

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:

Flow	??
FlowIMP	??
Exponencial	??
Flow_unit_test	??
Logistical	??
Model	??
ModelIMP	??
System	??
SystemIMP	??

Chapter 2

Class Index

2.1 Class List

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

Exponencial	??
Flow	??
Flow_unit_test	??
FlowIMP	??
Logistical	??
Model	??
ModelIMP	??
System	??
SystemIMP	??

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Flow.hpp . .	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/FlowIMP.cpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/FlowIMP.hpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.hpp .	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/ModelIMP.cpp	
??	
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/ModelIMP.hpp	
??	
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/System.hpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/SystemIMP.cpp	
??	
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/SystemIMP.hpp	
??	
/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/Exponencial.hpp	??
/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/Functional_tests.hpp	??
/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/Logistical.hpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional↔ _tests/src/main.cpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↔ tests/src/main.cpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↔ tests/src/unit_Flow.cpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↔ tests/src/unit_Flow.hpp	??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↔ tests/src/unit_Model.cpp	??

/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↵
tests/src/unit_Model.hpp ??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↵
tests/src/unit_System.cpp ??
/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↵
tests/src/unit_System.hpp ??

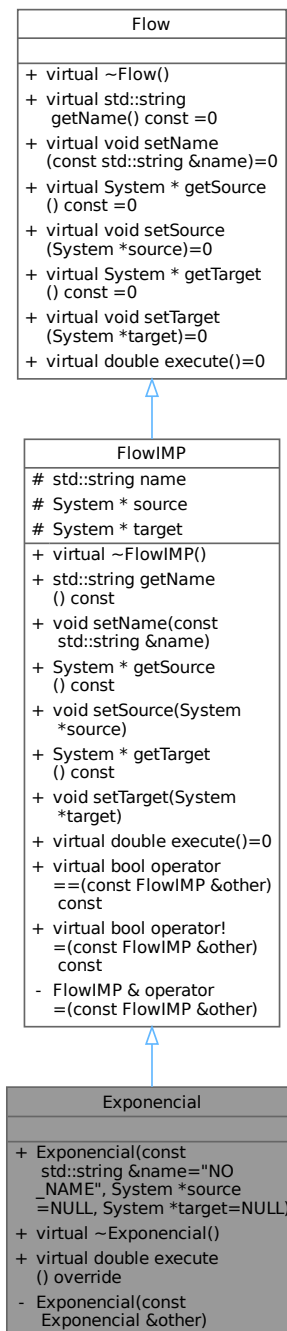
Chapter 4

Class Documentation

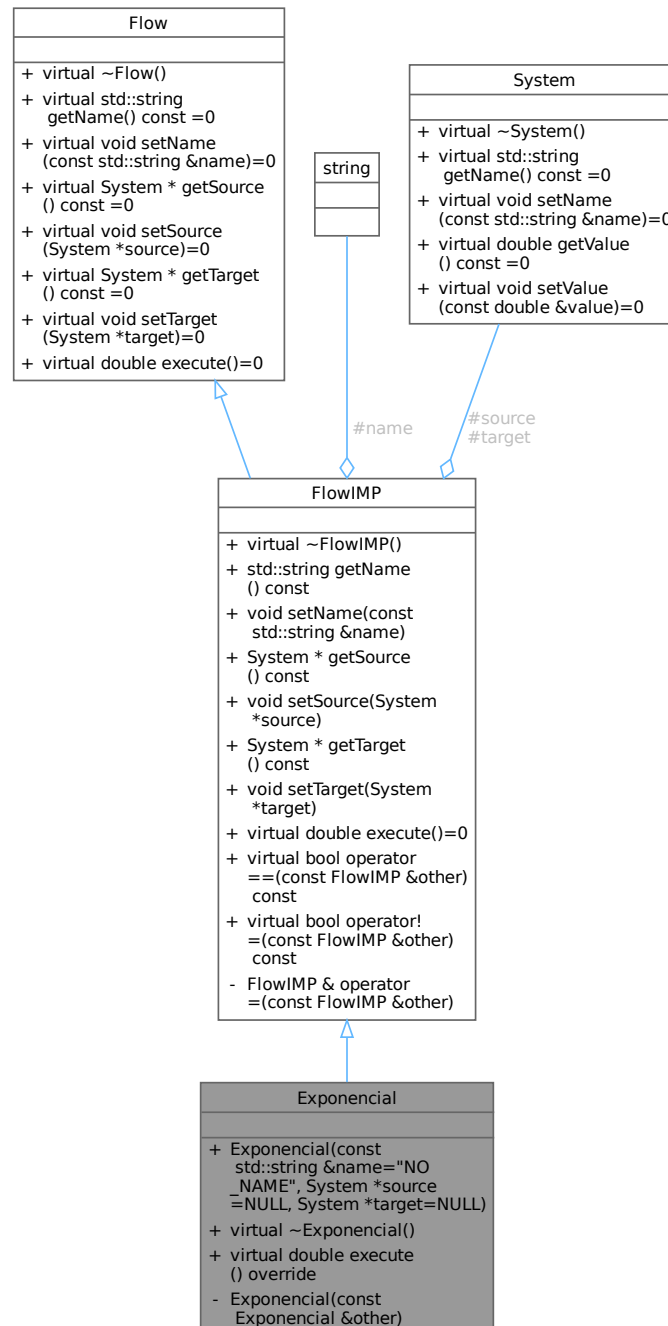
4.1 Exponencial Class Reference

```
#include <Exponencial.hpp>
```

Inheritance diagram for Exponential:



Collaboration diagram for Exponential:



Public Member Functions

- **Exponential** (const std::string &name="NO_NAME", System *source=NULL, System *target=NULL)
Construct a new **Exponential** by name, source and target.
- virtual ~**Exponential** ()
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 [FlowIMP](#)

- virtual [~FlowIMP](#) ()
This destructor is a virtual destructor of the class.
- std::string [getName](#) () const
This method returns the name of a flow.
- void [setName](#) (const std::string &[name](#))
This method assigns a string to the name of a flow obj.
- [System](#) * [getSource](#) () const
This method returns the source system pointer.
- void [setSource](#) ([System](#) *[source](#))
This method assigns a system pointer to the source of a flow obj.
- [System](#) * [getTarget](#) () const
This method returns the target system pointer.
- void [setTarget](#) ([System](#) *[target](#))
This method assigns a system pointer to the target of a flow obj.
- virtual bool [operator==](#) (const [FlowIMP](#) &[other](#)) const
This method is overloading the '==' operator, compare two flows objs.
- virtual bool [operator!=](#) (const [FlowIMP](#) &[other](#)) const
This method is overloading the '!=' operator, compare two flows objs.

Public Member Functions inherited from [Flow](#)

- virtual [~Flow](#) ()
This destructor is a virtual destructor of the class.

Private Member Functions

- [Exponencial](#) (const [Exponencial](#) &[other](#))
Construct a new [Exponencial](#) by a obj.

Additional Inherited Members

Protected Attributes inherited from [FlowIMP](#)

- std::string [name](#)
- [System](#) * [source](#)
- [System](#) * [target](#)

4.1.1 Constructor & Destructor Documentation

4.1.1.1 [Exponencial\(\)](#) [1/2]

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

Construct a new [Exponencial](#) by a obj.

Parameters

<i>other</i>	Exponential obj
--------------	---------------------------------

```

00011                                     {
00012     this->name = other.name;
00013     this->source = other.source;
00014     this->target = other.target;
00015 }
```

References [FlowIMP::name](#), [FlowIMP::source](#), and [FlowIMP::target](#).

4.1.1.2 Exponential() [2/2]

```

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

Construct a new [Exponential](#) by name, source and target.

Parameters

<i>name</i>	string with default value "NO_NAME"
<i>source</i>	System pointer with default value NULL
<i>target</i>	System pointer with default value NULL

```

00004                                     {
00005     this->name = name;
00006     this->source = source;
00007     this->target = target;
00008 }
```

References [FlowIMP::name](#), [FlowIMP::source](#), and [FlowIMP::target](#).

4.1.1.3 ~Exponential()

```

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

This destructor is a virtual destructor of the Class.

```

00018 {}
```

4.1.2 Member Function Documentation

4.1.2.1 execute()

```

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

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

Returns

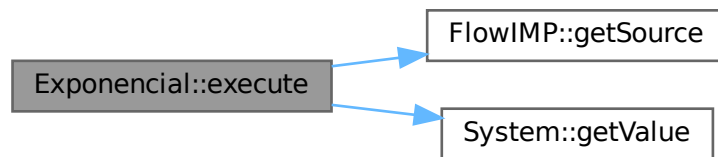
double

Implements [FlowIMP](#).

```
00020 {  
00021     return getSource()->getValue() * 0.01;  
00022 }
```

References [FlowIMP::getSource\(\)](#), and [System::getValue\(\)](#).

Here is the call graph for this function:



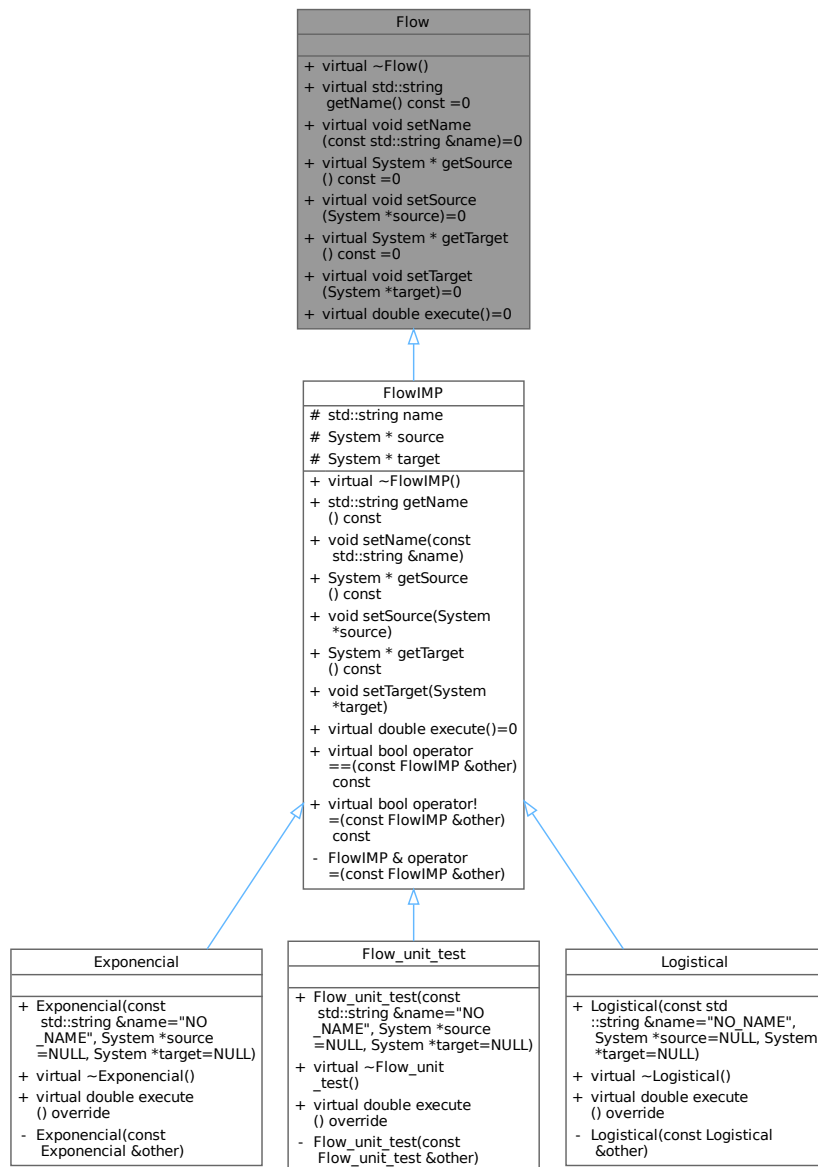
The documentation for this class was generated from the following files:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↔ tests/src/Exponencial.hpp](#)
- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↔ tests/src/Exponencial.cpp](#)

4.2 Flow Class Reference

```
#include <Flow.hpp>
```

Inheritance diagram for Flow:



Collaboration diagram for Flow:

Flow
<ul style="list-style-type: none"> + virtual ~Flow() + virtual std::string getName() const =0 + virtual void setName (const std::string &name)=0 + virtual System * getSource () const =0 + virtual void setSource (System *source)=0 + virtual System * getTarget () const =0 + virtual void setTarget (System *target)=0 + virtual double execute()=0

Public Member Functions

- virtual [~Flow](#) ()
This destructor is a virtual destructor of the class.
- virtual std::string [getName](#) () const =0
This method returns the name of a flow.
- virtual void [setName](#) (const std::string &name)=0
This method assigns a string to the name of a flow obj.
- virtual [System](#) * [getSource](#) () const =0
This method returns the source system poiter.
- virtual void [setSource](#) ([System](#) *source)=0
This method assigns a system poiter to the source of a flow obj.
- virtual [System](#) * [getTarget](#) () const =0
This method returns the target system poiter.
- virtual void [setTarget](#) ([System](#) *target)=0
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.

4.2.1 Constructor & Destructor Documentation

4.2.1.1 ~Flow()

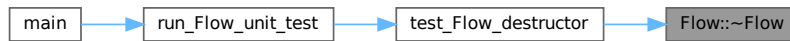
```
virtual Flow::~~Flow ( ) [inline], [virtual]
```


This destructor is a virtual destructor of the class.

```
00027 {};
```

Referenced by [test_Flow_destructor\(\)](#).

Here is the caller graph for this function:



4.2.2 Member Function Documentation

4.2.2.1 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](#), [Logistical](#), [Flow_unit_test](#), and [FlowIMP](#).

Referenced by [test_Flow_execute\(\)](#).

Here is the caller graph for this function:



4.2.2.2 getName()

```
virtual std::string Flow::getName ( ) const [pure virtual]
```

This method returns the name of a flow.

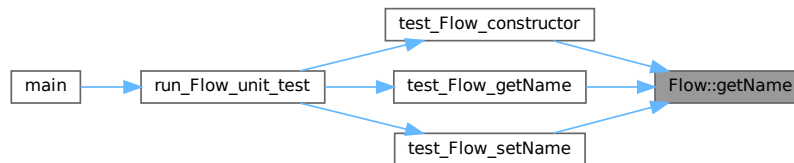
Returns

a string containing the name is returned

Implemented in [FlowIMP](#).

Referenced by [test_Flow_constructor\(\)](#), [test_Flow_getName\(\)](#), and [test_Flow_setName\(\)](#).

Here is the caller graph for this function:

**4.2.2.3 getSource()**

```
virtual System * Flow::getSource ( ) const [pure virtual]
```

This method returns the source system pointer.

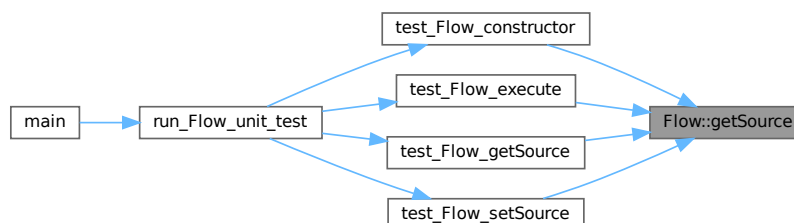
Returns

a system pointer containing the source memory address is returned

Implemented in [FlowIMP](#).

Referenced by [test_Flow_constructor\(\)](#), [test_Flow_execute\(\)](#), [test_Flow_getSource\(\)](#), and [test_Flow_setSource\(\)](#).

Here is the caller graph for this function:



4.2.2.4 getTarget()

```
virtual System * Flow::getTarget ( ) const [pure virtual]
```

This method returns the target system pointer.

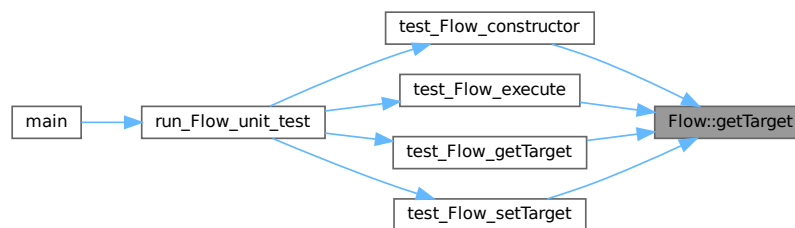
Returns

a system pointer containing the target memory address is returned

Implemented in [FlowIMP](#).

Referenced by [test_Flow_constructor\(\)](#), [test_Flow_execute\(\)](#), [test_Flow_getTarget\(\)](#), and [test_Flow_setTarget\(\)](#).

Here is the caller graph for this function:



4.2.2.5 setName()

```
virtual void Flow::setName (
    const std::string & name ) [pure virtual]
```

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

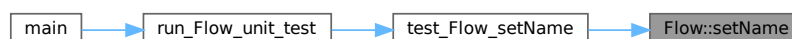
Parameters

<i>name</i>	string must be passed to the method
-------------	-------------------------------------

Implemented in [FlowIMP](#).

Referenced by [test_Flow_setName\(\)](#).

Here is the caller graph for this function:



4.2.2.6 setSource()

```
virtual void Flow::setSource (
    System * source ) [pure virtual]
```

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

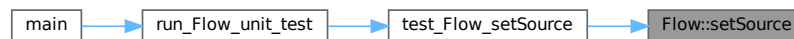
Parameters

<i>source</i>	system pointer must be passed to the method
---------------	---

Implemented in [FlowIMP](#).

Referenced by [test_Flow_setSource\(\)](#).

Here is the caller graph for this function:



4.2.2.7 setTarget()

```
virtual void Flow::setTarget (
    System * target ) [pure virtual]
```

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

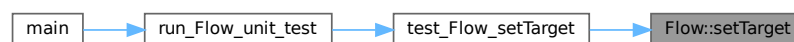
Parameters

<i>target</i>	system pointer must be passed to the method
---------------	---

Implemented in [FlowIMP](#).

Referenced by [test_Flow_setTarget\(\)](#).

Here is the caller graph for this function:



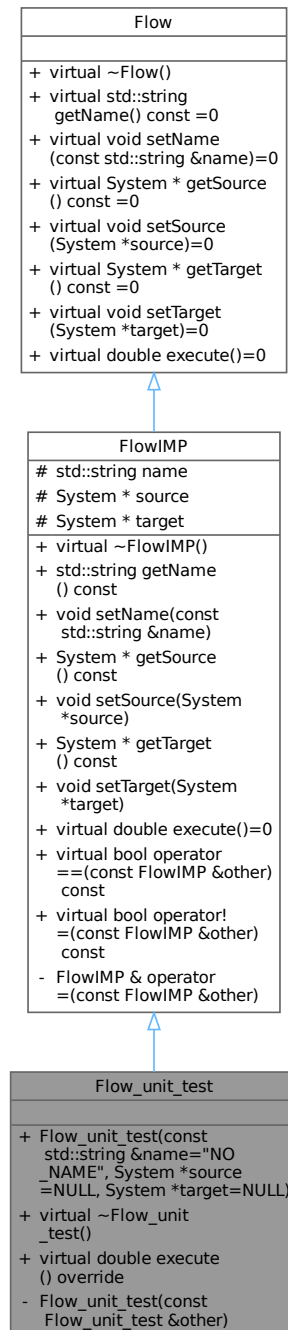
The documentation for this class was generated from the following file:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Flow.hpp](#)

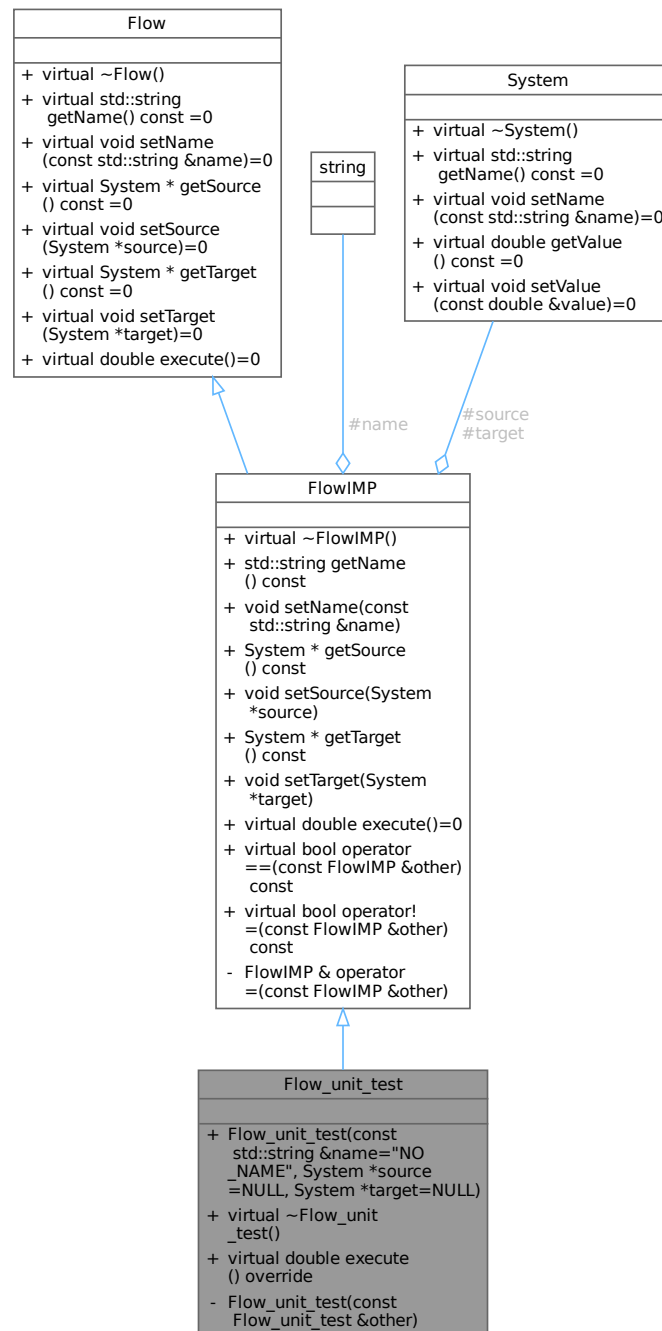
4.3 Flow_unit_test Class Reference

