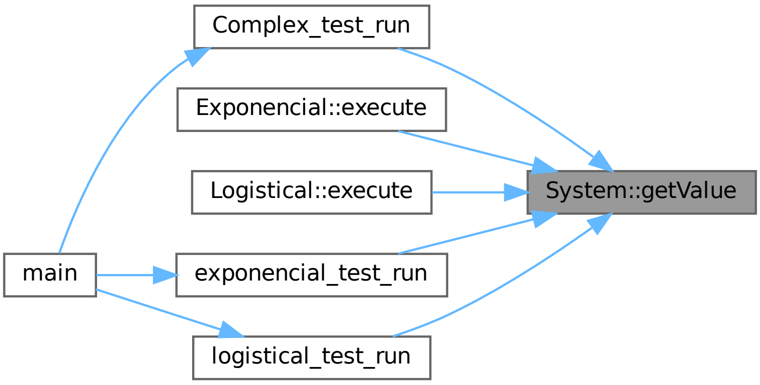
a double containing the value is returned

```
{ return value; }
```

References value.

Referenced by Complex_test_run(), Exponencial::execute(), Logistical::execute(), exponencial_test_run(), and logistical_test_run().

Here is the caller graph for this function:



4.5.2.3 operator=()

```
System & System::operator= (
              const System & other )
```

This method is overloading the '=' operator, "cloning" from one system to another.

Parameters

other	system obj to be cloned must be passed

Returns

A system is returned that is a clone of what was passed to the method

```
                                               {
    if(other == *this) return *this;
    name = other.name;
    value = other.value;
    return *this;
}
```

References name, and value.

4.5.2.4 operator==()

```
bool System::operator== (
            const System & other ) const
```

This method is overloading the '==' operator, compare two systems objs.

Parameters

other	system obj to be compare must be passed

Returns

A bool is returned, true if they are equal and false if not

```
                                             {
    return (name == other.name && value == other.value);
    // Compare todos os membros para verificar igualdade
}
```

References name, and value.

4.5.2.5 setName()

```
void System::setName (
            const std::string & name )
```

This method assigns a string to the name of a system.

Parameters

name	string must be passed to the method

```
{ this->name = name; }
```

References name.

4.5.2.6 setValue()

```
void System::setValue (
            const double & value )
```

This method assigns a double to the value of a system.

Parameters

value	double must be passed to the method

```
{ this->value = value; }
```

References value.

4.5.3 Friends And Related Symbol Documentation

4.5.3.1 operator<<

```
std::ostream & operator<< (
              std::ostream & out,
              const System & obj )  [friend]
```

This method is overloading the '<<' operator, print the system obj info.

Parameters

out	is a ostream obj
obj	is a system obj

Returns

a ostream obj to print the obj info

```
                                                        {
    out « "(System)(Name: " « obj.name « ", Value: " « obj.value « ")";
    return out;
}
```

4.5.4 Member Data Documentation

4.5.4.1 name

```
std::string System::name  [protected]
```

Name string attribute.

Referenced by getName(), operator=(), operator==(), and setName().

4.5.4.2 value

```
double System::value  [protected]
```

Value double attribute.

Referenced by getValue(), operator=(), operator==(), and setValue().

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

- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/System.h
- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/System.cpp

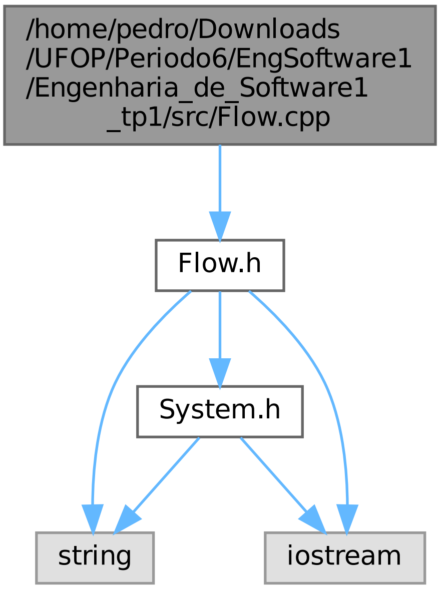
Chapter 5

File Documentation

5.1 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Flow.cpp File Reference

```
#include "Flow.h"
```

Include dependency graph for Flow.cpp:



Functions

- std::ostream & operator<< (std::ostream &out, const Flow &obj)

5.1.1 Function Documentation

5.1.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const Flow & obj )
```

Parameters

out	is a ostream obj
obj	is a flow obj

Returns

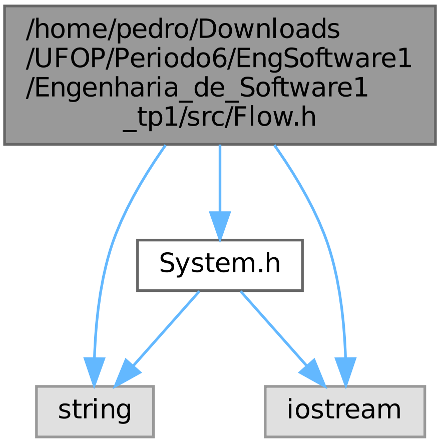
a ostream obj to print the obj info

```
                                                       {
        out « "(Flow) Name: " « obj.name « " - "
            « obj.source->getName() « " -----> " « obj.target->getName();
        return out;
}
```

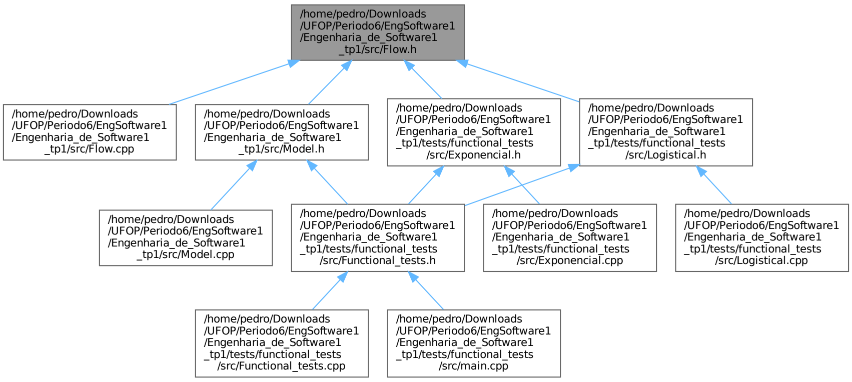
5.2 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Flow.h File Reference