```
#include <unit_Flow.hpp>
```

Inheritance diagram for Flow_unit_test:



Collaboration diagram for Flow_unit_test:



Public Member Functions

- **Flow_unit_test** (const std::string &name="NO_NAME", System *source=NULL, System *target=NULL)
Construct a new **Flow_unit_test** by name, source and target.
- virtual **~Flow_unit_test** ()
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 FlowIMP

- virtual `~FlowIMP()`
This destructor is a virtual destructor of the class.
- `std::string getName()` const
This method returns the name of a flow.
- `void setName(const std::string &name)`
This method assigns a string to the name of a flow obj.
- `System * getSource()` const
This method returns the source system pointer.
- `void setSource(System *source)`
This method assigns a system pointer to the source of a flow obj.
- `System * getTarget()` const
This method returns the target system pointer.
- `void setTarget(System *target)`
This method assigns a system pointer to the target of a flow obj.
- virtual `bool operator==(const FlowIMP &other)` const
This method is overloading the '=' operator, compare two flows objs.
- virtual `bool operator!=(const FlowIMP &other)` const
This method is overloading the '!=' operator, compare two flows objs.

Public Member Functions inherited from Flow

- virtual `~Flow()`
This destructor is a virtual destructor of the class.

Private Member Functions

- `Flow_unit_test(const Flow_unit_test &other)`
Construct a new Exponencial by a obj.

Additional Inherited Members

Protected Attributes inherited from FlowIMP

- `std::string name`
- `System * source`
- `System * target`

4.3.1 Constructor & Destructor Documentation

4.3.1.1 Flow_unit_test() [1/2]

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

Construct a new Exponencial by a obj.

Parameters

<i>other</i>	Exponential obj
--------------	---------------------------------

```

00010                                     {
00011     this->name = other.name;
00012     this->source = other.source;
00013     this->target = other.target;
00014 }
```

References [FlowIMP::name](#), [FlowIMP::source](#), and [FlowIMP::target](#).

4.3.1.2 Flow_unit_test() [2/2]

```

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

Construct a new [Flow_unit_test](#) by name, source and target.

Parameters

<i>name</i>	string with default value "NO_NAME"
<i>source</i>	System pointer with default value NULL
<i>target</i>	System pointer with default value NULL

```

00003                                     {
00004     this->name = name;
00005     this->source = source;
00006     this->target = target;
00007 }
```

References [FlowIMP::name](#), [FlowIMP::source](#), and [FlowIMP::target](#).

4.3.1.3 ~Flow_unit_test()

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

This destructor is a virtual destructor of the Class.

```
00017 {}
```

4.3.2 Member Function Documentation

4.3.2.1 execute()

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

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

Returns

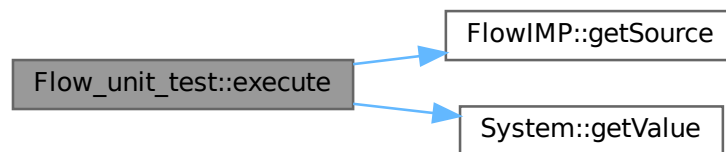
double

Implements [FlowIMP](#).

```
00019 {  
00020     return getSource()->getValue();  
00021 }
```

References [FlowIMP::getSource\(\)](#), and [System::getValue\(\)](#).

Here is the call graph for this function:



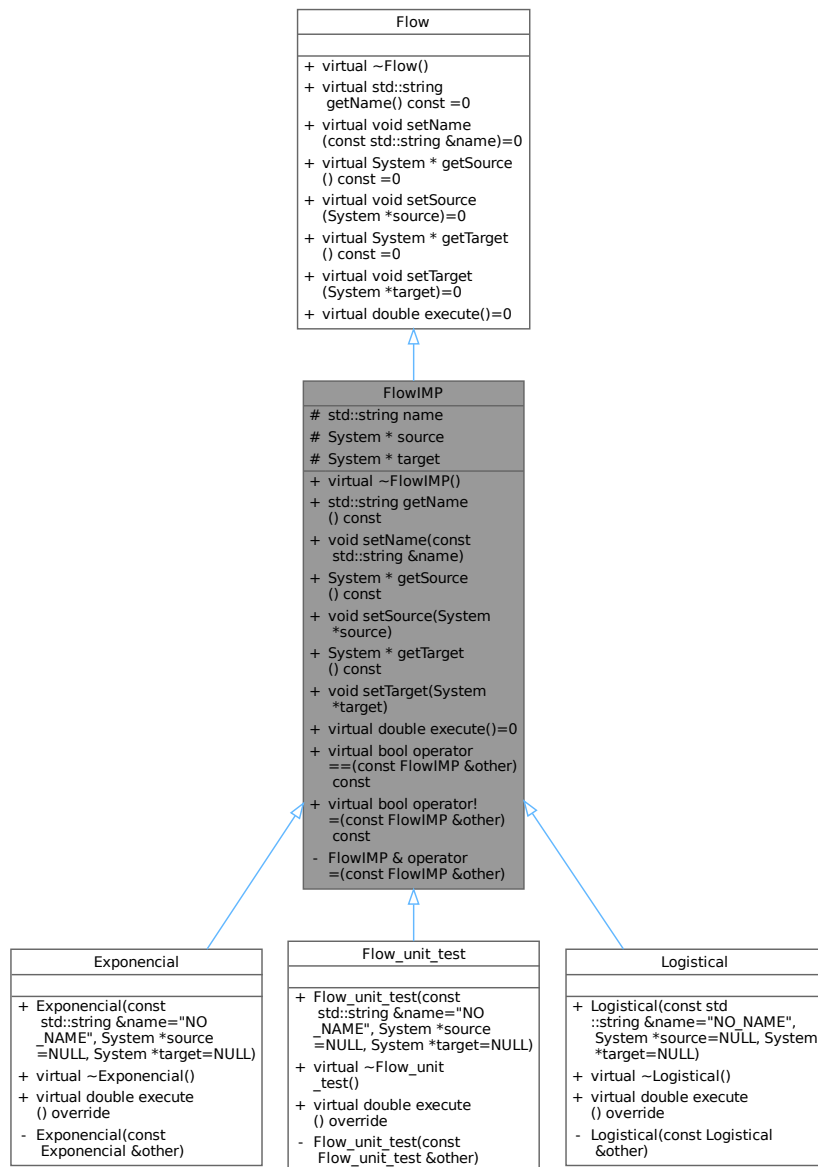
The documentation for this class was generated from the following files:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↔ tests/src/unit_Flow.hpp](#)
- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_↔ tests/src/unit_Flow.cpp](#)

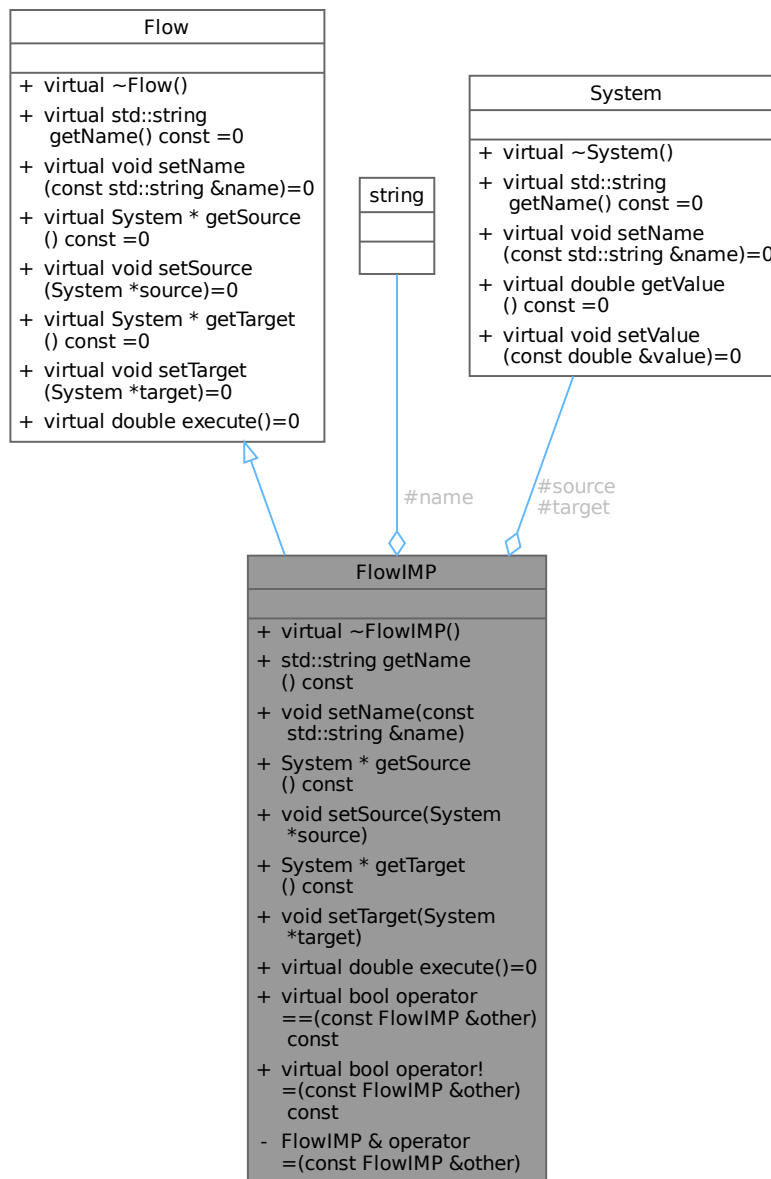
4.4 FlowIMP Class Reference

```
#include <FlowIMP.hpp>
```

Inheritance diagram for FlowIMP:



Collaboration diagram for FlowIMP:



Public Member Functions

- `virtual ~FlowIMP ()`
This destructor is a virtual destructor of the class.
- `std::string getName () const`
This method returns the name of a flow.
- `void setName (const std::string &name)`
This method assigns a string to the name of a flow obj.
- `System * getSource () const`
This method returns the source system pointer.

- void `setSource (System *source)`
This method assigns a system pointer to the source of a flow obj.
- `System * getTarget ()` const
This method returns the target system pointer.
- void `setTarget (System *target)`
This method assigns a system pointer 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.
- virtual bool `operator== (const FlowIMP &other)` const
This method is overloading the '==' operator, compare two flows objs.
- virtual bool `operator!= (const FlowIMP &other)` const
This method is overloading the '!=' operator, compare two flows objs.

Public Member Functions inherited from `Flow`

- virtual `~Flow ()`
This destructor is a virtual destructor of the class.

Protected Attributes

- `std::string name`
- `System * source`
- `System * target`

Private Member Functions

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

4.4.1 Constructor & Destructor Documentation

4.4.1.1 `~FlowIMP()`

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

This destructor is a virtual destructor of the class.

```
00004 {}
```

4.4.2 Member Function Documentation

4.4.2.1 `execute()`

```
virtual double FlowIMP::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

Implements `Flow`.

Implemented in `Exponencial`, `Logistical`, and `Flow_unit_test`.

4.4.2.2 getName()

```
std::string FlowIMP::getName ( ) const [virtual]
```

This method returns the name of a flow.

Returns

a string containing the name is returned

Implements [Flow](#).

```
00008 { return name; }
```

References [name](#).

4.4.2.3 getSource()

```
System * FlowIMP::getSource ( ) const [virtual]
```

This method returns the source system pointer.

Returns

a system pointer containing the source memory address is returned

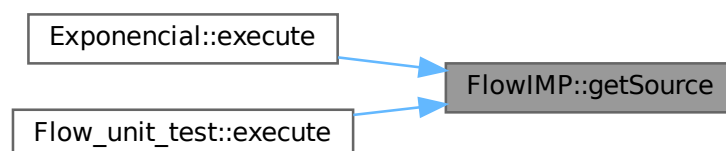
Implements [Flow](#).

```
00011 { return source; }
```

References [source](#).

Referenced by [Exponential::execute\(\)](#), and [Flow_unit_test::execute\(\)](#).

Here is the caller graph for this function:



4.4.2.4 `getTarget()`

```
System * FlowIMP::getTarget ( ) const [virtual]
```

This method returns the target system pointer.

Returns

a system pointer containing the target memory address is returned

Implements [Flow](#).

```
00014 { return target; }
```

References [target](#).

Referenced by [Logistical::execute\(\)](#).

Here is the caller graph for this function:



4.4.2.5 `operator!=()`

```
bool FlowIMP::operator!= (
    const FlowIMP & other ) const [virtual]
```

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

Parameters

<i>other</i>	flow obj to be compare must be passed
--------------	---------------------------------------

Returns

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

```
00033                                     {
00034     return (name != other.name || source != other.source || target != other.target);
00035 }
```

References [name](#), [source](#), and [target](#).

4.4.2.6 `operator=()`

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

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

Parameters

<i>other</i>	flow obj to be cloned must be passed
--------------	--------------------------------------

Returns

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

```
00019                                     {
00020     if(other == *this) return *this;
00021     name = other.name;
00022     source = other.source;
00023     target = other.target;
00024     return *this;
00025 }
```

References [name](#), [source](#), and [target](#).

4.4.2.7 operator==()

```
bool FlowIMP::operator==(
    const FlowIMP & other ) const [virtual]
```

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

Parameters

<i>other</i>	flow obj to be compare must be passed
--------------	---------------------------------------

Returns

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

```
00028                                     {
00029     return (name == other.name && source == other.source && target == other.target);
00030 }
```

References [name](#), [source](#), and [target](#).

4.4.2.8 setName()

```
void FlowIMP::setName (
    const std::string & name ) [virtual]
```

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

Parameters

<i>name</i>	string must be passed to the method
-------------	-------------------------------------

Implements [Flow](#).

```
00009 { this->name = name; }
```

References [name](#).

Referenced by [test_Flow_equal\(\)](#).

Here is the caller graph for this function:



4.4.2.9 setSource()

```
void FlowIMP::setSource (
    System * source ) [virtual]
```

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

Parameters

<i>source</i>	system poiter must be passed to the method
---------------	--

Implements [Flow](#).

```
00012 { this->source = source; }
```

References [source](#).

4.4.2.10 setTarget()

```
void FlowIMP::setTarget (
    System * target ) [virtual]
```

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

Parameters

<i>target</i>	system poiter must be passed to the method
---------------	--

Implements [Flow](#).

```
00015 { this->target = target; }
```

References [target](#).

4.4.3 Member Data Documentation

4.4.3.1 name

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


Name string attribute.

Referenced by [Exponencial::Exponencial\(\)](#), [Exponencial::Exponencial\(\)](#), [Flow_unit_test::Flow_unit_test\(\)](#), [Flow_unit_test::Flow_unit_test\(\)](#), [getName\(\)](#), [Logistical::Logistical\(\)](#), [Logistical::Logistical\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), and [setName\(\)](#).

4.4.3.2 source

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

Source system pointer attribute.

Referenced by [Exponencial::Exponencial\(\)](#), [Exponencial::Exponencial\(\)](#), [Flow_unit_test::Flow_unit_test\(\)](#), [Flow_unit_test::Flow_unit_test\(\)](#), [getSource\(\)](#), [Logistical::Logistical\(\)](#), [Logistical::Logistical\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), and [setSource\(\)](#).

4.4.3.3 target

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

Target system pointer attribute.

Referenced by [Exponencial::Exponencial\(\)](#), [Exponencial::Exponencial\(\)](#), [Flow_unit_test::Flow_unit_test\(\)](#), [Flow_unit_test::Flow_unit_test\(\)](#), [getTarget\(\)](#), [Logistical::Logistical\(\)](#), [Logistical::Logistical\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), and [setTarget\(\)](#).

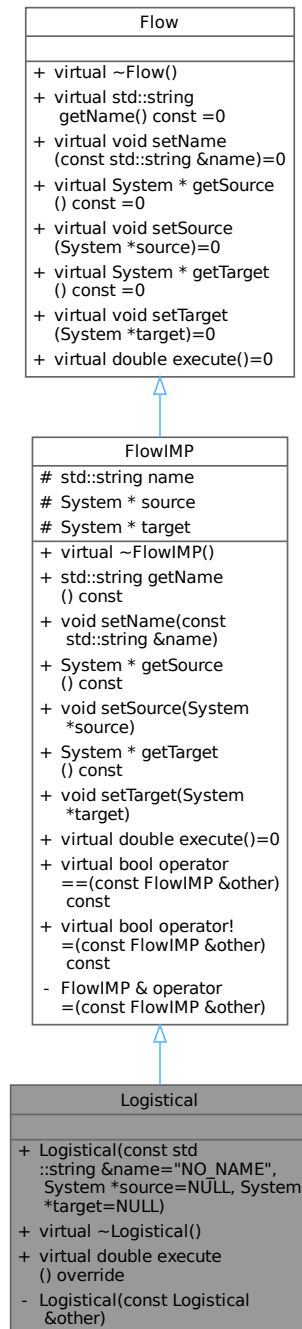
The documentation for this class was generated from the following files:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/FlowIMP.hpp](#)
- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/FlowIMP.cpp](#)

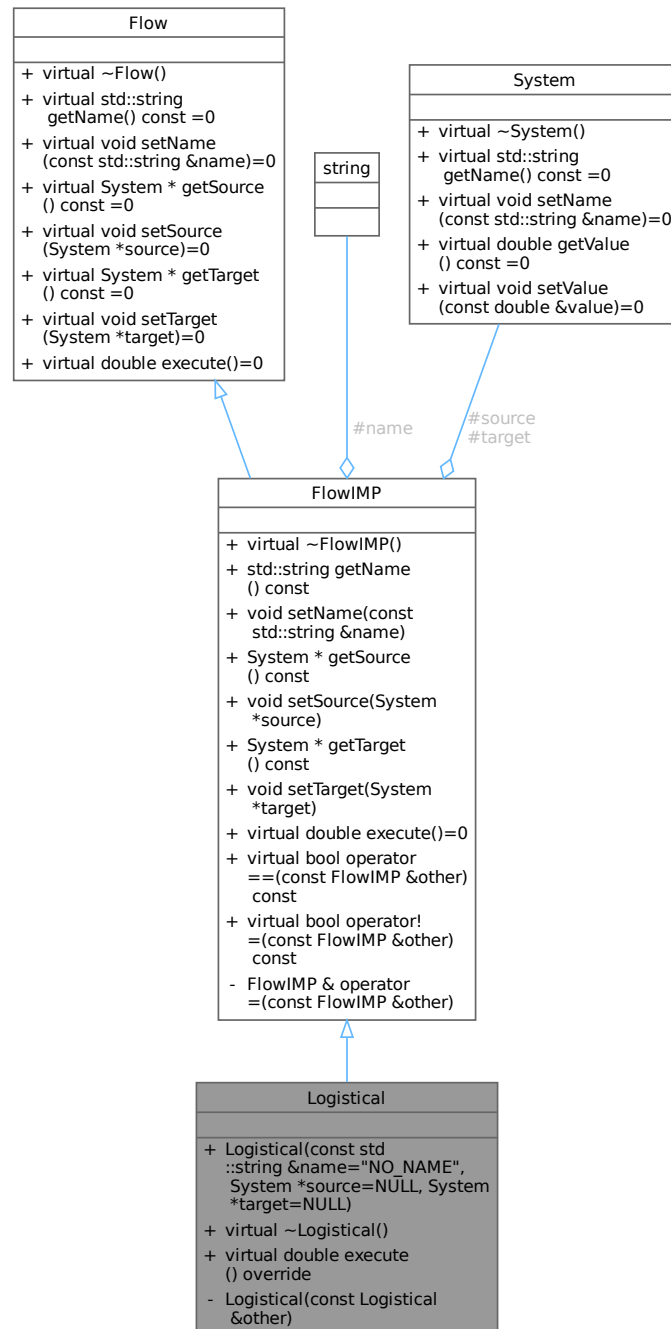
4.5 Logistical Class Reference

```
#include <Logistical.hpp>
```

Inheritance diagram for Logistical:



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.
- 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 FlowIMP

- virtual `~FlowIMP()`
This destructor is a virtual destructor of the class.
- `std::string getName()` const
This method returns the name of a flow.
- void `setName(const std::string &name)`
This method assigns a string to the name of a flow obj.
- `System * getSource()` const
This method returns the source system pointer.
- void `setSource(System *source)`
This method assigns a system pointer to the source of a flow obj.
- `System * getTarget()` const
This method returns the target system pointer.
- void `setTarget(System *target)`
This method assigns a system pointer to the target of a flow obj.
- virtual bool `operator==(const FlowIMP &other)` const
This method is overloading the '==' operator, compare two flows objs.
- virtual bool `operator!=(const FlowIMP &other)` const
This method is overloading the '!=' operator, compare two flows objs.

Public Member Functions inherited from Flow

- virtual `~Flow()`
This destructor is a virtual destructor of the class.

Private Member Functions

- `Logistical(const Logistical &other)`
Construct a new Logistical by a obj.

Additional Inherited Members

Protected Attributes inherited from FlowIMP

- `std::string name`
- `System * source`
- `System * target`

4.5.1 Constructor & Destructor Documentation

4.5.1.1 Logistical() [1/2]

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

Construct a new `Logistical` by a obj.

Parameters

<i>other</i>	Logistical obj
--------------	----------------

```

00011                                     {
00012     this->name = other.name;
00013     this->source = other.source;
00014     this->target = other.target;
00015 }
```

References [FlowIMP::name](#), [FlowIMP::source](#), and [FlowIMP::target](#).

4.5.1.2 Logistical() [2/2]

```

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

Construct a new [Logistical](#) by name, source and target.

Parameters

<i>name</i>	string with default value "NO_NAME"
<i>source</i>	System pointer with default value NULL
<i>target</i>	System pointer with default value NULL

```

00004                                     {
00005     this->name = name;
00006     this->source = source;
00007     this->target = target;
00008 }
```

References [FlowIMP::name](#), [FlowIMP::source](#), and [FlowIMP::target](#).

4.5.1.3 ~Logistical()

```

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

This destructor is a virtual destructor of the Class.

```

00018 {}
```

4.5.2 Member Function Documentation

4.5.2.1 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

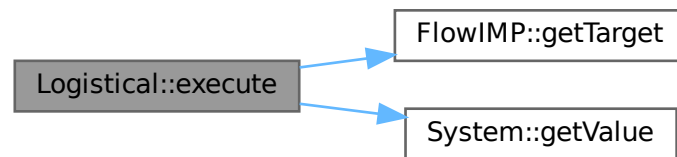
double

Implements [FlowIMP](#).

```
00020     {  
00021     return 0.01 * getTarget()->getValue() * (1.0 - getTarget()->getValue() / 70.0);  
00022 }
```

References [FlowIMP::getTarget\(\)](#), and [System::getValue\(\)](#).

Here is the call graph for this function:



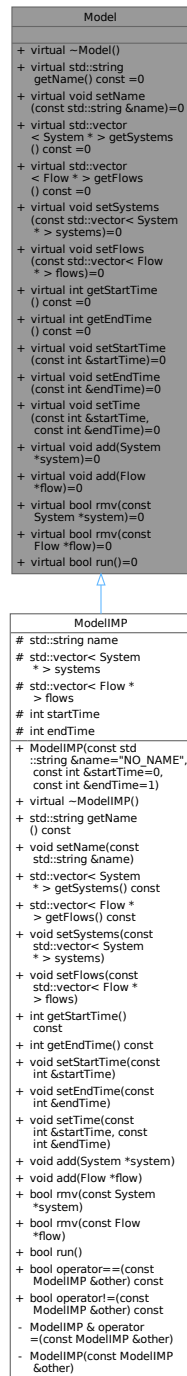
The documentation for this class was generated from the following files:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↔ tests/src/Logistical.hpp](#)
- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_↔ tests/src/Logistical.cpp](#)

4.6 Model Class Reference

```
#include <Model.hpp>
```

Inheritance diagram for Model:



Collaboration diagram for Model:

Model
<ul style="list-style-type: none"> + virtual ~Model() + virtual std::string getName() const =0 + virtual void setName (const std::string &name)=0 + virtual std::vector < System * > getSystems () const =0 + virtual std::vector < Flow * > getFlows () const =0 + virtual void setSystems (const std::vector< System * > systems)=0 + virtual void setFlows (const std::vector< Flow * > flows)=0 + virtual int getStartTime () const =0 + virtual int getEndTime () const =0 + virtual void setStartTime (const int &startTime)=0 + virtual void setEndTime (const int &endTime)=0 + virtual void setTime (const int &startTime, const int &endTime)=0 + virtual void add(System *system)=0 + virtual void add(Flow *flow)=0 + virtual bool rmv(const System *system)=0 + virtual bool rmv(const Flow *flow)=0 + virtual bool run()=0

Public Types

- typedef std::vector< [System](#) * >::iterator [systemIterator](#)
 typedef vectors iterators
- typedef std::vector< [Flow](#) * >::iterator [flowIterator](#)

Public Member Functions

- virtual `~Model ()`
This destructor is a virtual destructor of the class.
- virtual `std::string getName () const =0`
This method returns the name of a [Model](#).
- virtual `void setName (const std::string &name)=0`
This method assigns a string to the name of a [Model](#).
- virtual `std::vector< System * > getSystems () const =0`
This method returns the vector of Systems.
- virtual `std::vector< Flow * > getFlows () const =0`
This method returns the vector of flows.
- virtual `void setSystems (const std::vector< System * > systems)=0`
This method assigns a vector to the systems of a [Model](#).
- virtual `void setFlows (const std::vector< Flow * > flows)=0`
This method assigns a vector to the flows of a [Model](#).
- virtual `int getStartTime () const =0`
This method returns the startTime of a [Model](#).
- virtual `int getEndTime () const =0`
This method returns the end of a [Model](#).
- virtual `void setStartTime (const int &startTime)=0`
This method assigns a int to the startTime of a [Model](#).
- virtual `void setEndTime (const int &endTime)=0`
This method assigns a int to the endTime of a [Model](#).
- virtual `void setTime (const int &startTime, const int &endTime)=0`
This method assigns a int to the startTime and endTime of a [Model](#).
- virtual `void add (System *system)=0`
This method add a [System](#) pointer to the vector of a [Model](#).
- virtual `void add (Flow *flow)=0`
This method add a [Flow](#) pointer to the vector of a [Model](#).
- virtual `bool rmv (const System *system)=0`
This method remove a [System](#) pointer of the vector of a [Model](#).
- virtual `bool rmv (const Flow *flow)=0`
This method remove a [Flow](#) pointer of the vector of a [Model](#).
- virtual `bool run ()=0`
This method run all model.

4.6.1 Member Typedef Documentation

4.6.1.1 flowIterator

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

4.6.1.2 systemIterator

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

```
typedef vectors iterators
```

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ~Model()

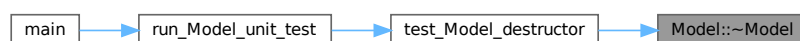
```
virtual Model::~~Model ( ) [inline], [virtual]
```

This destructor is a virtual destructor of the class.

```
00034 {};
```

Referenced by [test_Model_destructor\(\)](#).

Here is the caller graph for this function:



4.6.3 Member Function Documentation

4.6.3.1 add() [1/2]

```
virtual void Model::add (
    Flow * flow ) [pure virtual]
```

This method add a [Flow](#) pointer to the vector of a [Model](#).

Parameters

<i>flow</i>	Flow pointer must be passed to the method
-------------	---

Implemented in [ModelIMP](#).

4.6.3.2 add() [2/2]

```
virtual void Model::add (
    System * system ) [pure virtual]
```

This method add a [System](#) pointer to the vector of a [Model](#).

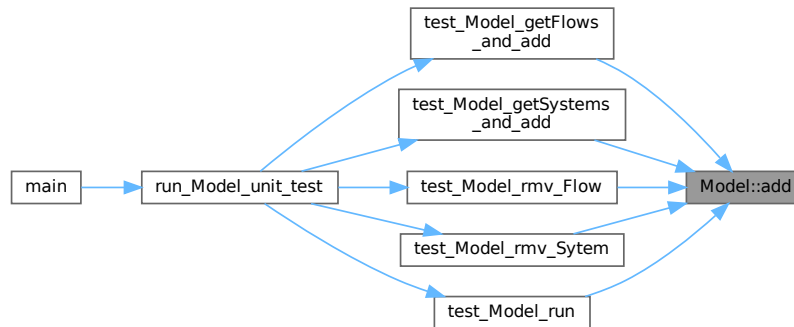
Parameters

<i>system</i>	System pointer must be passed to the method
---------------	---

Implemented in [ModelIMP](#).

Referenced by [test_Model_getFlows_and_add\(\)](#), [test_Model_getSystems_and_add\(\)](#), [test_Model_rmv_Flow\(\)](#), [test_Model_rmv_Sytem\(\)](#), and [test_Model_run\(\)](#).

Here is the caller graph for this function:



4.6.3.3 getEndTime()

```
virtual int Model::getEndTime ( ) const [pure virtual]
```

This method returns the end of a [Model](#).

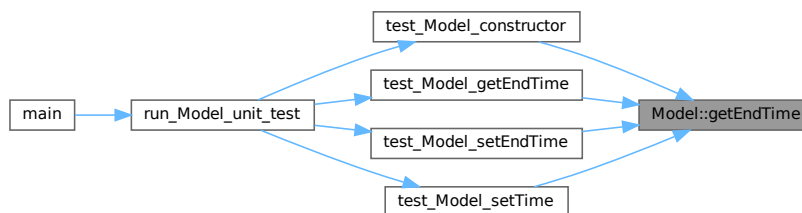
Returns

a int containing the end is returned

Implemented in [ModelIMP](#).

Referenced by [test_Model_constructor\(\)](#), [test_Model_getEndTime\(\)](#), [test_Model_setEndTime\(\)](#), and [test_Model_setTime\(\)](#).

Here is the caller graph for this function:



4.6.3.4 getFlows()

```
virtual std::vector< Flow * > Model::getFlows ( ) const [pure virtual]
```

This method returns the vector of flows.

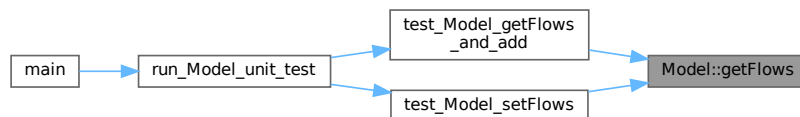
Returns

a vector containing Flows is returned

Implemented in [ModelIMP](#).

Referenced by [test_Model_getFlows_and_add\(\)](#), and [test_Model_setFlows\(\)](#).

Here is the caller graph for this function:



4.6.3.5 getName()

```
virtual std::string Model::getName ( ) const [pure virtual]
```

This method returns the name of a [Model](#).

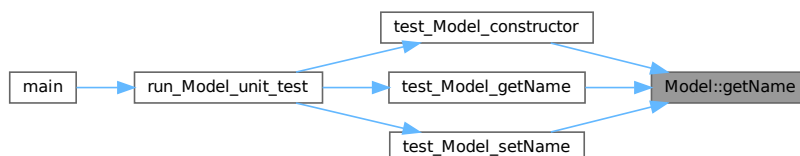
Returns

a string containing the name is returned

Implemented in [ModelIMP](#).

Referenced by [test_Model_constructor\(\)](#), [test_Model_getName\(\)](#), and [test_Model_setName\(\)](#).

Here is the caller graph for this function:



4.6.3.6 getStartTime()

```
virtual int Model::getStartTime ( ) const [pure virtual]
```

This method returns the startTime of a [Model](#).

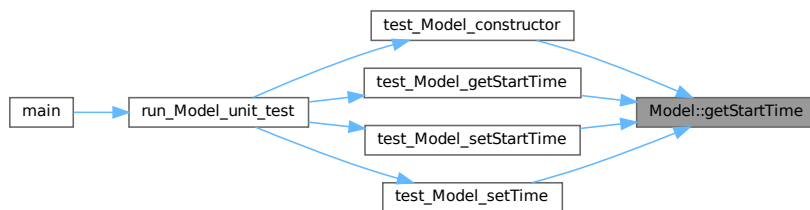
Returns

a int containing the startTime is returned

Implemented in [ModelIMP](#).

Referenced by [test_Model_constructor\(\)](#), [test_Model_getStartTime\(\)](#), [test_Model_setStartTime\(\)](#), and [test_Model_setTime\(\)](#).

Here is the caller graph for this function:



4.6.3.7 getSystems()

```
virtual std::vector< System * > Model::getSystems ( ) const [pure virtual]
```

This method returns the vector of Systems.

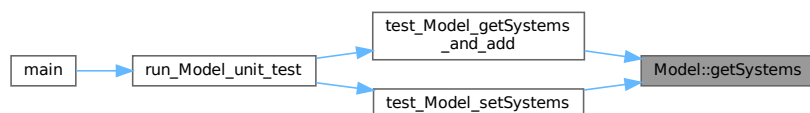
Returns

a vector containing Systems is returned

Implemented in [ModelIMP](#).

Referenced by [test_Model_getSystems_and_add\(\)](#), and [test_Model_setSystems\(\)](#).

Here is the caller graph for this function:



4.6.3.8 rmv() [1/2]

```
virtual bool Model::rmv (
    const Flow * flow ) [pure virtual]
```

This method remove a [Flow](#) pointer of the vector of a [Model](#).

Parameters

<i>flow</i>	Flow pointer iterator must be passed to the method
-------------	--

Returns

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

Implemented in [ModelIMP](#).

4.6.3.9 `rmv()` [2/2]

```
virtual bool Model::rmv (
    const System * system ) [pure virtual]
```

This method remove a [System](#) pointer of the vector of a [Model](#).

Parameters

<i>system</i>	System pointer iterator must be passed to the method
---------------	--

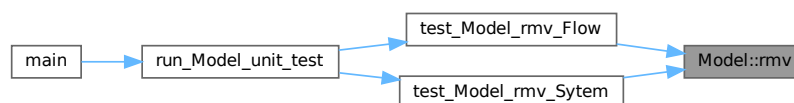
Returns

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

Implemented in [ModelIMP](#).

Referenced by [test_Model_rmv_Flow\(\)](#), and [test_Model_rmv_Sytem\(\)](#).

Here is the caller graph for this function:

4.6.3.10 `run()`

```
virtual bool Model::run ( ) [pure virtual]
```

This method run all model.

Returns

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

Implemented in [ModelIMP](#).

Referenced by [test_Model_run\(\)](#).

Here is the caller graph for this function:

**4.6.3.11 setEndTime()**

```
virtual void Model::setEndTime (
    const int & endTime ) [pure virtual]
```

This method assigns a int to the endTime of a [Model](#).

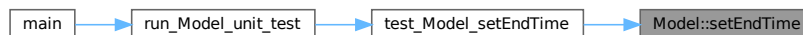
Parameters

<i>endTime</i>	int must be passed to the method
----------------	----------------------------------

Implemented in [ModelIMP](#).

Referenced by [test_Model_setEndTime\(\)](#).

Here is the caller graph for this function:

**4.6.3.12 setFlows()**

```
virtual void Model::setFlows (
    const std::vector< Flow * > flows ) [pure virtual]
```

This method assigns a vector to the flows of a [Model](#).

Parameters

<i>flows</i>	int must be passed to the method
--------------	----------------------------------

Implemented in [ModelIMP](#).

Referenced by [test_Model_setFlows\(\)](#).

Here is the caller graph for this function:

**4.6.3.13 setName()**

```
virtual void Model::setName (
    const std::string & name ) [pure virtual]
```

This method assigns a string to the name of a [Model](#).

Parameters

<i>name</i>	string must be passed to the method
-------------	-------------------------------------

Implemented in [ModelIMP](#).

Referenced by [test_Model_setName\(\)](#).

Here is the caller graph for this function:

**4.6.3.14 setStartTime()**

```
virtual void Model::setStartTime (
    const int & startTime ) [pure virtual]
```

This method assigns a int to the startTime of a [Model](#).

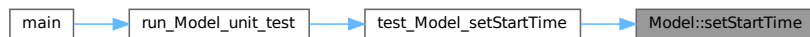
Parameters

<i>startTime</i>	int must be passed to the method
------------------	----------------------------------

Implemented in [ModellIMP](#).

Referenced by [test_Model_setStartTime\(\)](#).

Here is the caller graph for this function:

**4.6.3.15 setSystems()**

```

virtual void Model::setSystems (
    const std::vector< System * > systems ) [pure virtual]
  
```

This method assigns a vector to the systems of a [Model](#).

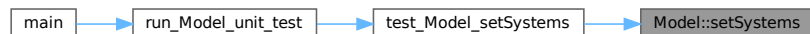
Parameters

<i>systems</i>	int must be passed to the method
----------------	----------------------------------

Implemented in [ModellIMP](#).

Referenced by [test_Model_setSystems\(\)](#).

Here is the caller graph for this function:

**4.6.3.16 setTime()**

```

virtual void Model::setTime (
    const int & startTime,
    const int & endTime ) [pure virtual]
  
```

This method assigns a int to the startTime and endTime of a [Model](#).

Parameters

<i>startTime</i>	int must be passed to the method
<i>endTime</i>	int must be passed to the method

Implemented in [ModelIMP](#).

Referenced by [test_Model_setTime\(\)](#).

Here is the caller graph for this function:



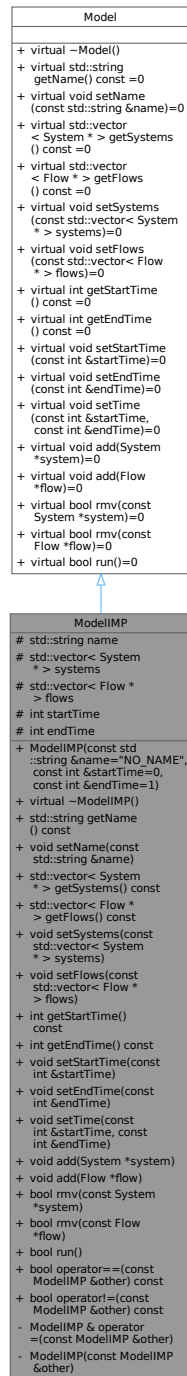
The documentation for this class was generated from the following file:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.hpp](#)

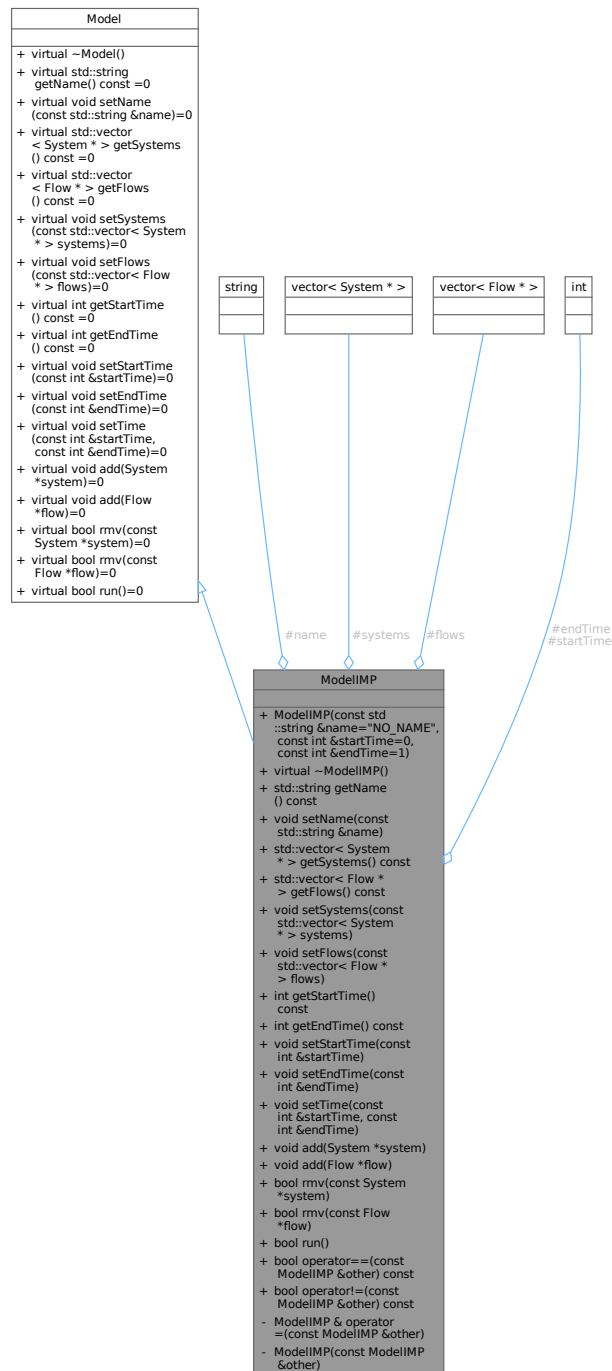
4.7 ModelIMP Class Reference