```
#include "System.h"
#include <string>
#include <iostream>
```

Include dependency graph for Flow.h:



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



Classes

- class Flow

5.3 Flow.h

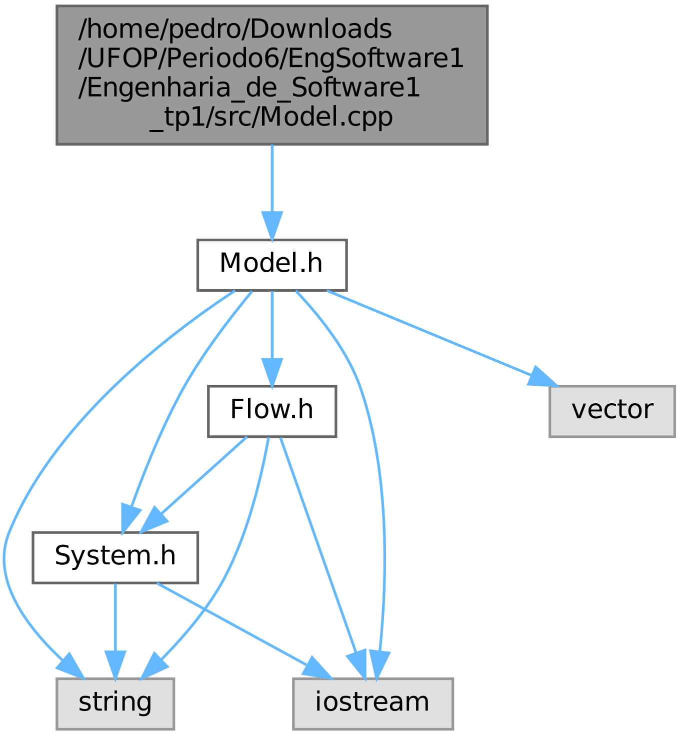
Go to the documentation of this file.

```
/***************************************************
 * @file Flow.h
 * @author Pedro Augusto Sousa Gonçalves
 * @brief This file represents the simulation flow
***************************************************/

#ifndef FLOW_H
#define FLOW_H

#include "System.h"
#include <string>
#include <iostream>

/*************************************************************************************
 *@brief The Flow Interface is the Interface that defines the methods to be implemented
*************************************************************************************/

class Flow{
    protected:
        std::string name;
        System* source;
        System* target;
    public:
        //Geters e seters
        //Name
        std::string getName() const;
        void setName(std::string& name);
        //Source
        System* getSource() const;
        void setSource(System* source);
        //Target
        System* getTarget() const;
        void setTarget(System* target);

        //Metodos
        virtual double execute() = 0;

        //Sobrecarga de operadores
        Flow& operator=(const Flow& other); // Operador de atribuição
        bool operator==(const Flow& other) const; // Operador de igualdade
        friend std::ostream& operator«(std::ostream& out, const Flow& obj); //Operador de saida
};


#endif
```

5.4 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia↩ _de_Software1_tp1/src/Model.cpp File Reference

```
#include "Model.h"
```

Include dependency graph for Model.cpp:


/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/src/Model.cpp
Model.h
Flow.h
vector
System.h
string
iostream

Functions

- std::ostream & operator<< (std::ostream &out, const Model &obj)

5.4.1 Function Documentation

5.4.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const Model & obj )
```

Parameters

out	is a ostream obj
obj	is a model obj

Returns

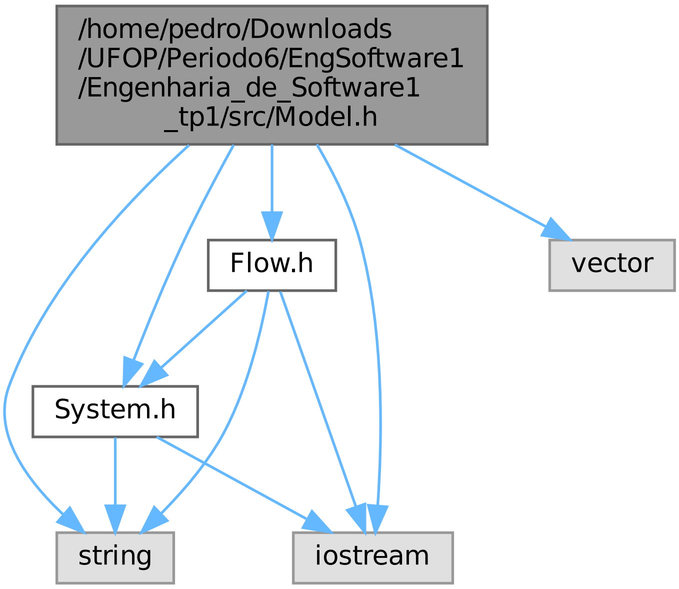
a ostream obj to print the obj info

```
                                                       {
     out « "Name: " « obj.name « ";\n"
         « "Systems:\n";
     for (auto item : obj.systems) out « item « "\n";
     out « "Flows:\n";
     for (auto item : obj.flows) out « item « "\n";
     return out;
}
```

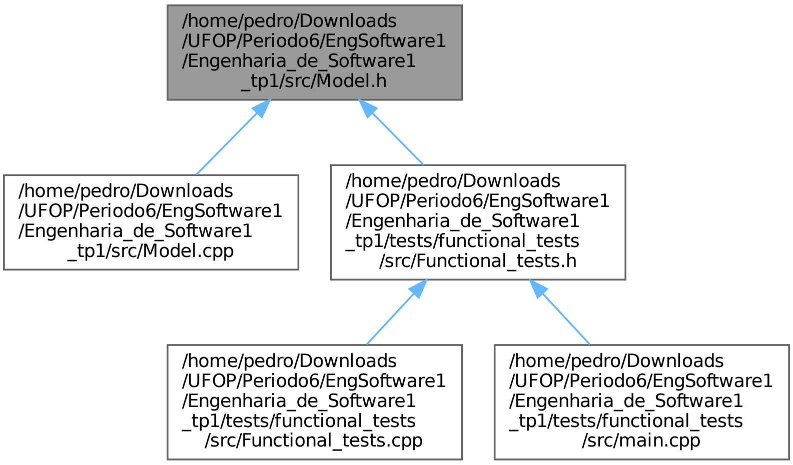
5.5 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.h File Reference

```
#include "System.h"
#include "Flow.h"
#include <string>
#include <iostream>
#include <vector>
```