```
#include <ModelIMP.hpp>
```

Inheritance diagram for ModelIMP:



Collaboration diagram for ModelIMP:



Public Member Functions

- **ModelIMP** (const std::string &name="NO_NAME", const int &startTime=0, const int &endTime=1)
Construct a new **Model** by name and start and end time.
- virtual **~ModelIMP** ()
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](#).
- std::vector< [System](#) * > [getSystems](#) () const

This method returns the vector of Systems.
- std::vector< [Flow](#) * > [getFlows](#) () const

This method returns the vector of flows.
- void [setSystems](#) (const std::vector< [System](#) * > systems)

This method assigns a vector to the systems of a [Model](#).
- void [setFlows](#) (const std::vector< [Flow](#) * > flows)

This method assigns a vector to the flows of a [Model](#).
- int [getStartTime](#) () const

This method returns the startTime of a [Model](#).
- int [getEndTime](#) () 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 [System](#) *system)

This method remove a [System](#) pointer of the vector of a [Model](#).
- bool [rmv](#) (const [Flow](#) *flow)

This method remove a [Flow](#) pointer of the vector of a [Model](#).
- bool [run](#) ()

This method run all model.
- bool [operator==](#) (const [ModelIMP](#) &other) const

This method is overloading the '==' operator, compare two models objs.
- bool [operator!=](#) (const [ModelIMP](#) &other) const

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

Public Member Functions inherited from [Model](#)

- virtual [~Model](#) ()
- This destructor is a virtual destructor of the class.*

Protected Attributes

- std::string [name](#)
- std::vector< [System](#) * > [systems](#)
- std::vector< [Flow](#) * > [flows](#)
- int [startTime](#)
- int [endTime](#)

Private Member Functions

- [ModelIMP](#) & [operator=](#) (const [ModelIMP](#) &other)
This method is overloading the '=' operator, "cloning" from one [Model](#) to another.
- [ModelIMP](#) (const [ModelIMP](#) &other)
Construct a new [Model](#) by a obj.

Additional Inherited Members

Public Types inherited from [Model](#)

- typedef std::vector< [System](#) * >::iterator [systemIterator](#)
typedef vetors iterators
- typedef std::vector< [Flow](#) * >::iterator [flowIterator](#)

4.7.1 Constructor & Destructor Documentation

4.7.1.1 [ModelIMP](#)() [1/2]

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

Construct a new [Model](#) by a obj.

Parameters

<i>other</i>	Model obj
--------------	---------------------------

```
00006                                     : name(other.name), startTime(other.startTime),
    endTime(other.endTime) {
00007     flows.clear();
00008     systems.clear();
00009     for (auto i : other.flows) flows.push_back(i);
00010     for (auto i : other.systems) systems.push_back(i);
00011 }
```

References [flows](#), and [systems](#).

4.7.1.2 [ModelIMP](#)() [2/2]

```
ModelIMP::ModelIMP (
    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

<i>name</i>	string with default value "NO_NAME"
<i>startTime</i>	int with default value 0
<i>endTime</i>	int with default value 1

```
00004 : name(name), startTime(startTime), endTime(endTime) {}
```

4.7.1.3 ~ModelIMP()

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

This destructor is a virtual destructor of the class.

```
00014 {systems.clear(); flows.clear();}
```

References [flows](#), and [systems](#).

4.7.2 Member Function Documentation

4.7.2.1 add() [1/2]

```
void ModelIMP::add (
    Flow * flow ) [virtual]
```

This method add a [Flow](#) pointer to the vector of a [Model](#).

Parameters

<i>flow</i>	Flow pointer must be passed to the method
-------------	---

Implements [Model](#).

```
00035 { flows.push_back(flow); }
```

References [flows](#).

4.7.2.2 add() [2/2]

```
void ModelIMP::add (
    System * system ) [virtual]
```

This method add a [System](#) pointer to the vector of a [Model](#).

Parameters

<i>system</i>	System pointer must be passed to the method
---------------	---

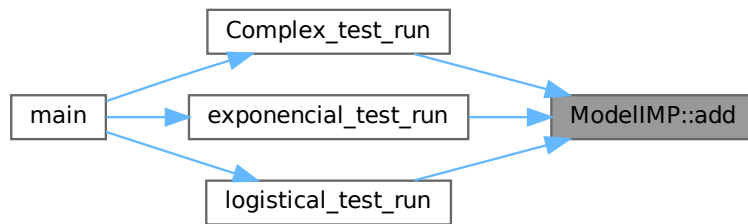
Implements [Model](#).

```
00034 { 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:



4.7.2.3 getEndTime()

```
int ModelIMP::getEndTime ( ) const [virtual]
```

This method returns the end of a [Model](#).

Returns

a int containing the end is returned

Implements [Model](#).

```
00027 { return endTime; }
```

References [endTime](#).

4.7.2.4 getFlows()

```
std::vector< Flow * > ModelIMP::getFlows ( ) const [virtual]
```

This method returns the vector of flows.

Returns

a vector containing Flows is returned

Implements [Model](#).

```
00022 { return flows; };
```

References [flows](#).

4.7.2.5 getName()

```
std::string ModelIMP::getName ( ) const [virtual]
```

This method returns the name of a [Model](#).

Returns

a string containing the name is returned

Implements [Model](#).

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

References [name](#).

4.7.2.6 getStartTime()

```
int ModelIMP::getStartTime ( ) const [virtual]
```

This method returns the startTime of a [Model](#).

Returns

a int containing the startTime is returned

Implements [Model](#).

```
00026 { return startTime; }
```

References [startTime](#).

4.7.2.7 getSystems()

```
std::vector< System * > ModelIMP::getSystems ( ) const [virtual]
```

This method returns the vector of Systems.

Returns

a vector containing Systems is returned

Implements [Model](#).

```
00021 { return systems; }
```

References [systems](#).

4.7.2.8 operator"!="()

```
bool ModelIMP::operator!= (
    const ModelIMP & other ) const
```

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

Parameters

<i>other</i>	model obj to be compare must be passed
--------------	--

Returns

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

```

00109                                     {
00110     return (name != other.name || systems != other.systems || flows != other.flows || startTime !=
other.startTime || endTime != other.endTime);
00111 }
```

References [endTime](#), [flows](#), [name](#), [startTime](#), and [systems](#).

4.7.2.9 operator=()

```

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

This method is overloading the '=' operator, "cloning" from one [Model](#) to another.

Parameters

<i>other</i>	Model obj to be cloned must be passed
--------------	---

Returns

A [Model](#) is returned that is a clone of what was passed to the method

```

00092                                     {
00093     if(other == *this) return *this;
00094     name = other.name;
00095     systems = other.systems;
00096     flows.clear();
00097     systems.clear();
00098     for (auto i : other.flows) flows.push_back(i);
00099     for (auto i : other.systems) systems.push_back(i);
00100     startTime = other.startTime;
00101     endTime = other.endTime;
00102     return *this;
00103 }
```

References [endTime](#), [flows](#), [name](#), [startTime](#), and [systems](#).

4.7.2.10 operator==()

```

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

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

Parameters

<i>other</i>	model obj to be compare must be passed
--------------	--

Returns

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

```
00105         {
00106     return (name == other.name && systems == other.systems && flows == other.flows && startTime ==
other.startTime && endTime == other.endTime);
00107 }
```

References [endTime](#), [flows](#), [name](#), [startTime](#), and [systems](#).

4.7.2.11 rmv() [1/2]

```
bool ModelIMP::rmv (
    const Flow * flow ) [virtual]
```

This method remove a [Flow](#) pointer of the vector of a [Model](#).

Parameters

<i>flow</i>	Flow pointer iterator must be passed to the method
-------------	--

Returns

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

Implements [Model](#).

```
00045     {
00046     for(flowIterator i = flows.begin(); i < flows.end(); i++)
00047     if(*i == flow){
00048         flows.erase(i);
00049         return true;
00050     }
00051     return false;
00052 }
```

References [flows](#).

4.7.2.12 rmv() [2/2]

```
bool ModelIMP::rmv (
    const System * system ) [virtual]
```

This method remove a [System](#) pointer of the vector of a [Model](#).

Parameters

<i>system</i>	System pointer iterator must be passed to the method
---------------	--

Returns

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

Implements [Model](#).

```
00037     {
```

```

00038     for(systemIterator i = systems.begin(); i < systems.end(); i++)
00039         if(*i == system){
00040             systems.erase(i);
00041             return true;
00042         }
00043     return false;
00044 }

```

References [systems](#).

4.7.2.13 run()

```
bool ModelIMP::run ( ) [virtual]
```

This method run all model.

Returns

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

Implements [Model](#).

```

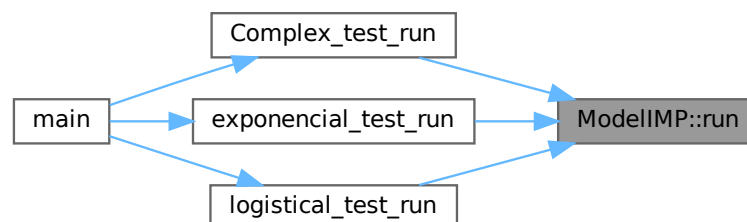
00056     {
00057         std::vector<double> flowValue;
00058         flowIterator f;
00059         std::vector<double>::iterator d;
00060         double calcValue;
00061
00062         for(int i = startTime; i < endTime; i++){
00063
00064             f = flows.begin();
00065
00066             while (f != flows.end()) {
00067                 flowValue.push_back ((*f)->execute());
00068                 f++;
00069             }
00070
00071             f = flows.begin();
00072             d = flowValue.begin();
00073
00074             while (f != flows.end()) {
00075                 calcValue = (*f)->getSource()->getValue() - (*d);
00076                 (*f)->getSource()->setValue(calcValue);
00077                 calcValue = (*f)->getTarget()->getValue() + (*d);
00078                 (*f)->getTarget()->setValue(calcValue);
00079                 f++;
00080                 d++;
00081             }
00082
00083             flowValue.clear();
00084
00085         }
00086
00087         return true;
00088     }

```

References [endTime](#), [flows](#), and [startTime](#).

Referenced by [Complex_test_run\(\)](#), [exponential_test_run\(\)](#), and [logistical_test_run\(\)](#).

Here is the caller graph for this function:



4.7.2.14 setEndTime()

```
void ModelIMP::setEndTime (
    const int & endTime ) [virtual]
```

This method assigns a int to the endTime of a [Model](#).

Parameters

<i>endTime</i>	int must be passed to the method
----------------	----------------------------------

Implements [Model](#).

```
00029 { this->endTime = endTime; }
```

References [endTime](#).

4.7.2.15 setFlows()

```
void ModelIMP::setFlows (
    const std::vector< Flow * > flows ) [virtual]
```

This method assigns a vector to the flows of a [Model](#).

Parameters

<i>flows</i>	int must be passed to the method
--------------	----------------------------------

Implements [Model](#).

```
00024 { for(auto i : flows) this->flows.push_back(i); }
```

References [flows](#).

4.7.2.16 setName()

```
void ModelIMP::setName (
    const std::string & name ) [virtual]
```

This method assigns a string to the name of a [Model](#).

Parameters

<i>name</i>	string must be passed to the method
-------------	-------------------------------------

Implements [Model](#).

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

References [name](#).

Referenced by [test_Model_equal\(\)](#).

Here is the caller graph for this function:



4.7.2.17 setStartTime()

```
void ModelIMP::setStartTime (
    const int & startTime ) [virtual]
```

This method assigns a int to the startTime of a [Model](#).

Parameters

<i>startTime</i>	int must be passed to the method
------------------	----------------------------------

Implements [Model](#).

```
00028 { this->startTime = startTime; }
```

References [startTime](#).

4.7.2.18 setSystems()

```
void ModelIMP::setSystems (
    const std::vector< System * > systems ) [virtual]
```

This method assigns a vector to the systems of a [Model](#).

Parameters

<i>systems</i>	int must be passed to the method
----------------	----------------------------------

Implements [Model](#).

```
00023 { for(auto i : systems) this->systems.push_back(i); }
```

References [systems](#).

4.7.2.19 setTime()

```
void ModelIMP::setTime (
    const int & startTime,
    const int & endTime ) [virtual]
```

This method assigns a int to the startTime and endTime of a [Model](#).

Parameters

<i>startTime</i>	int must be passed to the method
<i>endTime</i>	int must be passed to the method

Implements [Model](#).

```
00030 { this->startTime = startTime; this->endTime = endTime; }
```

References [endTime](#), and [startTime](#).

4.7.3 Member Data Documentation

4.7.3.1 endTime

```
int ModelIMP::endTime [protected]
```

End time simulation integer attribute.

Referenced by [getEndTime\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), [run\(\)](#), [setEndTime\(\)](#), and [setTime\(\)](#).

4.7.3.2 flows

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

[Flow](#) pointers vector.

Referenced by [add\(\)](#), [getFlows\(\)](#), [ModelIMP\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), [rmv\(\)](#), [run\(\)](#), [setFlows\(\)](#), and [~ModelIMP\(\)](#).

4.7.3.3 name

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

Name string attribute.

Referenced by [getName\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), and [setName\(\)](#).

4.7.3.4 startTime

```
int ModelIMP::startTime [protected]
```

Start time simulation integer attribute.

Referenced by [getStartTime\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), [run\(\)](#), [setStartTime\(\)](#), and [setTime\(\)](#).

4.7.3.5 systems

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

[System](#) pointers vector.

Referenced by [add\(\)](#), [getSystems\(\)](#), [ModelIMP\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), [rmv\(\)](#), [setSystems\(\)](#), and [~ModelIMP\(\)](#).

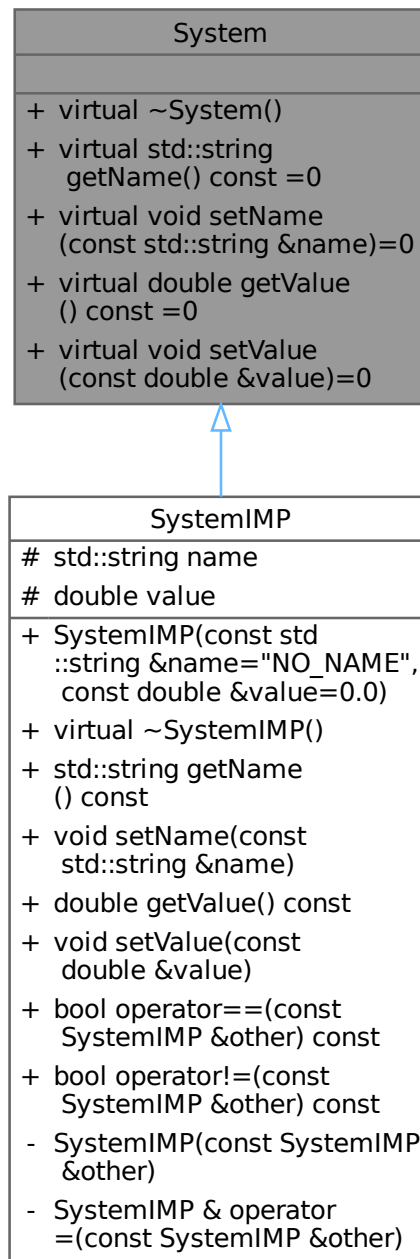
The documentation for this class was generated from the following files:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/ModelIMP.hpp](#)
- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/ModelIMP.cpp](#)

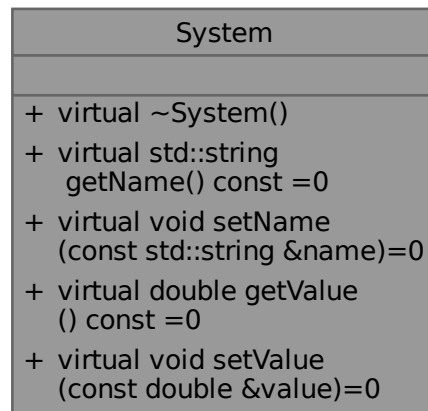
4.8 System Class Reference

```
#include <System.hpp>
```


Inheritance diagram for System:



Collaboration diagram for System:



Public Member Functions

- virtual [~System](#) ()
This destructor is a virtual destructor of the Class.
- virtual std::string [getName](#) () const =0
This method returns the name of a system.
- virtual void [setName](#) (const std::string &name)=0
This method assigns a string to the name of a system.
- virtual double [getValue](#) () const =0
This method returns the value of a system.
- virtual void [setValue](#) (const double &value)=0
This method assigns a double to the value of a system.

4.8.1 Constructor & Destructor Documentation

4.8.1.1 ~System()

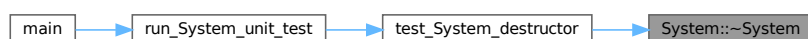
```
virtual System::~~System ( ) [inline], [virtual]
```

This destructor is a virtual destructor of the Class.

```
00023 {};
```

Referenced by [test_System_destructor\(\)](#).

Here is the caller graph for this function:



4.8.2 Member Function Documentation

4.8.2.1 getName()

```
virtual std::string System::getName ( ) const [pure virtual]
```

This method returns the name of a system.

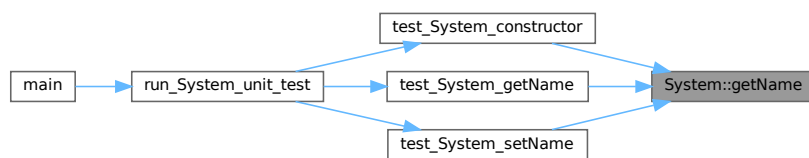
Returns

a string containing the name is returned

Implemented in [SystemIMP](#).

Referenced by [test_System_constructor\(\)](#), [test_System_getName\(\)](#), and [test_System_setName\(\)](#).

Here is the caller graph for this function:



4.8.2.2 getValue()

```
virtual double System::getValue ( ) const [pure virtual]
```

This method returns the value of a system.

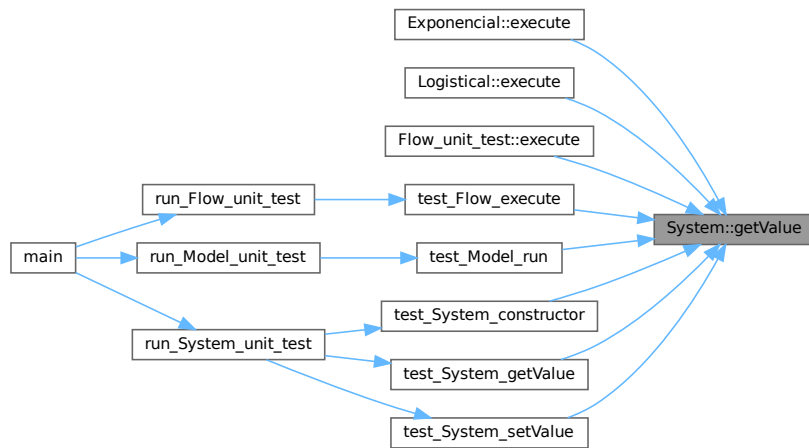
Returns

a double containing the value is returned

Implemented in [SystemIMP](#).

Referenced by [Exponencial::execute\(\)](#), [Logistical::execute\(\)](#), [Flow_unit_test::execute\(\)](#), [test_Flow_execute\(\)](#), [test_Model_run\(\)](#), [test_System_constructor\(\)](#), [test_System_getValue\(\)](#), and [test_System_setValue\(\)](#).

Here is the caller graph for this function:



4.8.2.3 setName()

```
virtual void System::setName (
    const std::string & name ) [pure virtual]
```

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

Parameters

<i>name</i>	string must be passed to the method
-------------	-------------------------------------

Implemented in [SystemIMP](#).

Referenced by [test_System_setName\(\)](#).

Here is the caller graph for this function:



4.8.2.4 setValue()

```
virtual void System::setValue (
    const double & value ) [pure virtual]
```

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

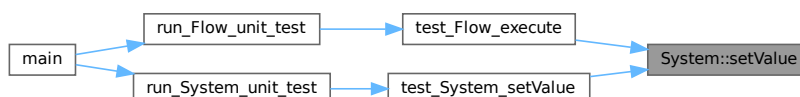
Parameters

<i>value</i>	double must be passed to the method
--------------	-------------------------------------

Implemented in [SystemIMP](#).

Referenced by [test_Flow_execute\(\)](#), and [test_System_setValue\(\)](#).

Here is the caller graph for this function:



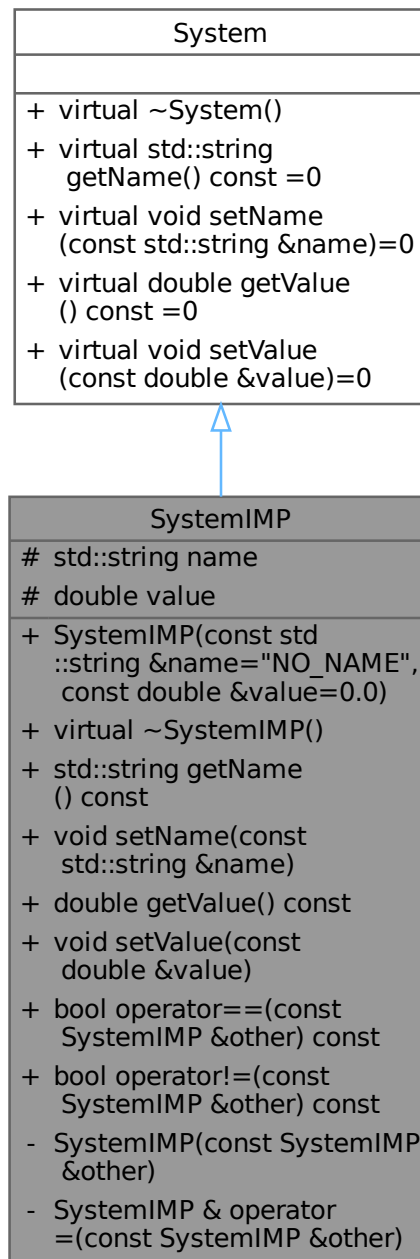
The documentation for this class was generated from the following file:

- [/home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/System.hpp](#)

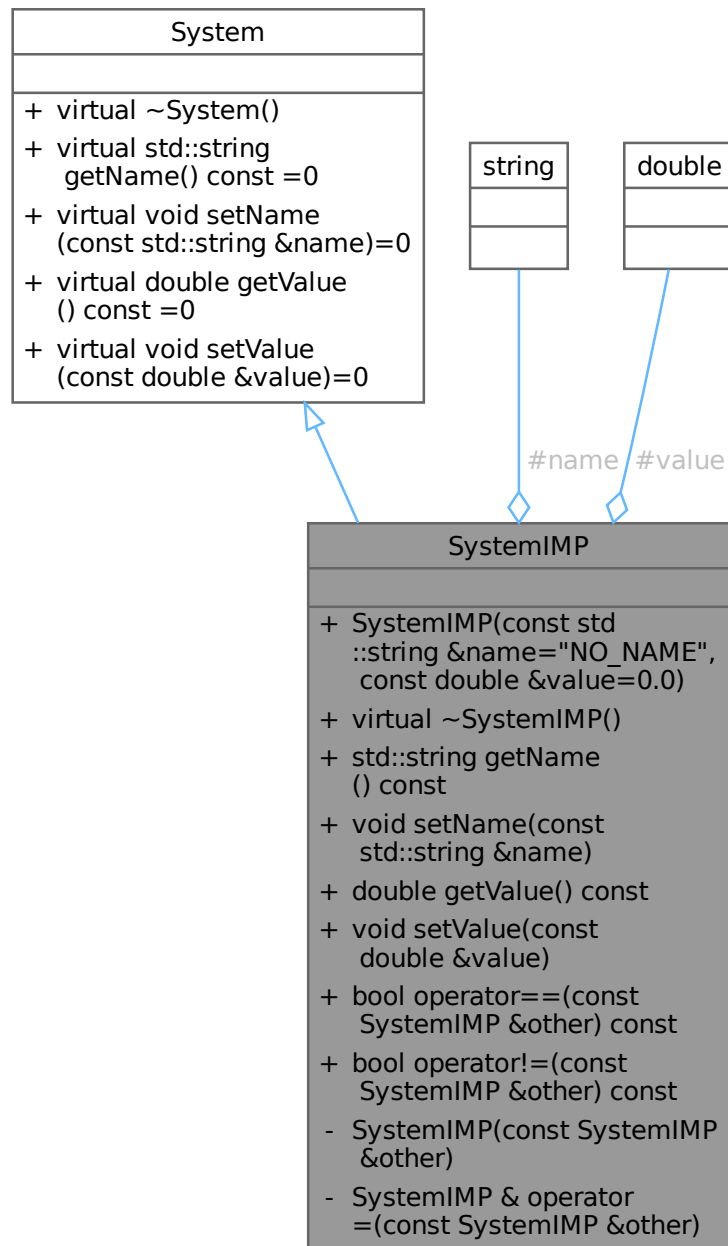
4.9 SystemIMP Class Reference

```
#include <SystemIMP.hpp>
```

Inheritance diagram for SystemIMP:



Collaboration diagram for SystemIMP:



Public Member Functions

- [SystemIMP](#) (const std::string &name="NO_NAME", const double &value=0.0)
Construct a new [System](#) by name and value.
- virtual [~SystemIMP](#) ()
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.

- bool `operator==` (const `SystemIMP` &`other`) const

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

- bool `operator!=` (const `SystemIMP` &`other`) const

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

Public Member Functions inherited from `System`

- virtual `~System` ()

This destructor is a virtual destructor of the Class.

Protected Attributes

- std::string `name`
- double `value`

Private Member Functions

- `SystemIMP` (const `SystemIMP` &`other`)

Construct a new `System` by a obj.

- `SystemIMP` & `operator=` (const `SystemIMP` &`other`)

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

4.9.1 Constructor & Destructor Documentation

4.9.1.1 `SystemIMP()` [1/2]

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

Construct a new `System` by a obj.

Parameters

<i>other</i>	<code>System</code> obj
--------------	-------------------------

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

4.9.1.2 `SystemIMP()` [2/2]

```
SystemIMP::SystemIMP (
```



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

Construct a new [System](#) by name and value.

Parameters

<i>name</i>	string with default value "NO_NAME"
<i>value</i>	double with default value 0.0

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

4.9.1.3 ~SystemIMP()

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

This destructor is a virtual destructor of the Class.

```
00009 {};
```

4.9.2 Member Function Documentation

4.9.2.1 getName()

```
std::string SystemIMP::getName ( ) const [virtual]
```

This method returns the name of a system.

Returns

a string containing the name is returned

Implements [System](#).

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

References [name](#).

4.9.2.2 getValue()

```
double SystemIMP::getValue ( ) const [virtual]
```

This method returns the value of a system.

Returns

a double containing the value is returned

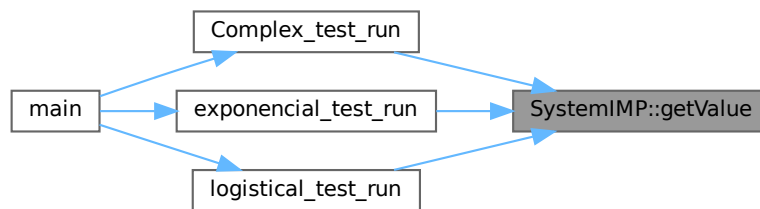
Implements [System](#).

```
00016 { return value; }
```

References [value](#).

Referenced by [Complex_test_run\(\)](#), [exponencial_test_run\(\)](#), and [logistical_test_run\(\)](#).

Here is the caller graph for this function:

**4.9.2.3 operator!=()**

```
bool SystemIMP::operator!= (
    const SystemIMP & other ) const
```

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

Parameters

<i>other</i>	system obj to be compare must be passed
--------------	---

Returns

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

```
00034 {
00035     return (name != other.name || value != other.value);
00036     // Compare todos os membros para verificar igualdade
00037 }
```

References [name](#), and [value](#).

4.9.2.4 operator=()

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

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

Parameters

<i>other</i>	system obj to be cloned must be passed
--------------	--

Returns

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

```

00021                                     {
00022     if(other == *this) return *this;
00023     name = other.name;
00024     value = other.value;
00025     return *this;
00026 }
```

References [name](#), and [value](#).

4.9.2.5 operator==()

```

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

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

Parameters

<i>other</i>	system obj to be compare must be passed
--------------	---

Returns

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

```

00028                                     {
00029     return (name == other.name && value == other.value);
00030     // Compare todos os membros para verificar igualdade
00031 }
```

References [name](#), and [value](#).

4.9.2.6 setName()

```

void SystemIMP::setName (
    const std::string & name ) [virtual]
```

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

Parameters

<i>name</i>	string must be passed to the method
-------------	-------------------------------------

Implements [System](#).

```

00014 { this->name = name; }
```

References [name](#).

4.9.2.7 setValue()

```
void SystemIMP::setValue (
    const double & value ) [virtual]
```

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

Parameters

<i>value</i>	double must be passed to the method
--------------	-------------------------------------

Implements [System](#).

```
00017 { this->value = value; }
```

References [value](#).

4.9.3 Member Data Documentation

4.9.3.1 name

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

Name string attribute.

Referenced by [getName\(\)](#), [operator!=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), and [setName\(\)](#).

4.9.3.2 value

```
double SystemIMP::value [protected]
```

Value double attribute.

Referenced by [getValue\(\)](#), [operator!=\(\)](#), [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/SystemIMP.hpp
- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/SystemIMP.cpp

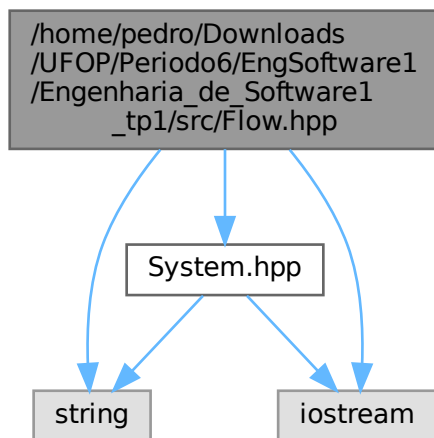
Chapter 5

File Documentation

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

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

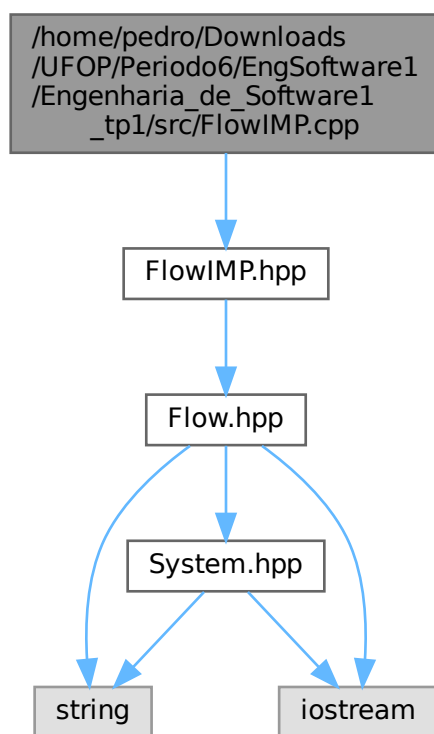
Include dependency graph for Flow.hpp:



5.3 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/FlowIMP.cpp File Reference

```
#include "FlowIMP.hpp"
```

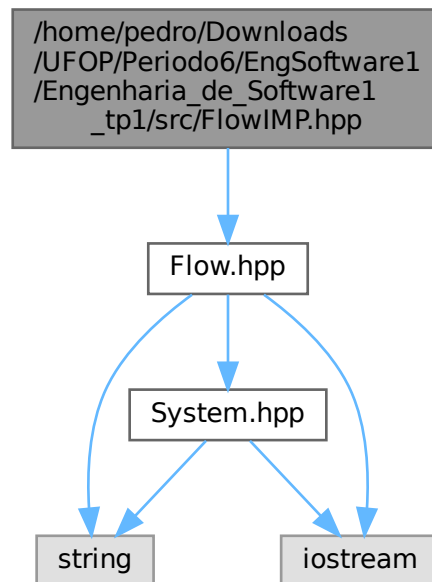
Include dependency graph for FlowIMP.cpp:



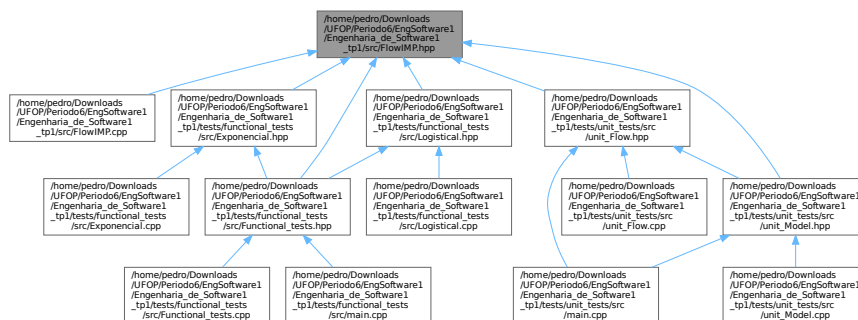
5.4 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/FlowIMP.hpp File Reference

```
#include "Flow.hpp"
```

Include dependency graph for FlowIMP.hpp:



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



Classes

- class [FlowIMP](#)

5.5 FlowIMP.hpp

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * @file FlowIMP.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the flow implementation
00005  *****/
00006
00007 #ifndef FLOWIMP_HPP
00008 #define FLOWIMP_HPP
00009
00010 #include "Flow.hpp"
00011
00012 /*****
00013  * @brief The Flow implementation defines the attributes and implements the methods
00014  *****/
00015
00016 class FlowIMP : public Flow{
00017     private:
00023         FlowIMP& operator=(const FlowIMP& other); // Operador de atribuição
00024
00025     protected:
00026         std::string name;
00027         System* source;
00028         System* target;
00030     public:
00031         //Destructor
00035         virtual ~FlowIMP();
00036
00037         //Getters e setters
00038         //Name
00043         std::string getName() const;
00048         void setName(const std::string& name);
00049         //Source
00054         System* getSource() const;
00059         void setSource(System* source);
00060         //Target
00065         System* getTarget() const;
00070         void setTarget(System* target);
00071
00072         //Metodos
00077         virtual double execute() = 0;
00078
00079         //Sobrecarga de operadores
00085         virtual bool operator==(const FlowIMP& other) const; // Operador de igualdade
00091         virtual bool operator!=(const FlowIMP& other) const; // Operador de diferença
00092 };
00093
00094
00095 #endif

```

5.6 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.hpp File Reference ↩

```

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

```



```

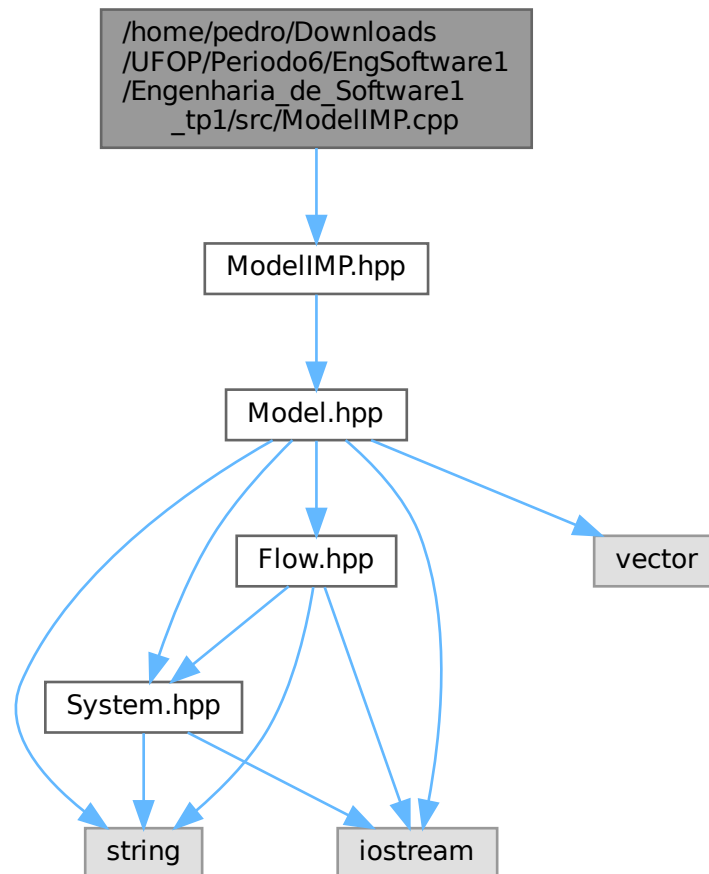
00001 /*****
00002  * @file Model.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the simulation model
00005  *****/
00006
00007 #ifndef MODEL_HPP
00008 #define MODEL_HPP
00009
00010 #include "System.hpp"
00011 #include "Flow.hpp"
00012
00013
00014 /*****
00015  * @brief This class represents the general simulation model, it contains figures for simulation and
00016  its execution.
00017  *****/
00018
00019 #include <string>
00020 #include <iostream>
00021 #include <vector>
00022
00023 class Model{
00024 public:
00025     //Iteradores
00026     typedef std::vector<System*>::iterator systemIterator;
00027     typedef std::vector<Flow*>::iterator flowIterator;
00028
00029     //Destrutor
00030     virtual ~Model() {};
00031
00032     //Getters e setters
00033     //Name
00034     virtual std::string getName() const = 0;
00035     virtual void setName(const std::string& name) = 0;
00036     //Vector
00037     virtual std::vector<System*> getSystems() const = 0;
00038     virtual std::vector<Flow*> getFlows() const = 0;
00039     virtual void setSystems(const std::vector<System*> systems) = 0;
00040     virtual void setFlows(const std::vector<Flow*> flows) = 0;
00041     //Time
00042     virtual int getStartTime() const = 0;
00043     virtual int getEndTime() const = 0;
00044     virtual void setStartTime(const int& startTime) = 0;
00045     virtual void setEndTime(const int& endTime) = 0;
00046     virtual void setTime(const int& startTime, const int& endTime) = 0;
00047
00048     //Metodos
00049     //add
00050     virtual void add(System* system) = 0;
00051     virtual void add(Flow* flow) = 0;
00052     //remove
00053     virtual bool rmv(const System* system) = 0;
00054     virtual bool rmv(const Flow* flow) = 0;
00055     //Others
00056     virtual bool run() = 0;
00057 };
00058
00059 #endif

```

5.8 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/ModelIMP.cpp File Reference

```
#include "ModelIMP.hpp"
```

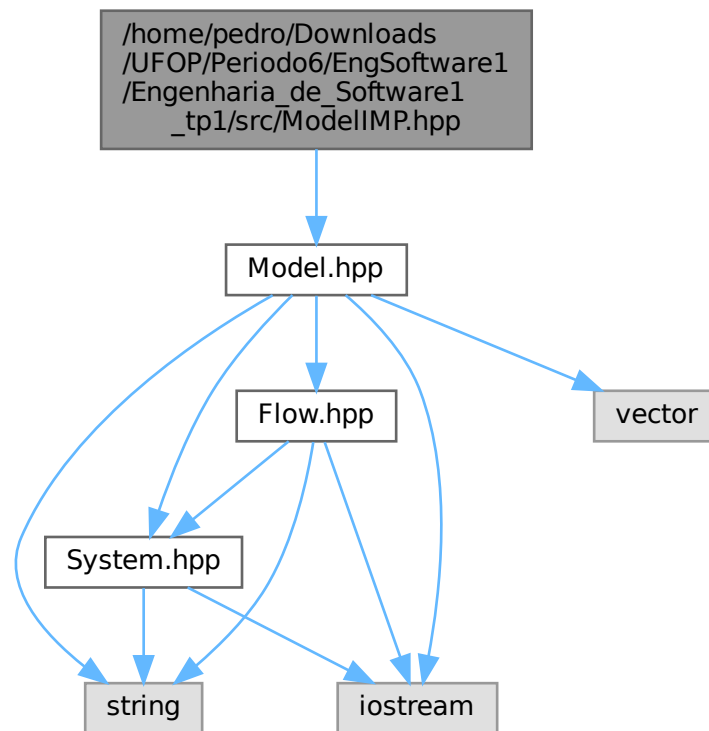
Include dependency graph for ModelIMP.cpp:



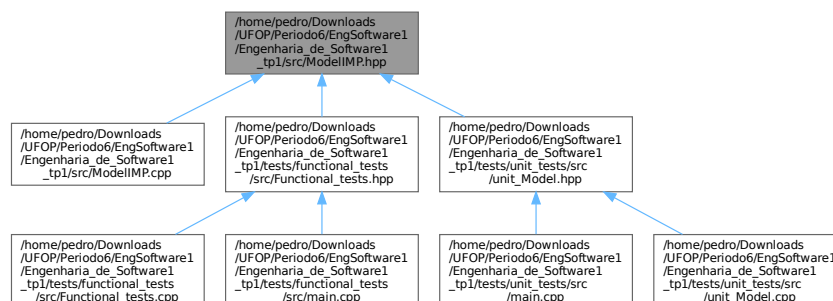
5.9 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/ModelIMP.hpp File Reference