Include dependency graph for Model.h:



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


/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.h
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.cpp
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_tests/src/Functional_tests.h
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_tests/src/Functional_tests.cpp
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_tests/src/main.cpp

Classes

- class Model

5.6 Model.h

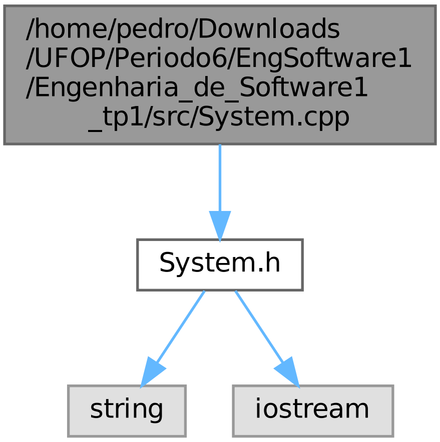
Go to the documentation of this file.

```
/****************************************************
 * @file Model.h
 * @author Pedro Augusto Sousa Gonçalves
 * @brief This file represents the simulation model
****************************************************/

#ifndef MODEL_H
#define MODEL_H

#include "System.h"
#include "Flow.h"
#include <string>
#include <iostream>
#include <vector>


/*****************************************************************************************************************
 *@brief This class represents the general simulation model, it contains figures for simulation and its execution.
*****************************************************************************************************************/
class Model{
    private:
        Model& operator=(const Model& other); // Operador de atribuição
        Model(const Model& other); //Copia outro flow

    protected:
        std::string name;
        std::vector<System*> systems;
        std::vector<Flow*> flows;
        int startTime;
        int endTime;
    public:
        //Iteradores
        typedef std::vector<System*>::iterator systemIterator;
        typedef std::vector<Flow*>::iterator flowIterator;

        //Contructors
        Model(const std::string& name = "NO_NAME", const int& startTime = 0, const int& endTime = 1);

        //Destrutor
        virtual ~Model();

        //Geters e seters
        //Name
        //Nome
        std::string getName() const;
        void setName(const std::string& name);
        //Time
        int getStartTime() const;
        int getEndtTime() const;
        void setStartTime(const int& startTime);
        void setEndTime(const int& endTime);
        void setTime(const int& startTime, const int& endTime);

        //Metodos
        //add
        void add(System* system);
        void add(Flow* flow);
        //remove
        bool rmv(const systemIterator& system);
        bool rmv(const flowIterator& flow);
        //Others
        bool run();

        //Sobrecarga de operadores
        bool operator==(const Model& other) const; // Operador de igualdade
        friend std::ostream& operator«(std::ostream& out, const Model& obj); //Operador de saida
};

#endif
```

5.7 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia↩_de_Software1_tp1/src/System.cpp File Reference

`#include "System.h"`

Include dependency graph for System.cpp:



Functions

- std::ostream & operator<< (std::ostream &out, const System &obj)

5.7.1 Function Documentation

5.7.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const System & obj )
```

Parameters

out	is a ostream obj
obj	is a system obj

Returns

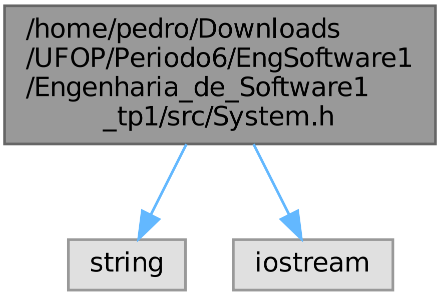
a ostream obj to print the obj info

```
                                                {
    out « "(System)(Name: " « obj.name « ", Value: " « obj.value « ")";
    return out;
}
```

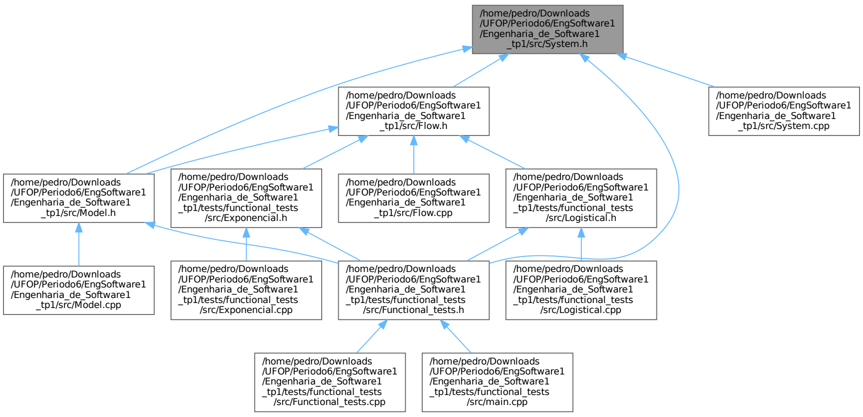
5.8 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/System.h File Reference