```
#include "Model.hpp"
```

Include dependency graph for ModelIMP.hpp:



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



Classes

- class `ModelIMP`

5.10 ModelIMP.hpp

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * @file ModelIMP.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the model implementation
00005  *****/
00006
00007 #ifndef MODELIMP_HPP
00008 #define MODELIMP_HPP
00009
00010 #include "Model.hpp"
00011
00012 /*****
00013  * @brief This class implementation defines the attributes and implements the methods
00014  *****/
00015 class ModelIMP : public Model{
00016     private:
00022         ModelIMP& operator=(const ModelIMP& other); // Operador de atribuição
00027         ModelIMP(const ModelIMP& other); //Copia outro flow
00028
00029     protected:
00030         std::string name;
00031         std::vector<System*> systems;
00032         std::vector<Flow*> flows;
00033         int startTime;
00034         int endTime;
00036     public:
00037         //Constructors
00044         ModelIMP(const std::string& name = "NO_NAME", const int& startTime = 0, const int& endTime =
1);
00045
00046         //Destrutor
00050         virtual ~ModelIMP();
00051
00052         //Getters e setters
00053         //Name
00058         std::string getName() const;
00063         void setName(const std::string& name);
00064         //Vector
00069         std::vector<System*> getSystems() const;
00074         std::vector<Flow*> getFlows() const;
00079         void setSystems(const std::vector<System*> systems);
00084         void setFlows(const std::vector<Flow*> flows);
00085         //Time
00090         int getStartTime() const;
00095         int getEndTime() const;
00100         void setStartTime(const int& startTime);
00105         void setEndTime(const int& endTime);
00111         void setTime(const int& startTime, const int& endTime);
00112
00113         //Metodos
00114         //add
00119         void add(System* system);
00124         void add(Flow* flow);
00125         //remove
00131         bool rmv(const System* system);
00137         bool rmv(const Flow* flow);
00138         //Others
00143         bool run();
00144
00145         //Sobrecarga de operadores
00151         bool operator==(const ModelIMP& other) const; // Operador de igualdade
00157         bool operator!=(const ModelIMP& other) const; // Operador de igualdade
00158 };
00159
00160 #endif

```

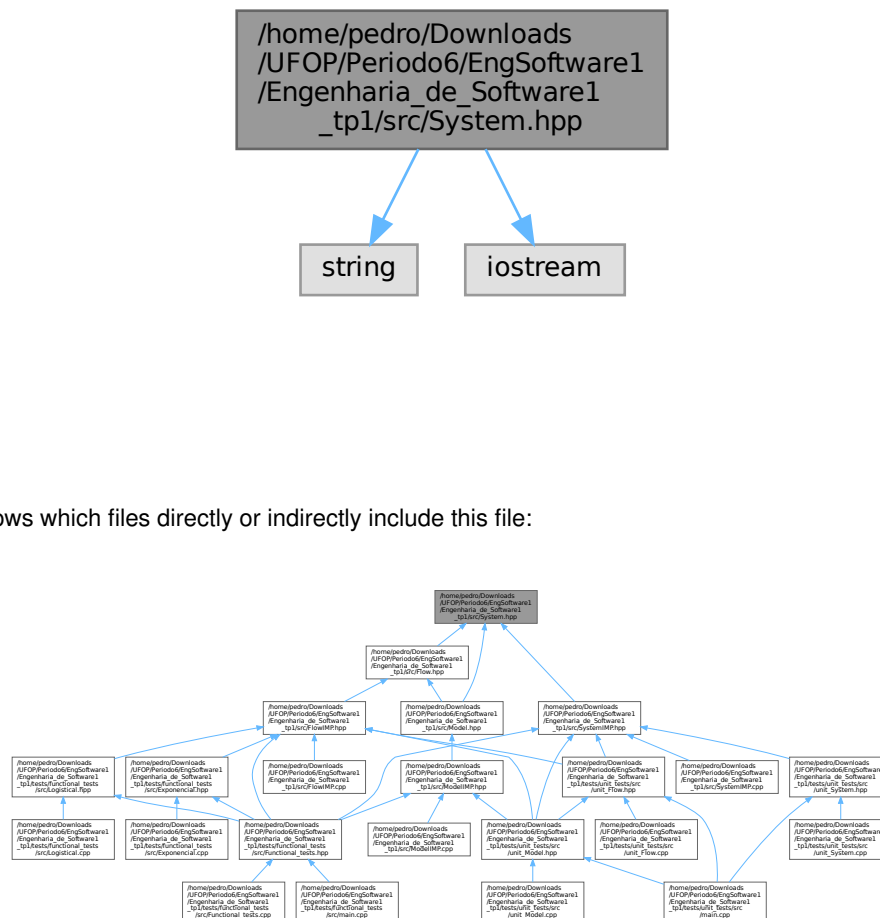
5.11 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/src/System.hpp File Reference

```

#include <string>
#include <iostream>

```

Include dependency graph for System.hpp:



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

Classes

- class [System](#)

5.12 System.hpp

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * @file System.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the System interface
00005  *****/
00006
00007 #ifndef SYSTEM_HPP
00008 #define SYSTEM_HPP
00009
00010 /*****
00011  * @brief The System Interface is the Interface that defines the methods to be implemented
00012  *****/
00013
00014 #include <string>
00015 #include <iostream>
00016

```

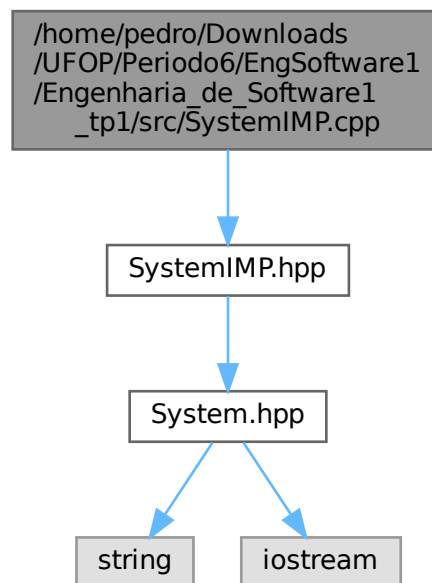
```

00017 class System{
00018     public:
00019         //Destructors
00023         virtual ~System() {};
00024
00025         //Getters e setters
00026         //Nome
00031         virtual std::string getName() const = 0;
00036         virtual void setName(const std::string& name) = 0;
00037         //Value
00042         virtual double getValue() const = 0;
00047         virtual void setValue(const double& value) = 0;
00048 };
00049
00050 #endif

```

5.13 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/src/SystemIMP.cpp File Reference

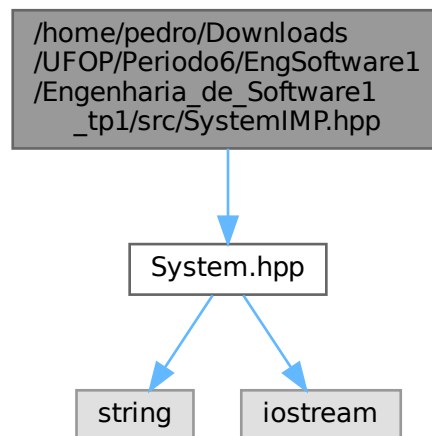
#include "SystemIMP.hpp"
Include dependency graph for SystemIMP.cpp:



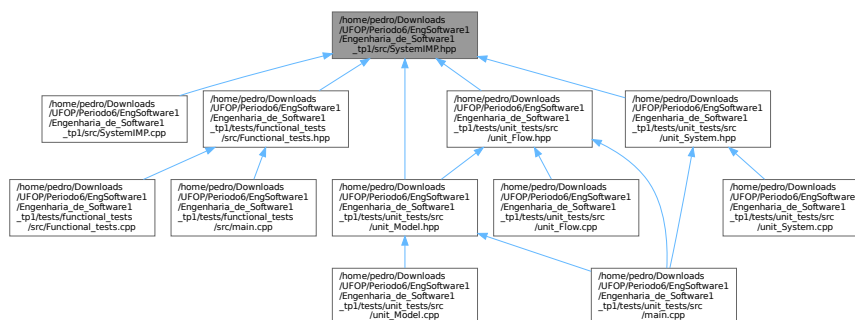
Engenharia_de_Software1_tp1/src/SystemIMP.hpp File Reference

```
#include "System.hpp"
```

Include dependency graph for SystemIMP.hpp:



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



Classes

- class [SystemIMP](#)

5.15 SystemIMP.hpp

[Go to the documentation of this file.](#)

```
00001 /*****
```

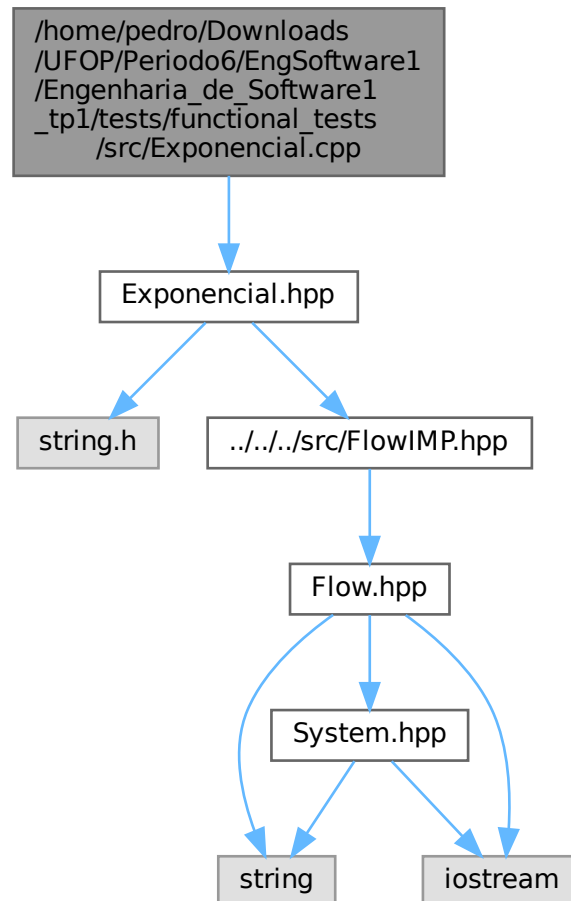
```

00002  * @file SystemIMP.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the system implementation
00005  *****/
00006
00007  #ifndef SYSTEMIMP_HPP
00008  #define SYSTEMIMP_HPP
00009
00010  //Bibliotecas
00011  #include "System.hpp"
00012
00013  /**
00014   * @brief The System implementation defines the attributes and implements the methods
00015   *****/
00016
00017  class SystemIMP : public System{
00018  private:
00023      SystemIMP(const SystemIMP& other); //Copia outro system
00029      SystemIMP& operator=(const SystemIMP& other); // Operador de atribuição
00030
00031  protected:
00032      std::string name;
00033      double value;
00035  public:
00036      //Constructors
00042      SystemIMP(const std::string& name = "NO_NAME", const double& value = 0.0);
00043
00047      //Destructors
00048      virtual ~SystemIMP();
00049
00050      //Getters e setters
00051      //Nome
00056      std::string getName() const;
00061      void setName(const std::string& name);
00062      //Value
00067      double getValue() const;
00072      void setValue(const double& value);
00073
00074      //Sobrecarga de operadores
00080      bool operator==(const SystemIMP& other) const; // Operador de igualdade
00086      bool operator!=(const SystemIMP& other) const; // Operador de diferença
00087  };
00088
00089  #endif

```

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

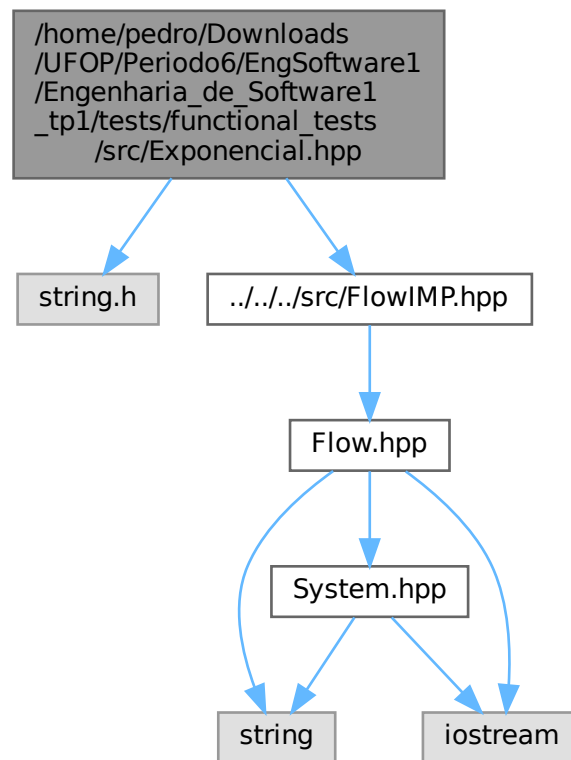
```
#include "Exponencial.hpp"  
Include dependency graph for Exponencial.cpp:
```



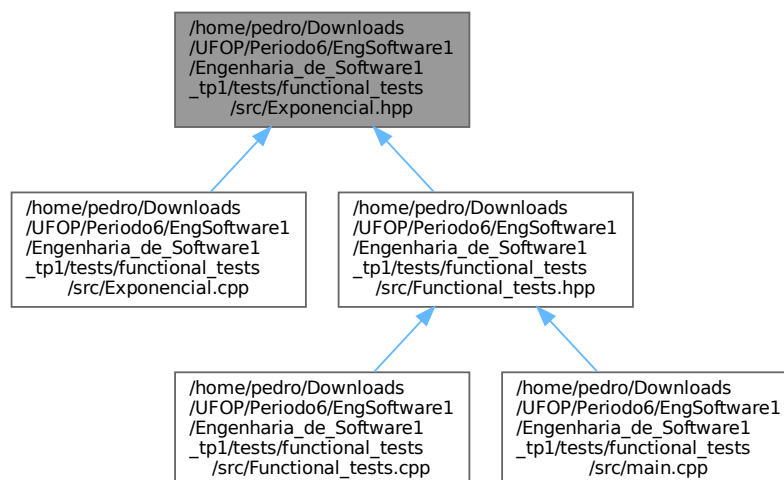
5.17 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/functional_tests/src/↵ Exponencial.hpp File Reference

```
#include <string.h>  
#include "../../src/FlowIMP.hpp"
```

Include dependency graph for Exponencial.hpp:



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



Classes

- class [Exponencial](#)

5.18 Exponencial.hpp

[Go to the documentation of this file.](#)

```

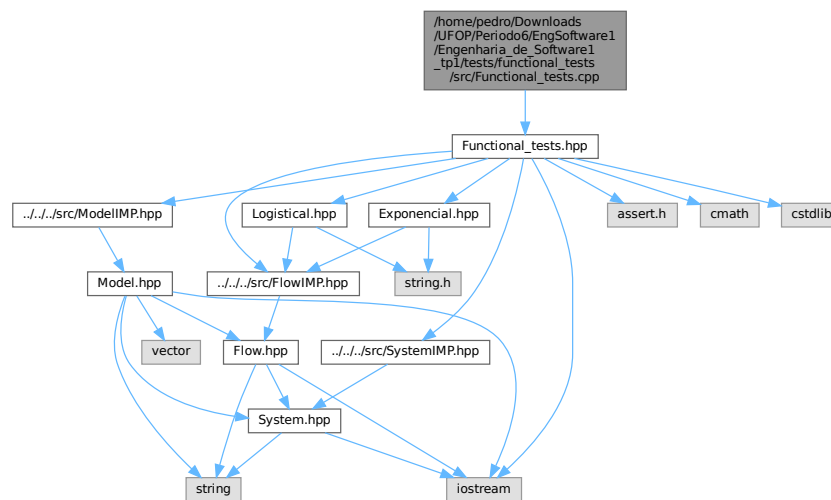
00001 /*****
00002  * @file Exponencial.h
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the exponential simulation flow
00005  *****/
00006
00007 #ifndef EXPONENCIAL_HPP
00008 #define EXPONENCIAL_HPP
00009
00010 #include <string.h>
00011 #include "../src/FlowIMP.hpp"
00012
00013 /*****
00014  * @brief This Flow class connects two systems and through the entered equation transfers values from
00015  one system to another
00016  *****/
00017 class Exponencial : public FlowIMP{
00018     private:
00023         Exponencial(const Exponencial& other);
00024
00025     public:
00026         //Constructor
00033         Exponencial(const std::string& name = "NO_NAME", System* source = NULL, System* target =
00034         NULL);
00035
00036         //Destructor
00039         virtual ~Exponencial();
00040
00041         //Metodos
00046         virtual double execute() override;
00047 };
00048
00049 #endif

```

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

```
#include "Functional_tests.hpp"
```

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.

5.19.1 Function Documentation

5.19.1.1 Complex_test_run()

```
void Complex_test_run ( )
```

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

```

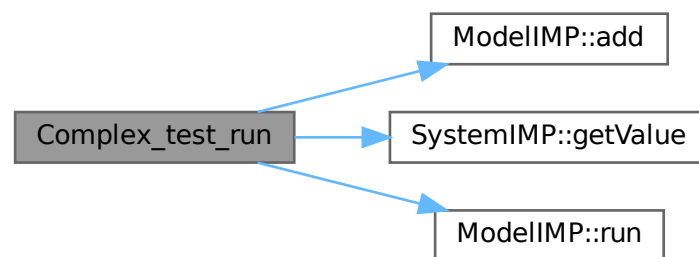
00057     {
00058         std::cout << "   Complex functional test" << std::endl;
00059
00060         ModelIMP* model = new ModelIMP("Model", 0, 100);
00061         SystemIMP* q1 = new SystemIMP("q1", 100.0);
00062         SystemIMP* q2 = new SystemIMP("q2", 0.0);
00063         SystemIMP* q3 = new SystemIMP("q3", 100.0);
00064         SystemIMP* q4 = new SystemIMP("q4", 0.0);
00065         SystemIMP* q5 = new SystemIMP("q5", 0.0);
00066         Exponencial* f = new Exponencial("f", q1, q2);
00067         Exponencial* t = new Exponencial("t", q2, q3);
00068         Exponencial* u = new Exponencial("u", q3, q4);
00069         Exponencial* v = new Exponencial("v", q4, q1);
  
```

```
00070     Exponencial* g = new Exponencial("g", q1, q3);
00071     Exponencial* r = new Exponencial("r", q2, q5);
00072
00073     model->add(q1);
00074     model->add(q2);
00075     model->add(q3);
00076     model->add(q4);
00077     model->add(q5);
00078     model->add(f);
00079     model->add(t);
00080     model->add(u);
00081     model->add(v);
00082     model->add(g);
00083     model->add(r);
00084
00085     model->run();
00086
00087     assert(fabs((round((q1->getValue() * 10000)) - 10000 * 31.8513)) < 0.0001);
00088     assert(fabs((round((q2->getValue() * 10000)) - 10000 * 18.4003)) < 0.0001);
00089     assert(fabs((round((q3->getValue() * 10000)) - 10000 * 77.1143)) < 0.0001);
00090     assert(fabs((round((q4->getValue() * 10000)) - 10000 * 56.1728)) < 0.0001);
00091     assert(fabs((round((q5->getValue() * 10000)) - 10000 * 16.4612)) < 0.0001);
00092
00093     delete model;
00094     delete q1;
00095     delete q2;
00096     delete q3;
00097     delete q4;
00098     delete q5;
00099     delete f;
00100     delete t;
00101     delete u;
00102     delete v;
00103     delete g;
00104     delete r;
00105
00106     std::cout << "   Complex functional test passed" << std::endl;
00107 }
```

References [ModelIMP::add\(\)](#), [SystemIMP::getValue\(\)](#), and [ModelIMP::run\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.19.1.2 exponencial_test_run()

```
void exponencial_test_run ( )
```

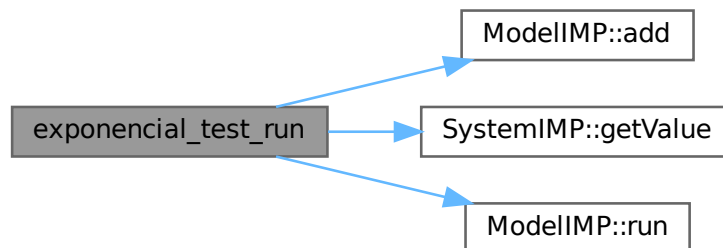
This function performs the exponential functional test.

```
00003      {
00004          std::cout << "   Exponencial functional test" << std::endl;
00005
00006          SystemIMP* pop1 = new SystemIMP("pop1", 100.0);
00007          SystemIMP* pop2 = new SystemIMP("pop2", 0.0);
00008          Exponencial* exp = new Exponencial("exp", pop1, pop2);
00009          ModelIMP* exponencial = new ModelIMP("Exponencial", 0, 100);
00010
00011          //Add os systems e flows ao modelo
00012          exponencial->add(pop1);
00013          exponencial->add(pop2);
00014          exponencial->add(exp);
00015
00016          //Roda o modelo
00017          exponencial->run();
00018
00019          assert(fabs((round(pop1->getValue() * 10000) - 10000 * 36.6032)) < 0.0001);
00020          assert(fabs((round(pop2->getValue() * 10000) - 10000 * 63.3968)) < 0.0001);
00021
00022          delete(exponencial);
00023          delete(exp);
00024          delete(pop1);
00025          delete(pop2);
00026
00027          std::cout << "   Exponencial functional test passed\n" << std::endl;
00028      }
```

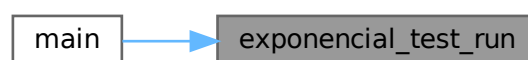
References [ModelIMP::add\(\)](#), [SystemIMP::getValue\(\)](#), and [ModelIMP::run\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.19.1.3 logistical_test_run()

```
void logistical_test_run ( )
```

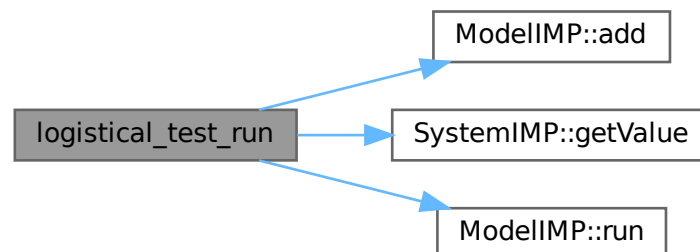
This function performs the logistic test.

```
00030     {
00031         std::cout << "   Logistical functional test" << std::endl;
00032
00033         SystemIMP* p1 = new SystemIMP("p1", 100.0);
00034         SystemIMP* p2 = new SystemIMP("p2", 10.0);
00035         Logistical* log = new Logistical("log", p1, p2);
00036         ModelIMP* logistical = new ModelIMP("Logistical", 0, 100);
00037
00038         //Add os systems e flows ao modelo
00039         logistical->add(p1);
00040         logistical->add(p2);
00041         logistical->add(log);
00042
00043         //Roda o modelo
00044         logistical->run();
00045
00046         assert(fabs(round(p1->getValue() * 10000) - 10000 * 88.2167) < 0.0001);
00047         assert(fabs(round(p2->getValue() * 10000) - 10000 * 21.7833) < 0.0001);
00048
00049         delete(logistical);
00050         delete(log);
00051         delete(p1);
00052         delete(p2);
00053
00054         std::cout << "   Logistical functional test passed\n" << std::endl;
00055     }
```

References [ModelIMP::add\(\)](#), [SystemIMP::getValue\(\)](#), and [ModelIMP::run\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



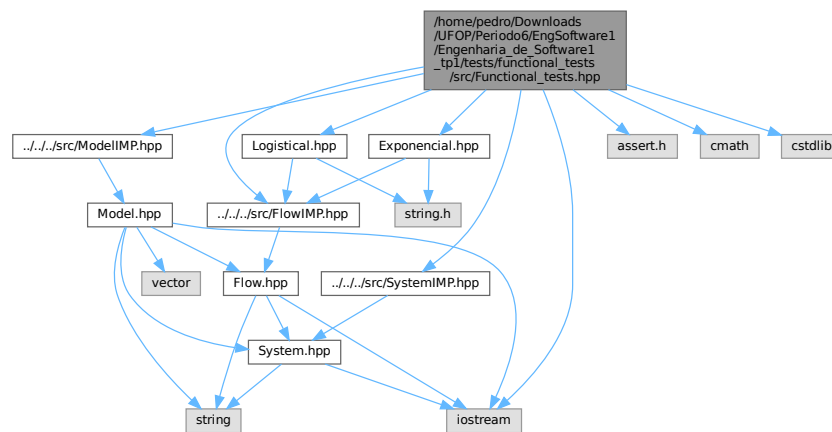
Here is the caller graph for this function:



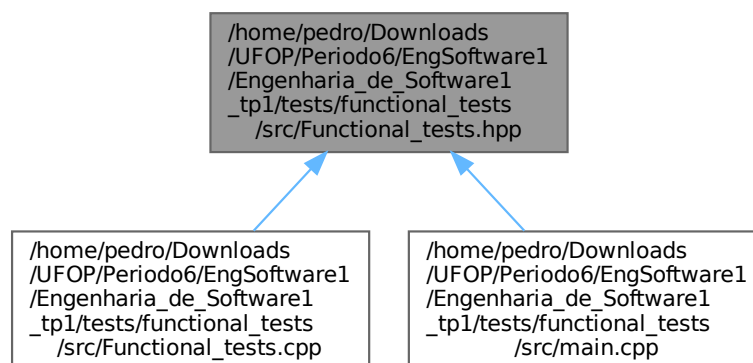
5.20 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/functional_tests/src/↵ Functional_tests.hpp File Reference

```
#include "../.../src/ModelIMP.hpp"
#include "../.../src/SystemIMP.hpp"
#include "../.../src/FlowIMP.hpp"
#include "Exponencial.hpp"
#include "Logistical.hpp"
#include <assert.h>
#include <cmath>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for Functional_tests.hpp:



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



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.

5.20.1 Function Documentation

5.20.1.1 Complex_test_run()

```
void Complex_test_run ( )
```

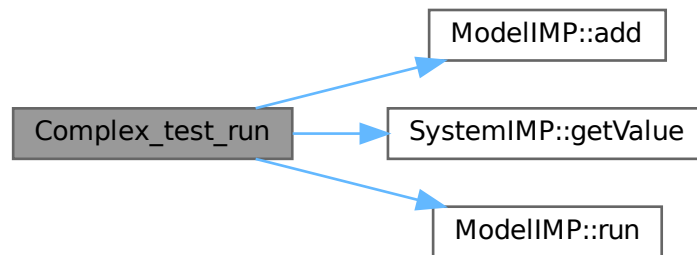
This function runs the "complex" test, which has multiple systems and flows.

```
00057     {
00058         std::cout << "   Complex functional test" << std::endl;
00059
00060         ModelIMP* model = new ModelIMP("Model", 0, 100);
00061         SystemIMP* q1 = new SystemIMP("q1", 100.0);
00062         SystemIMP* q2 = new SystemIMP("q2", 0.0);
00063         SystemIMP* q3 = new SystemIMP("q3", 100.0);
00064         SystemIMP* q4 = new SystemIMP("q4", 0.0);
00065         SystemIMP* q5 = new SystemIMP("q5", 0.0);
00066         Exponencial* f = new Exponencial("f", q1, q2);
00067         Exponencial* t = new Exponencial("t", q2, q3);
00068         Exponencial* u = new Exponencial("u", q3, q4);
00069         Exponencial* v = new Exponencial("v", q4, q1);
00070         Exponencial* g = new Exponencial("g", q1, q3);
00071         Exponencial* r = new Exponencial("r", q2, q5);
00072
00073         model->add(q1);
00074         model->add(q2);
00075         model->add(q3);
00076         model->add(q4);
00077         model->add(q5);
00078         model->add(f);
00079         model->add(t);
00080         model->add(u);
00081         model->add(v);
00082         model->add(g);
00083         model->add(r);
00084
00085         model->run();
00086
00087         assert(fabs((round((q1->getValue() * 10000)) - 10000 * 31.8513)) < 0.0001);
00088         assert(fabs((round((q2->getValue() * 10000)) - 10000 * 18.4003)) < 0.0001);
00089         assert(fabs((round((q3->getValue() * 10000)) - 10000 * 77.1143)) < 0.0001);
00090         assert(fabs((round((q4->getValue() * 10000)) - 10000 * 56.1728)) < 0.0001);
00091         assert(fabs((round((q5->getValue() * 10000)) - 10000 * 16.4612)) < 0.0001);
00092
00093         delete model;
00094         delete q1;
00095         delete q2;
00096         delete q3;
00097         delete q4;
00098         delete q5;
00099         delete f;
00100         delete t;
00101         delete u;
00102         delete v;
00103         delete g;
00104         delete r;
00105
00106         std::cout << "   Complex functional test passed" << std::endl;
00107     }
```

References [ModelIMP::add\(\)](#), [SystemIMP::getValue\(\)](#), and [ModelIMP::run\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.20.1.2 exponencial_test_run()

```
void exponencial_test_run ( )
```

This function performs the exponential functional test.

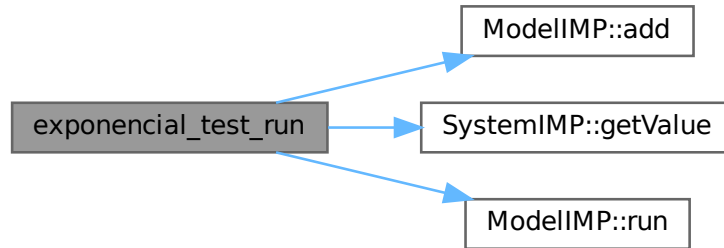
```

00003      {
00004      std::cout << "   Exponencial functional test" << std::endl;
00005
00006      SystemIMP* pop1 = new SystemIMP("pop1", 100.0);
00007      SystemIMP* pop2 = new SystemIMP("pop2", 0.0);
00008      Exponencial* exp = new Exponencial("exp", pop1, pop2);
00009      ModelIMP* exponencial = new ModelIMP("Exponencial", 0, 100);
00010
00011      //Add os systems e flows ao modelo
00012      exponencial->add(pop1);
00013      exponencial->add(pop2);
00014      exponencial->add(exp);
00015
00016      //Roda o modelo
00017      exponencial->run();
00018
00019      assert(fabs((round(pop1->getValue() * 10000) - 10000 * 36.6032)) < 0.0001);
00020      assert(fabs((round(pop2->getValue() * 10000) - 10000 * 63.3968)) < 0.0001);
00021
00022      delete(exponencial);
00023      delete(exp);
00024      delete(pop1);
00025      delete(pop2);
00026
00027      std::cout << "   Exponencial functional test passed\n" << std::endl;
00028  }
```

References [ModelIMP::add\(\)](#), [SystemIMP::getValue\(\)](#), and [ModelIMP::run\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.20.1.3 logistical_test_run()

```
void logistical_test_run ( )
```

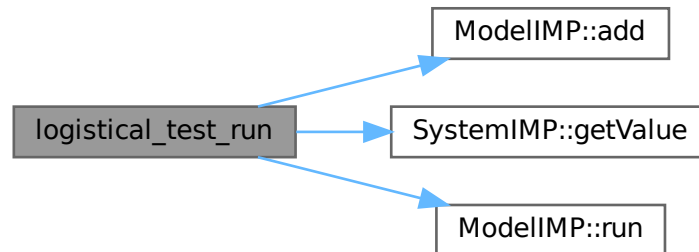
This function performs the logistic test.

```
00030     {
00031         std::cout << "    Logistical functional test" << std::endl;
00032
00033         SystemIMP* p1 = new SystemIMP("p1", 100.0);
00034         SystemIMP* p2 = new SystemIMP("p2", 10.0);
00035         Logistical* log = new Logistical("log", p1, p2);
00036         ModelIMP* logistical = new ModelIMP("Logistical", 0, 100);
00037
00038         //Add os systems e flows ao modelo
00039         logistical->add(p1);
00040         logistical->add(p2);
00041         logistical->add(log);
00042
00043         //Roda o modelo
00044         logistical->run();
00045
00046         assert(fabs(round(p1->getValue() * 10000) - 10000 * 88.2167) < 0.0001);
00047         assert(fabs(round(p2->getValue() * 10000) - 10000 * 21.7833) < 0.0001);
00048
00049         delete(logistical);
00050         delete(log);
00051         delete(p1);
00052         delete(p2);
00053
00054         std::cout << "    Logistical functional test passed\n" << std::endl;
00055     }
```

References [ModelIMP::add\(\)](#), [SystemIMP::getValue\(\)](#), and [ModelIMP::run\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.21 Functional_tests.hpp

[Go to the documentation of this file.](#)

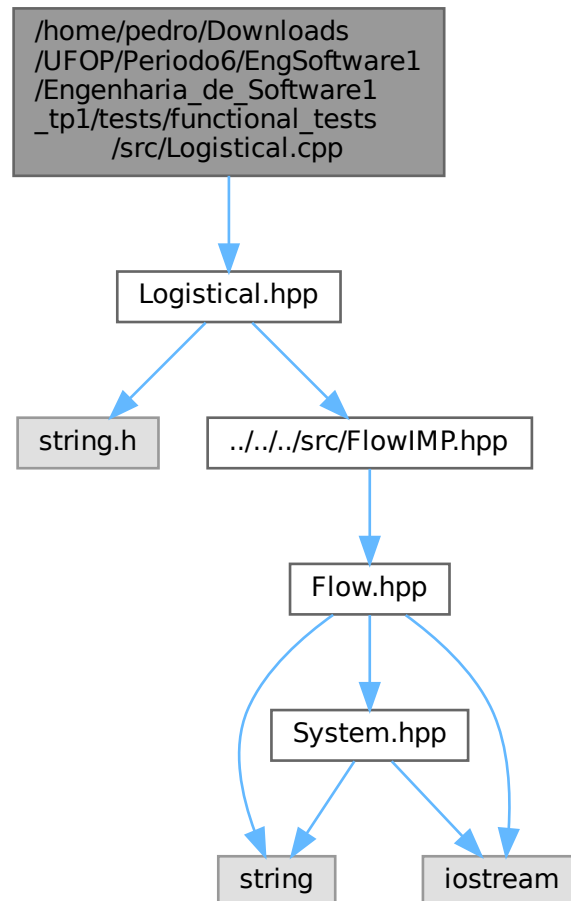
```

00001 /*****
00002  * @file Exponencial.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the logistical simulation flow
00005  *****/
00006
00007 #ifndef FUNCTIONAL_TESTS_HPP
00008 #define FUNCTIONAL_TESTS_HPP
00009
00010 #include "../src/ModelIMP.hpp"
00011 #include "../src/SystemIMP.hpp"
00012 #include "../src/FlowIMP.hpp"
00013 #include "Exponencial.hpp"
00014 #include "Logistical.hpp"
00015 #include <assert.h>
00016 #include <cmath>
00017 #include <iostream>
00018 #include <cstdlib>
00019
00020 /*****
00021  * @brief execution of functional tests
00022  *****/
00023
00027 void exponencial_test_run();
00028
00032 void logistical_test_run();
00033
00037 void Complex_test_run();
00038
00039 #endif
  
```

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

```
#include "Logistical.hpp"
```

Include dependency graph for Logistical.cpp:

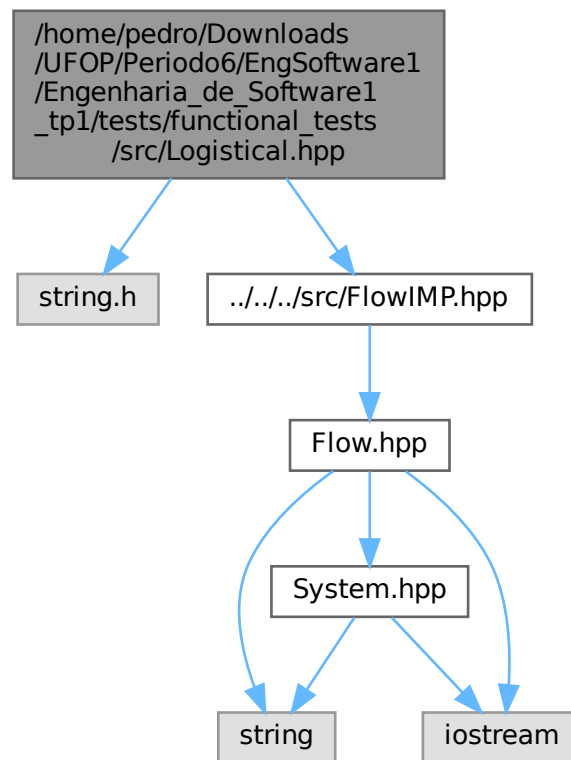


5.23 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/functional_tests/src/↵ Logistical.hpp File Reference

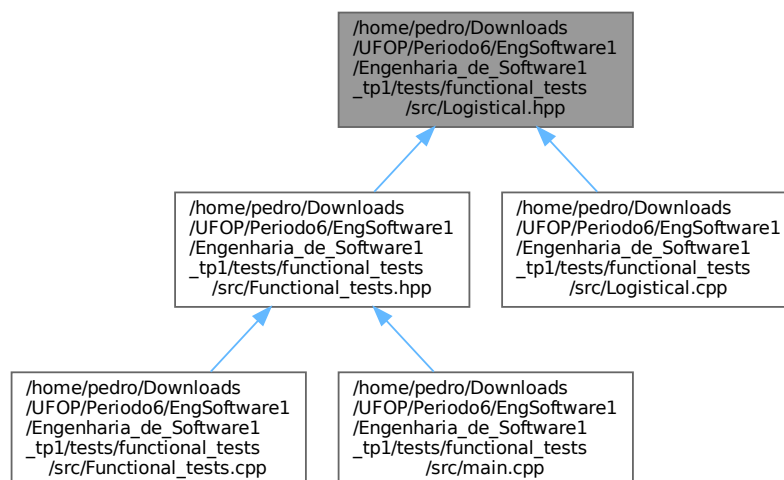
```
#include <string.h>
```

```
#include "../../src/FlowIMP.hpp"
```

Include dependency graph for Logistical.hpp:



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



Classes

- class [Logistical](#)

5.24 Logistical.hpp

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * @file Logistical.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the logistical simulation flow
00005  *****/
00006
00007 #ifndef LOGISTICAL_HPP
00008 #define LOGISTICAL_HPP
00009
00010 #include <string.h>
00011 #include "../src/FlowIMP.hpp"
00012
00013 /*****
00014  * @brief This Flow class connects two systems and through the entered equation transfers values from
00015  one system to another
00016  *****/
00017
00018 class Logistical : public FlowIMP{
00019     private:
00020         Logistical(const Logistical& other);
00021
00022     public:
00023         //Constructor
00024         Logistical(const std::string& name = "NO_NAME", System* source = NULL, System* target = NULL);
00025
00026         //Destructor
00027         virtual ~Logistical();
00028
00029         //Metodos
00030         virtual double execute() override;
00031 };
00032
00033 #endif

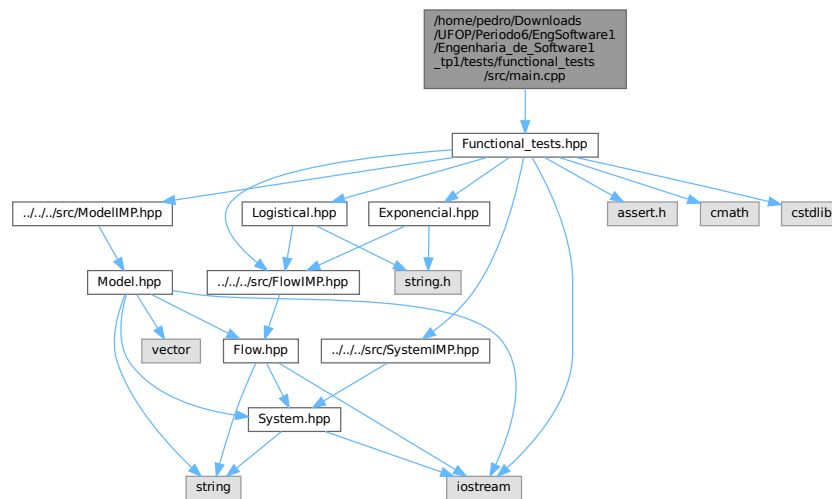
```

5.25 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/functional_tests/src/main.cpp

File Reference

```
#include "Functional_tests.hpp"
```

Include dependency graph for main.cpp:



Functions

- int [main](#) ()

5.25.1 Function Documentation

5.25.1.1 main()

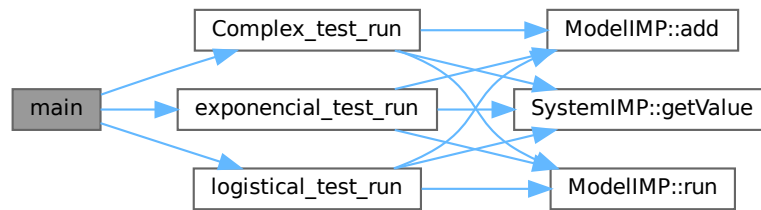
```

int main ( )
00003     {
00004         std::cout << "\nStart functional tests\n"
00005                 << "*****\n";
00006         exponencial_test_run();
00007         logistical_test_run();
00008         Complex_test_run();
00009         std::cout << "*****\n"
00010                 << "End functional tests\n\n";
00011         return 0;
00012     }

```

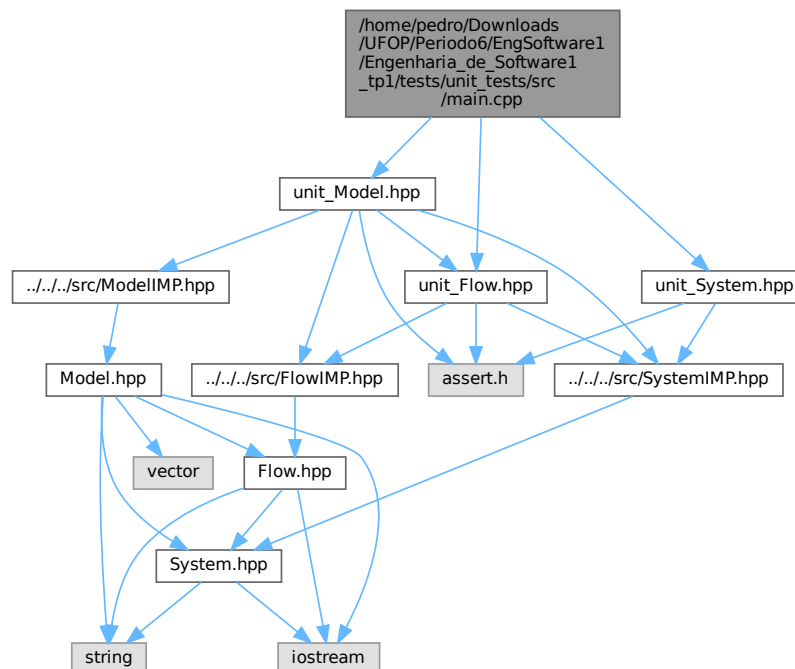
References [Complex_test_run\(\)](#), [exponencial_test_run\(\)](#), and [logistical_test_run\(\)](#).

Here is the call graph for this function:



5.26 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/unit_tests/src/main.cpp File Reference