```
#include <string>
#include <iostream>
```

Include dependency graph for System.h:



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


/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/src/System.h
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/src/Flow.h
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/src/System.cpp
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/src/Model.h
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/tests/functional_tests /src/Exponencial.h
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/src/Flow.cpp
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/tests/functional_tests /src/Logistical.h
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/src/Model.cpp
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/tests/functional_tests /src/Exponencial.cpp
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/tests/functional_tests /src/Functional_tests.h
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/tests/functional_tests /src/Logistical.cpp
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/tests/functional_tests /src/Functional_tests.cpp
/home/pedro/Downloads /UFOP/Periodo6/EngSoftware1 /Engenharia_de_Software1 _tp1/tests/functional_tests /src/main.cpp

Classes

- class System

5.9 System.h

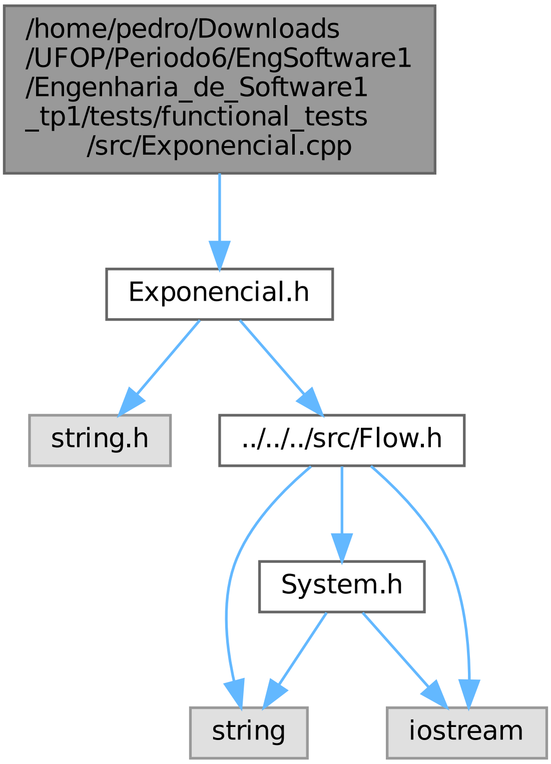
Go to the documentation of this file.

```
/******************************************************
 * @file System.h
 * @author Pedro Augusto Sousa Gonçalves
 * @brief This file represents the simulation system
*****************************************************/

#ifndef SYSTEM_H
#define SYSTEM_H

//Bibliotecas
#include <string>
#include <iostream>

/*******************************************************************************************
 *@brief The System Interface is the Interface that defines the methods to be implemented
******************************************************************************************/

class System{
    protected:
        std::string name;
        double value;
    public:
        //Contructors
        System(const std::string& name = "NO_NAME", const double& value = 0.0);
        System(const System& other); //Copia outro system

        //Destructors
        virtual ~System();

        //Geters e seters
        //Nome
        std::string getName() const;
        void setName(const std::string& name);
        //Value
        double getValue() const;
        void setValue(const double& value);

        //Sobrecarga de operadores
        System& operator=(const System& other);  // Operador de atribuição
        bool operator==(const System& other) const;  // Operador de igualdade
        friend std::ostream& operator«(std::ostream& out, const System& obj); //Operador de saida
};

#endif
```

5.10 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/functional_tests/src/↵ Exponencial.cpp File Reference

```
#include "Exponencial.h"
```

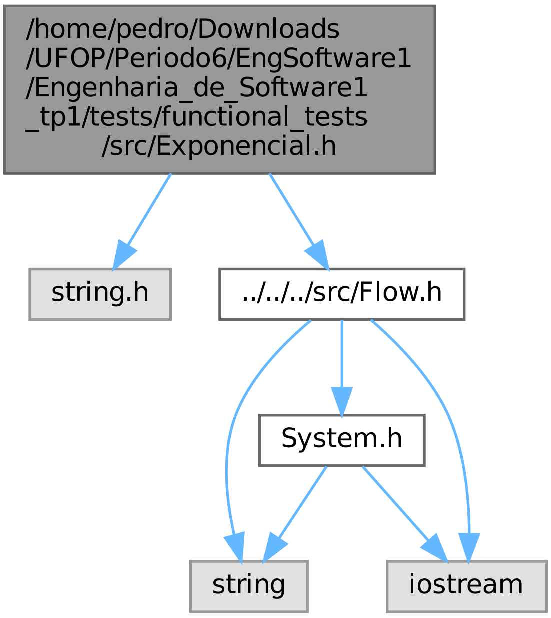
Include dependency graph for Exponencial.cpp:



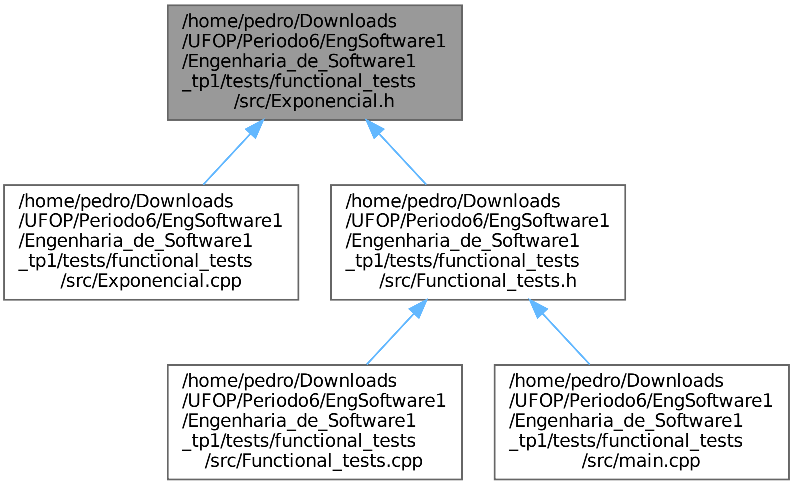
5.11 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/functional_tests/src/↵ Exponencial.h File Reference

```
#include <string.h>
#include "../../../src/Flow.h"
```

Include dependency graph for Exponencial.h:



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



Classes

- class Exponencial

5.12 Exponencial.h

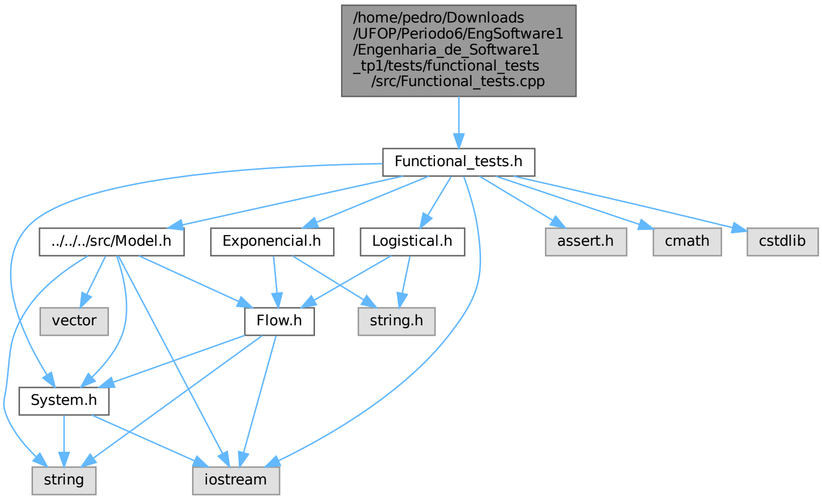
Go to the documentation of this file.

```
/**************************************************************
 * @file Exponencial.h
 * @author Pedro Augusto Sousa Gonçalves
 * @brief This file represents the exponential simulation flow
**************************************************************/

#ifndef EXPONENCIAL_DEF
#define EXPONENCIAL_DEF

#include <string.h>
#include "../../../src/Flow.h"

/*************************************************************************************************************************
 * @brief This Flow class connects two systems and through the entered equation transfers values from one system to another
*************************************************************************************************************************
class Exponencial : public Flow{
    public:
        //Contructor
        Exponencial(const std::string& name = "NO_NAME", System* source = NULL, System* target = NULL);
        Exponencial(const Exponencial& other);

        //Destructor
        virtual ~Exponencial();

        //Metodos
        virtual double execute() override;
};

#endif
```

5.13 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/functional_tests/src/↩ Functional_tests.cpp File Reference

`#include "Functional_tests.h"`

Include dependency graph for Functional_tests.cpp:



Functions

- void exponencial_test_run ()

  This function performs the exponential functional test.
- void logistical_test_run ()

  This function performs the logistic test.
- void Complex_test_run ()

  This function runs the "complex" test, which has multiple systems and flows.

5.13.1 Function Documentation

5.13.1.1 Complex_test_run()

```
void Complex_test_run ( )
```

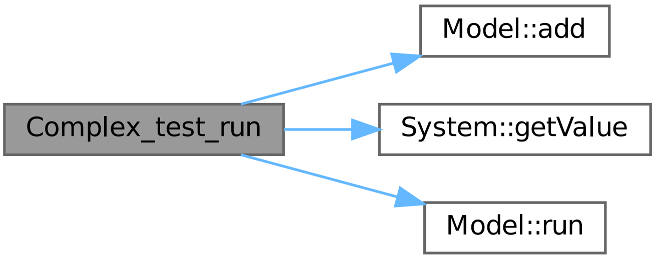
This function runs the "complex" test, which has multiple systems and flows.

```
                              {
    std::cout « "Complex funcional test" « std::endl;

    Model* model = new Model("Model", 0, 100);
    System* q1 = new System("q1", 100.0);
    System* q2 = new System("q2", 0.0);
    System* q3 = new System("q3", 100.0);
    System* q4 = new System("q4", 0.0);
    System* q5 = new System("q5", 0.0);
    Exponencial* f = new Exponencial("f", q1, q2);
    Exponencial* t = new Exponencial("t", q2, q3);
    Exponencial* u = new Exponencial("u", q3, q4);
    Exponencial* v = new Exponencial("v", q4, q1);
    Exponencial* g = new Exponencial("g", q1, q3);
    Exponencial* r = new Exponencial("r", q2, q5);

    model->add(q1);
    model->add(q2);
    model->add(q3);
    model->add(q4);
    model->add(q5);
    model->add(f);
    model->add(t);
    model->add(u);
    model->add(v);
    model->add(g);
    model->add(r);

    model->run();

    assert(fabs((round((q1->getValue() * 10000)) - 10000 * 31.8513)) < 0.0001);
    assert(fabs((round((q2->getValue() * 10000)) - 10000 * 18.4003)) < 0.0001);
    assert(fabs((round((q3->getValue() * 10000)) - 10000 * 77.1143)) < 0.0001);
    assert(fabs((round((q4->getValue() * 10000)) - 10000 * 56.1728)) < 0.0001);
    assert(fabs((round((q5->getValue() * 10000)) - 10000 * 16.4612)) < 0.0001);

    delete model;
    delete q1;
    delete q2;
    delete q3;
    delete q4;
    delete q5;
    delete f;
    delete t;
    delete u;
    delete v;
    delete g;
    delete r;

    std::cout « "Passed Complex funcional test" « std::endl;
}
```

References Model::add(), System::getValue(), and Model::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

main
Complex_test_run

5.13.1.2 exponencial_test_run()

```
void exponencial_test_run ( )
```

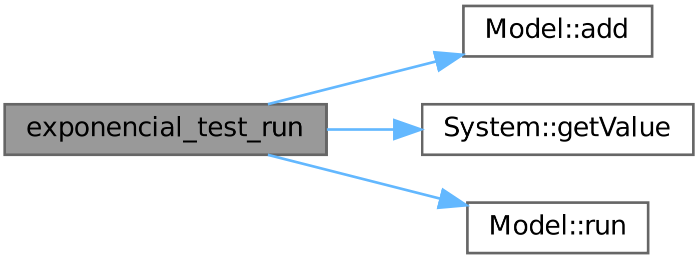
This function performs the exponential functional test.

```
                           {
    std::cout « "Exponencial funcional test" « std::endl;

    System* pop1 = new System("pop1", 100.0);
    System* pop2 = new System("pop2", 0.0);
    Exponencial* exp = new Exponencial("exp", pop1, pop2);
    Model* exponencial = new Model("Exponencial", 0, 100);

    //Add os systems e flows ao modelo
    exponencial->add(pop1);
    exponencial->add(pop2);
    exponencial->add(exp);

    //Roda o modelo
    exponencial->run();

    assert(fabs((round(pop1->getValue() * 10000) - 10000 * 36.6032)) < 0.0001);
    assert(fabs((round(pop2->getValue() * 10000) - 10000 * 63.3968)) < 0.0001);

    delete(exponencial);
    delete(exp);
    delete(pop1);
    delete(pop2);

    std::cout « "Passed exponencial funcional test" « std::endl;
}
```

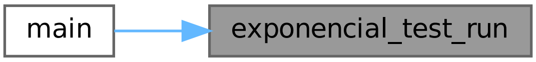
References Model::add(), System::getValue(), and Model::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.1.3 logistical_test_run()

```
void logistical_test_run ( )
```

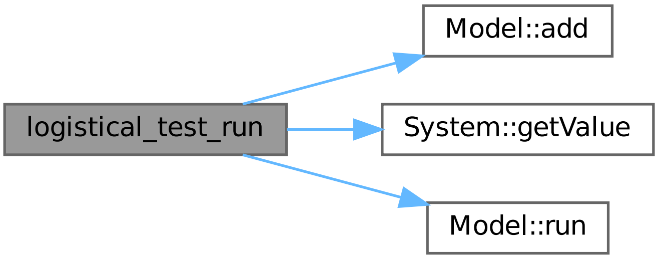
This function performs the logistic test.

```
                          {
    std::cout « "Logistical funcional test" « std::endl;

    System* p1 = new System("p1", 100.0);
    System* p2 = new System("p2", 10.0);
    Logistical* log = new Logistical("log", p1, p2);
    Model* logistical = new Model("Logistical", 0, 100);

    //Add os systems e flows ao modelo
    logistical->add(p1);
    logistical->add(p2);
    logistical->add(log);

    //Roda o modelo
    logistical->run();

    assert(fabs(round(p1->getValue() * 10000) - 10000 * 88.2167) < 0.0001);
    assert(fabs(round(p2->getValue() * 10000) - 10000 * 21.7833) < 0.0001);

    delete(logistical);
    delete(log);
    delete(p1);
    delete(p2);
}
```

References Model::add(), System::getValue(), and Model::run().

Referenced by main().

Here is the call graph for this function:



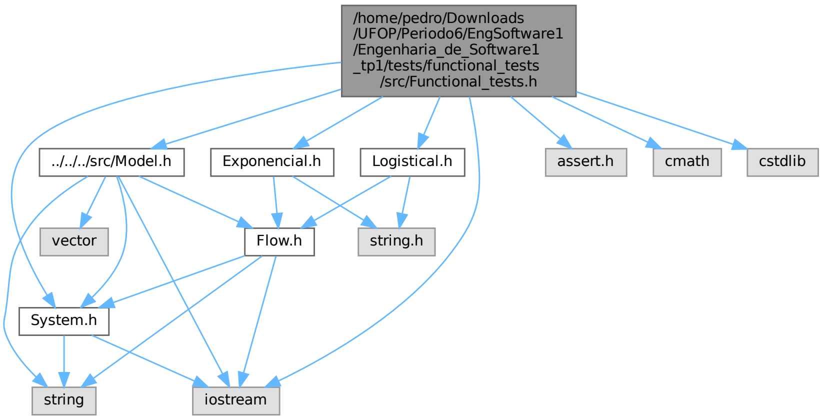
Here is the caller graph for this function:

main
logistical_test_run

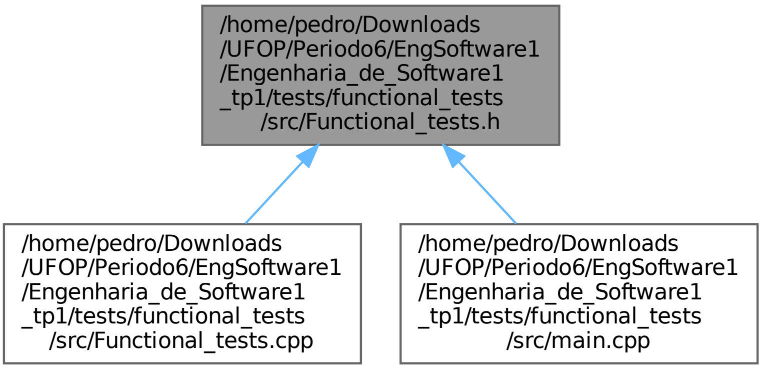
5.14 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/functional_tests/src/↩ Functional_tests.h File Reference

```
#include "../../../src/Model.h"
#include "../../../src/System.h"
#include "Exponencial.h"
#include "Logistical.h"
#include <assert.h>
#include <cmath>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for Functional_tests.h:



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


/home/pedro/Downloads
/UFOP/Periodo6/EngSoftware1
/Engenharia_de_Software1
_tp1/tests/functional_tests
/src/Functional_tests.h
/home/pedro/Downloads
/UFOP/Periodo6/EngSoftware1
/Engenharia_de_Software1
_tp1/tests/functional_tests
/src/Functional_tests.cpp
/home/pedro/Downloads
/UFOP/Periodo6/EngSoftware1
/Engenharia_de_Software1
_tp1/tests/functional_tests
/src/main.cpp

Functions

- void exponencial_test_run ()

  This function performs the exponential functional test.
- void logistical_test_run ()

  This function performs the logistic test.
- void Complex_test_run ()

  This function runs the "complex" test, which has multiple systems and flows.

5.14.1 Function Documentation

5.14.1.1 Complex_test_run()

```
void Complex_test_run ( )
```

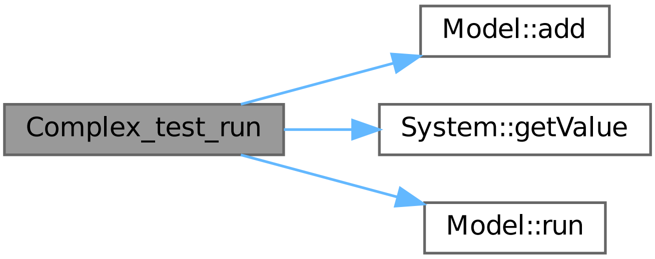
This function runs the "complex" test, which has multiple systems and flows.

```
                       {
    std::cout « "Complex funcional test" « std::endl;

    Model* model = new Model("Model", 0, 100);
    System* q1 = new System("q1", 100.0);
    System* q2 = new System("q2", 0.0);
    System* q3 = new System("q3", 100.0);
    System* q4 = new System("q4", 0.0);
    System* q5 = new System("q5", 0.0);
    Exponencial* f = new Exponencial("f", q1, q2);
    Exponencial* t = new Exponencial("t", q2, q3);
    Exponencial* u = new Exponencial("u", q3, q4);
    Exponencial* v = new Exponencial("v", q4, q1);
    Exponencial* g = new Exponencial("g", q1, q3);
    Exponencial* r = new Exponencial("r", q2, q5);

    model->add(q1);
    model->add(q2);
    model->add(q3);
    model->add(q4);
    model->add(q5);
    model->add(f);
    model->add(t);
    model->add(u);
    model->add(v);
    model->add(g);
    model->add(r);

    model->run();

    assert(fabs((round((q1->getValue() * 10000)) - 10000 * 31.8513)) < 0.0001);
    assert(fabs((round((q2->getValue() * 10000)) - 10000 * 18.4003)) < 0.0001);
    assert(fabs((round((q3->getValue() * 10000)) - 10000 * 77.1143)) < 0.0001);
    assert(fabs((round((q4->getValue() * 10000)) - 10000 * 56.1728)) < 0.0001);
    assert(fabs((round((q5->getValue() * 10000)) - 10000 * 16.4612)) < 0.0001);

    delete model;
    delete q1;
    delete q2;
    delete q3;
    delete q4;
    delete q5;
    delete f;
    delete t;
    delete u;
    delete v;
    delete g;
    delete r;

    std::cout « "Passed Complex funcional test" « std::endl;
}
```

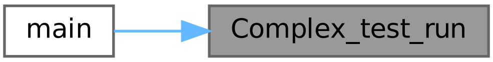
References Model::add(), System::getValue(), and Model::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.1.2 exponencial_test_run()

```
void exponencial_test_run ( )
```

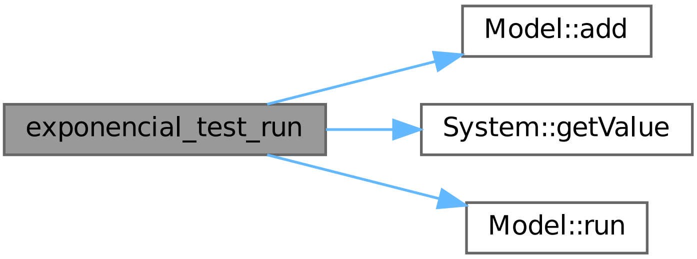
This function performs the exponential functional test.

```
                           {
    std::cout « "Exponencial funcional test" « std::endl;

    System* pop1 = new System("pop1", 100.0);
    System* pop2 = new System("pop2", 0.0);
    Exponencial* exp = new Exponencial("exp", pop1, pop2);
    Model* exponencial = new Model("Exponencial", 0, 100);

    //Add os systems e flows ao modelo
    exponencial->add(pop1);
    exponencial->add(pop2);
    exponencial->add(exp);

    //Roda o modelo
    exponencial->run();

    assert(fabs((round(pop1->getValue() * 10000) - 10000 * 36.6032)) < 0.0001);
    assert(fabs((round(pop2->getValue() * 10000) - 10000 * 63.3968)) < 0.0001);

    delete(exponencial);
    delete(exp);
    delete(pop1);
    delete(pop2);

    std::cout « "Passed exponencial funcional test" « std::endl;
}
```

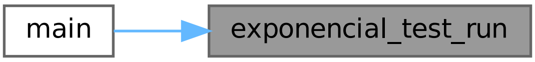
References Model::add(), System::getValue(), and Model::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.1.3 logistical_test_run()

```
void logistical_test_run ( )
```

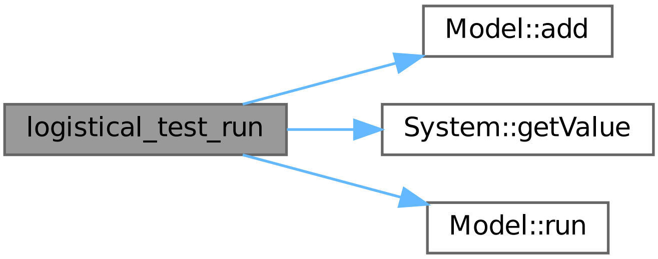
This function performs the logistic test.

```
                             {
    std::cout « "Logistical funcional test" « std::endl;

    System* p1 = new System("p1", 100.0);
    System* p2 = new System("p2", 10.0);
    Logistical* log = new Logistical("log", p1, p2);
    Model* logistical = new Model("Logistical", 0, 100);

    //Add os systems e flows ao modelo
    logistical->add(p1);
    logistical->add(p2);
    logistical->add(log);

    //Roda o modelo
    logistical->run();

    assert(fabs(round(p1->getValue() * 10000) - 10000 * 88.2167) < 0.0001);
    assert(fabs(round(p2->getValue() * 10000) - 10000 * 21.7833) < 0.0001);

    delete(logistical);
    delete(log);
    delete(p1);
    delete(p2);
}
```

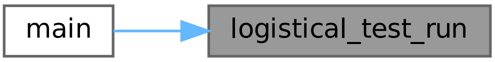
References Model::add(), System::getValue(), and Model::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.15 Functional_tests.h

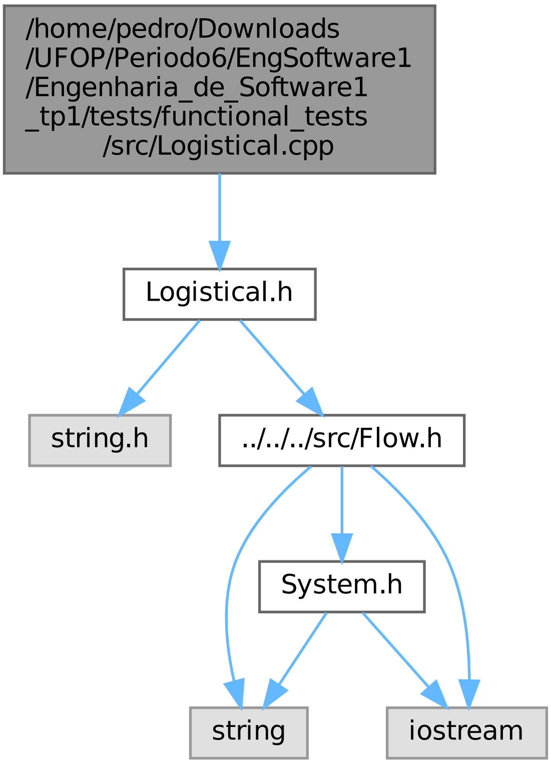
Go to the documentation of this file.

```
/****************************************************************
 * @file Exponencial.h
 * @author Pedro Augusto Sousa Gonçalves
 * @brief This file represents the logistical simulation flow
*****************************************************************/

#ifndef FUNCTIONAL_TESTS_H
#define FUNCTIONAL_TESTS_H

#include "../../../src/Model.h"
#include "../../../src/System.h"
#include "Exponencial.h"
#include "Logistical.h"
#include <assert.h>
#include <cmath>
#include <iostream>
#include <cstdlib>

/*************************************
 * @brief execution of functional tests
*************************************/

void exponencial_test_run();

void logistical_test_run();

void Complex_test_run();

#endif
```

5.16 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/functional_tests/src/↵ Logistical.cpp File Reference

```
#include "Logistical.h"
```

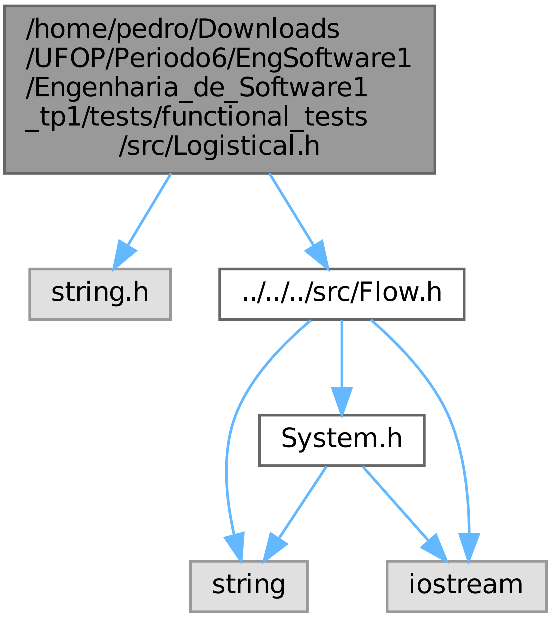
Include dependency graph for Logistical.cpp:



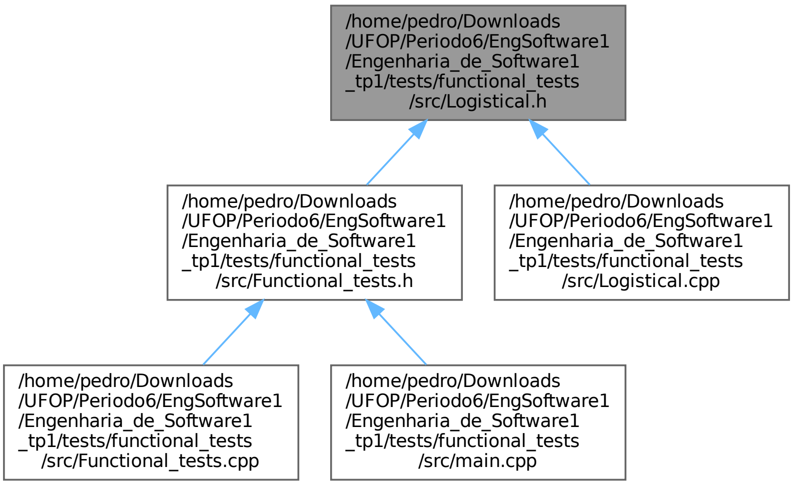
5.17 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/functional_tests/src/↵ Logistical.h File Reference

```
#include <string.h>
#include "../../../src/Flow.h"
```

Include dependency graph for Logistical.h:



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



Classes

- class Logistical

5.18 Logistical.h

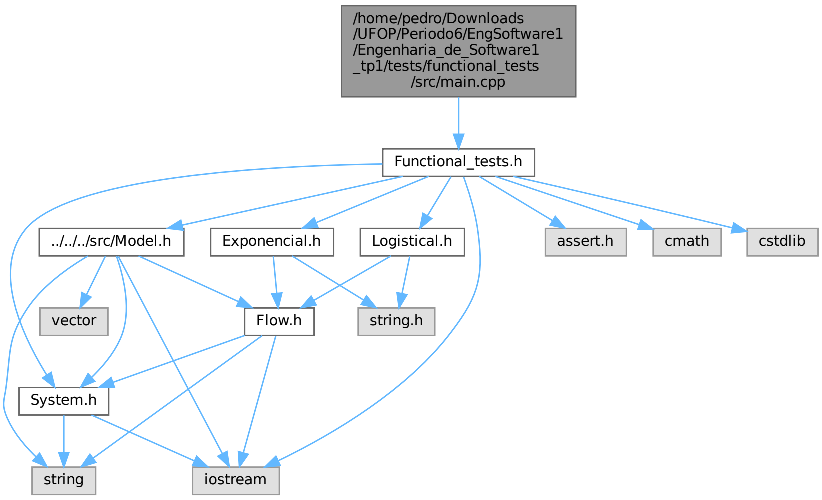
Go to the documentation of this file.

```
/*****************************************************************
 * @file Exponencial.h
 * @author Pedro Augusto Sousa Gonçalves
 * @brief This file represents the logistical simulation flow
*****************************************************************/

#ifndef LOGISTICAL_DEF
#define LOGISTICAL_DEF

#include <string.h>
#include "../../../src/Flow.h"

class Logistical : public Flow{
    public:
        //Contructor
        Logistical(const std::string& name = "NO_NAME", System* source = NULL, System* target = NULL);
        Logistical(const Logistical& other);

        //Destructor
        virtual ~Logistical();

        //Metodos
        virtual double execute() override;
};

#endif
```

5.19 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_tests/src/main.cpp File Reference

```
#include "Functional_tests.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

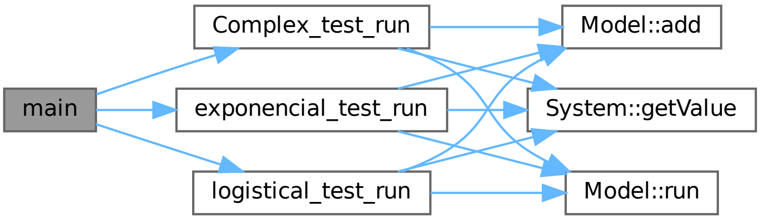
5.19.1 Function Documentation

5.19.1.1 main()

```
int main ( )
               {
         exponencial_test_run();
         logistical_test_run();
         Complex_test_run();
         return 0;
 }
```

References Complex_test_run(), exponencial_test_run(), and logistical_test_run().

Here is the call graph for this function:


Complex_test_run
Model::add
main
exponencial_test_run
System::getValue
logistical_test_run
Model::run