```
#include "unit_System.hpp"
#include "unit_Flow.hpp"
#include "unit_Model.hpp"
Include dependency graph for main.cpp:
```



Functions

- int [main](#) ()

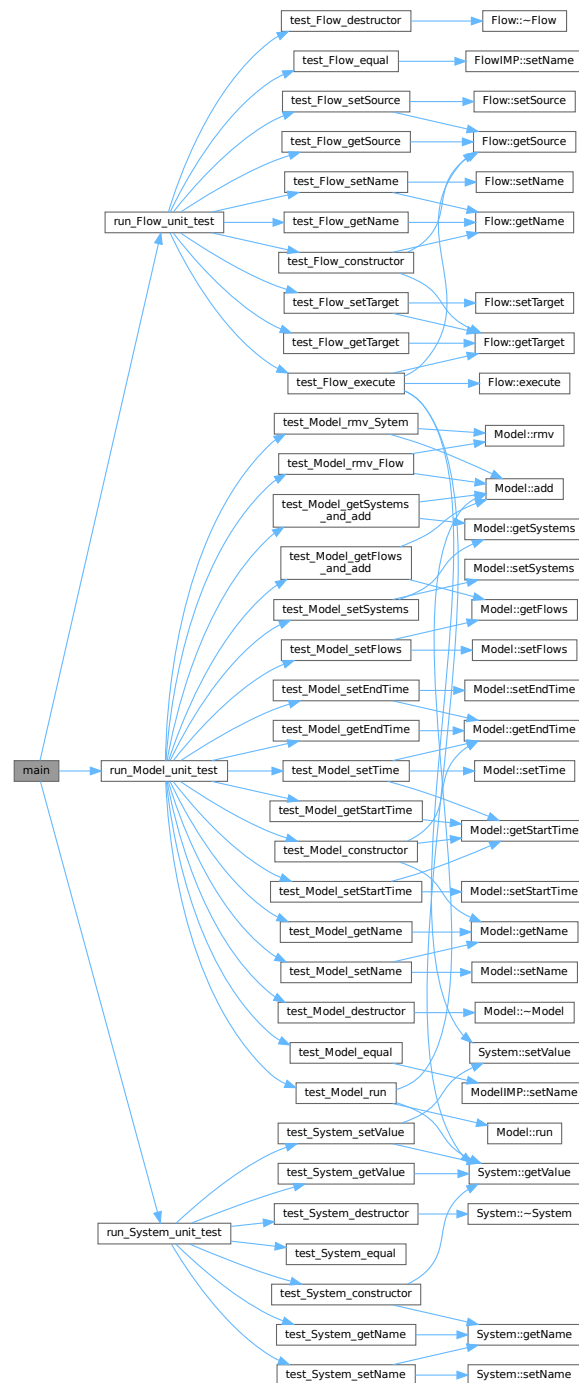
5.26.1 Function Documentation

5.26.1.1 main()

```
int main ( )
00005     {
00006     std::cout << "\nStart unit tests\n"
00007               << "*****\n";
00008     run_System_unit_test();
00009     run_Flow_unit_test();
00010     run_Model_unit_test();
00011     std::cout << "*****\n"
00012               << "End unit tests\n\n";
00013     return 0;
00014 }
```

References [run_Flow_unit_test\(\)](#), [run_Model_unit_test\(\)](#), and [run_System_unit_test\(\)](#).

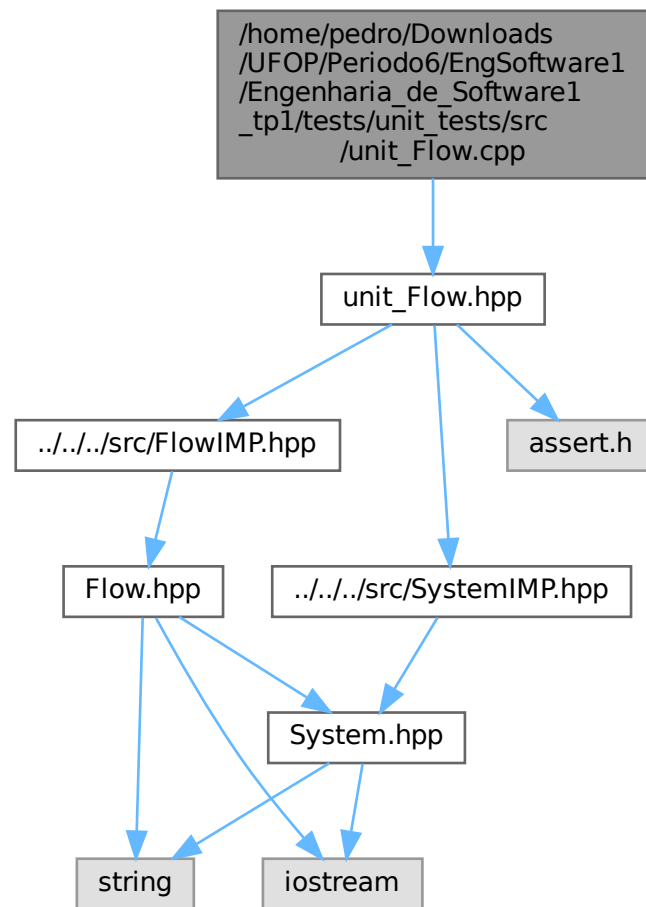
Here is the call graph for this function:



5.27 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_Flow.cpp File Reference

```
#include "unit_Flow.hpp"
```

Include dependency graph for `unit_Flow.cpp`:



Functions

- void `test_Flow_constructor()`
This function run the unit test of the flow constructor.
- void `test_Flow_destructor()`
This function run the unit test of the flow destructor.
- void `test_Flow_getName()`
This function run the unit test of the flow getName.
- void `test_Flow_getSource()`
This function run the unit test of the flow getSource.
- void `test_Flow_getTarget()`
This function run the unit test of the flow getTarge.
- void `test_Flow_setName()`
This function run the unit test of the flow setName.
- void `test_Flow_setSource()`
This function run the unit test of the flow setSource.

- void [test_Flow_setTarget\(\)](#)
This function run the unit test of the flow setTarge.
- void [test_Flow_execute\(\)](#)
This function run the unit test of the flow execute.
- void [test_Flow_equal\(\)](#)
This function run the unit test of the flow equal comparison.
- void [run_Flow_unit_test\(\)](#)
This function run the unit tests of the flow.

5.27.1 Function Documentation

5.27.1.1 run_Flow_unit_test()

```
void run_Flow_unit_test ( )
```

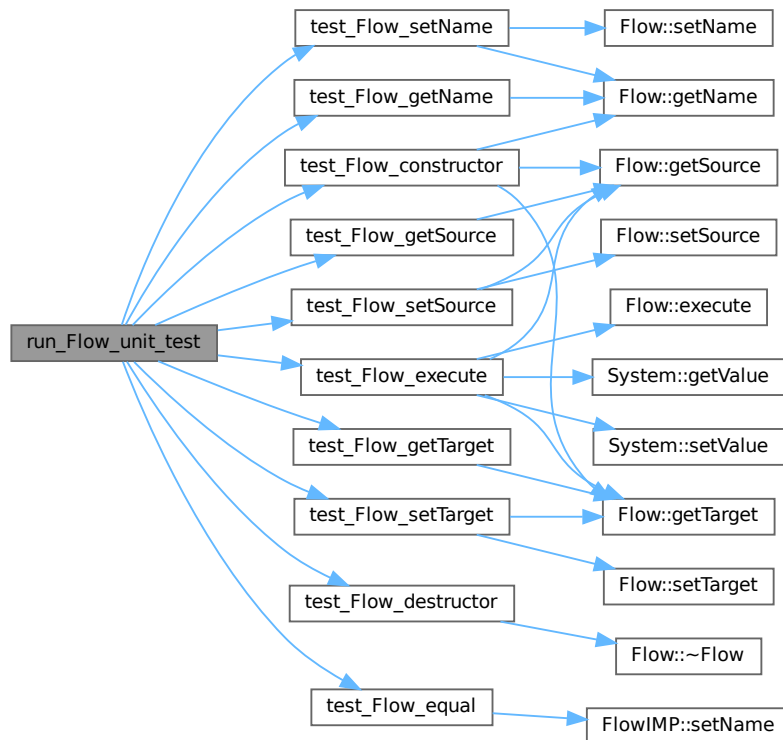
This function run the unit tests of the flow.

```
00189         {  
00190             std::cout << "    Start Flow unit tests\n";  
00191             test\_Flow\_constructor\(\);  
00192             test\_Flow\_destructor\(\);  
00193             test\_Flow\_getName\(\);  
00194             test\_Flow\_getSource\(\);  
00195             test\_Flow\_getTarget\(\);  
00196             test\_Flow\_setName\(\);  
00197             test\_Flow\_setSource\(\);  
00198             test\_Flow\_setTarget\(\);  
00199             test\_Flow\_execute\(\);  
00200             test\_Flow\_equal\(\);  
00201             std::cout << "    End Flow unit tests\n\n";  
00202     }
```

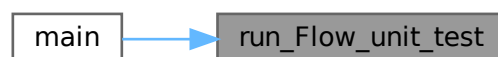
References [test_Flow_constructor\(\)](#), [test_Flow_destructor\(\)](#), [test_Flow_equal\(\)](#), [test_Flow_execute\(\)](#), [test_Flow_getName\(\)](#), [test_Flow_getSource\(\)](#), [test_Flow_getTarget\(\)](#), [test_Flow_setName\(\)](#), [test_Flow_setSource\(\)](#), and [test_Flow_setTarget\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.2 test_Flow_constructor()

```
void test_Flow_constructor ( )
```

This function run the unit test of the flow constructor.

```

00023     {
00024     std::cout << "          * Constructor tests\n";
00025     Flow* f1 = new Flow_unit_test();
00026     assert(f1->getName() == "NO_NAME");
00027     assert(f1->getSource() == NULL);
00028     assert(f1->getTarget() == NULL);
00029
00030     Flow* f2 = new Flow_unit_test("f2");
00031     assert(f2->getName() == "f2");
  
```



```

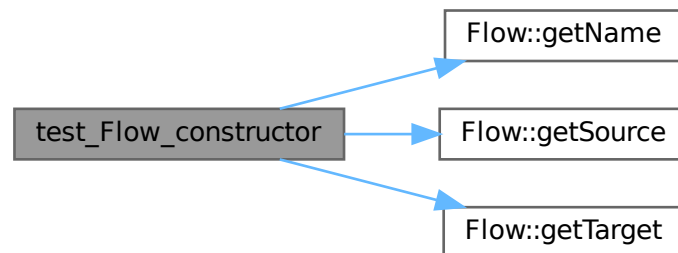
00032     assert(f2->getSource() == NULL);
00033     assert(f2->getTarget() == NULL);
00034
00035     System* s1 = new SystemIMP();
00036     Flow* f3 = new Flow_unit_test("f3", s1);
00037     assert(f3->getName() == "f3");
00038     assert(f3->getSource() == s1);
00039     assert(f3->getTarget() == NULL);
00040
00041     System* s2 = new SystemIMP();
00042     System* s3 = new SystemIMP();
00043     Flow* f4 = new Flow_unit_test("f4", s2, s3);
00044     assert(f4->getName() == "f4");
00045     assert(f4->getSource() == s2);
00046     assert(f4->getTarget() == s3);
00047
00048     delete f1;
00049     delete f2;
00050     delete s1;
00051     delete f3;
00052     delete s2;
00053     delete s3;
00054     delete f4;
00055 }

```

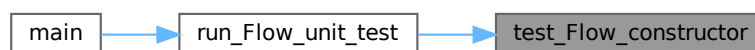
References [Flow::getName\(\)](#), [Flow::getSource\(\)](#), and [Flow::getTarget\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.3 test_Flow_destructor()

```
void test_Flow_destructor ( )
```

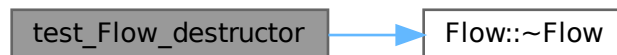
This function run the unit test of the flow destructor.

```
00057     {
00058         std::cout << "          * Destructor tests\n";
00059         Flow* f1 = new Flow_unit_test();
00060         f1->~Flow();
00061     }
```

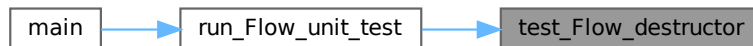
References [Flow::~~Flow\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.4 test_Flow_equal()

```
void test_Flow_equal ( )
```

This function run the unit test of the flow equal comparison.

```
00175     {
00176         std::cout << "          * Equal tests\n";
00177
00178         FlowIMP* f1 = new Flow_unit_test();
00179         FlowIMP* f2 = new Flow_unit_test();
00180         assert(*f1 == *f2);
00181
00182         f1->setName("f1");
00183         assert(*f1 != *f2);
00184
00185         delete f1;
00186         delete f2;
00187     }
```

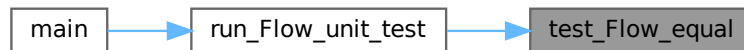
References [FlowIMP::setName\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.5 test_Flow_execute()

```
void test_Flow_execute ( )
```

This function run the unit test of the flow execute.

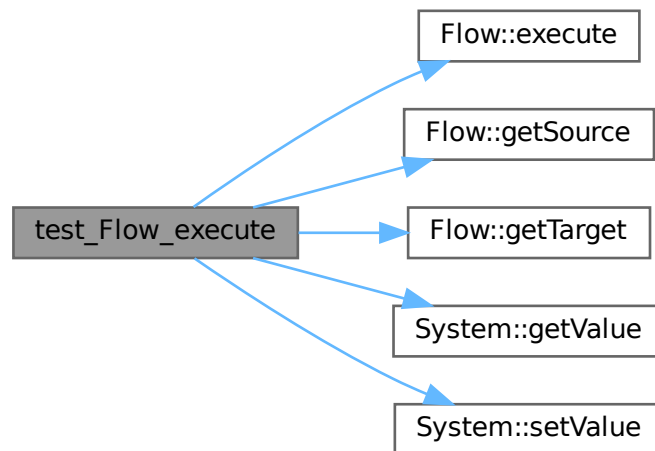
```

00157     {
00158         std::cout << "          * Execute tests\n";
00159
00160         System* s1 = new SystemIMP("s1", 100);
00161         System* s2 = new SystemIMP("s2", 0.0);
00162         Flow* f1 = new Flow_unit_test("f1", s1, s2);
00163         double result = f1->execute();
00164         f1->getTarget()->setValue(f1->getTarget()->getValue() + result);
00165         f1->getSource()->setValue(f1->getSource()->getValue() - result);
00166
00167         assert(s1->getValue() == 0);
00168         assert(s2->getValue() == 100);
00169
00170         delete s1;
00171         delete s2;
00172         delete f1;
00173     }
  
```

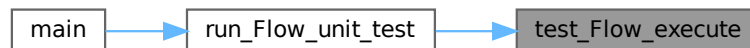
References [Flow::execute\(\)](#), [Flow::getSource\(\)](#), [Flow::getTarget\(\)](#), [System::getValue\(\)](#), and [System::setValue\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.6 test_Flow_getName()

```
void test_Flow_getName ( )
```

This function run the unit test of the flow getName.

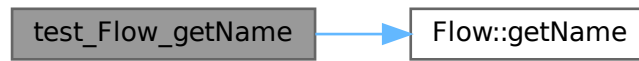
```

00063     {
00064         std::cout << "          * getName tests\n";
00065         Flow* f1 = new Flow_unit_test();
00066         assert(f1->getName() == "NO_NAME");
00067
00068         Flow* f2 = new Flow_unit_test("f2");
00069         assert(f2->getName() == "f2");
00070
00071         delete f1;
00072         delete f2;
00073     }
  
```

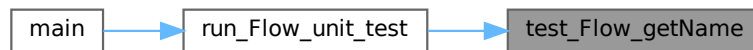
References [Flow::getName\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.7 test_Flow_getSource()

```
void test_Flow_getSource ( )
```

This function run the unit test of the flow getSource.

```

00075     {
00076     std::cout << "          * getSource tests\n";
00077     Flow* f1 = new Flow_unit_test();
00078     assert(f1->getSource() == NULL);
00079
00080     System* s1 = new SystemIMP();
00081     Flow* f2 = new Flow_unit_test("f2", s1);
00082     assert(f2->getSource() == s1);
00083
00084     delete f1;
00085     delete s1;
00086     delete f2;
00087 }
```

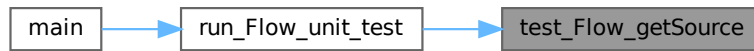
References [Flow::getSource\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.8 test_Flow_getTarget()

```
void test_Flow_getTarget ( )
```

This function run the unit test of the flow getTarge.

```

00089      {
00090          std::cout << "          * getTarget tests\n";
00091          Flow* f1 = new Flow_unit_test();
00092          assert(f1->getTarget() == NULL);
00093
00094          System* s1 = new SystemIMP();
00095          System* s2 = new SystemIMP();
00096          Flow* f2 = new Flow_unit_test("f2", s1, s2);
00097          assert(f2->getTarget() == s2);
00098
00099          delete f1;
00100          delete s1;
00101          delete s2;
00102          delete f2;
00103  }
```

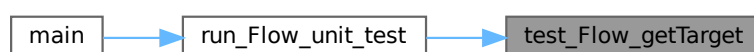
References [Flow::getTarget\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.9 test_Flow_setName()

```
void test_Flow_setName ( )
```

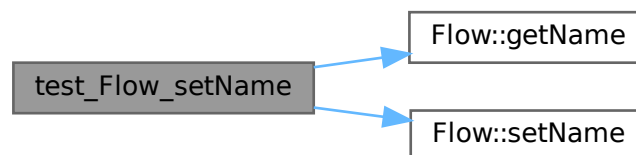
This function run the unit test of the flow setName.

```
00105         {
00106     std::cout << "          * setName tests\n";
00107     Flow* f1 = new Flow_unit_test();
00108     f1->setName("f1");
00109     assert(f1->getName() != "NO_NAME");
00110
00111     Flow* f2 = new Flow_unit_test("f");
00112     f2->setName("f2");
00113     assert(f2->getName() == "f2");
00114
00115     delete f1;
00116     delete f2;
00117 }
```

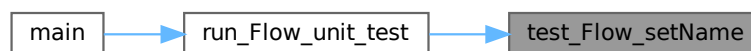
References [Flow::getName\(\)](#), and [Flow::setName\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.10 test_Flow_setSource()

```
void test_Flow_setSource ( )
```

This function run the unit test of the flow setSource.

```
00119         {
00120     std::cout << "          * setSource tests\n";
00121     System* s1 = new SystemIMP();
00122     Flow* f1 = new Flow_unit_test();
00123     f1->setSource(s1);
00124     assert(f1->getSource() != NULL);
00125 }
```

```

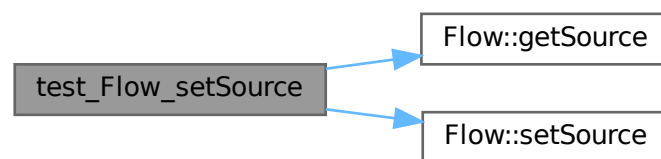
00126     System* s2 = new SystemIMP();
00127     Flow* f2 = new Flow_unit_test("f2", s1);
00128     f2->setSource(s2);
00129     assert(f2->getSource() == s2);
00130
00131     delete s1;
00132     delete s2;
00133     delete f1;
00134     delete f2;
00135 }

```

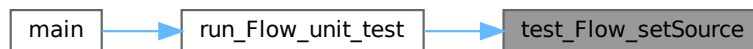
References [Flow::getSource\(\)](#), and [Flow::setSource\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.1.11 test_Flow_setTarget()

```
void test_Flow_setTarget ( )
```

This function run the unit test of the flow setTarge.

```

00137     {
00138         std::cout << "          * setTarget tests\n";
00139         System* s1 = new SystemIMP();
00140         Flow* f1 = new Flow_unit_test();
00141         f1->setTarget(s1);
00142         assert(f1->getTarget() != NULL);
00143
00144         System* s2 = new SystemIMP();
00145         System* s3 = new SystemIMP();
00146         Flow* f2 = new Flow_unit_test("f2", s1, s2);
00147         f2->setTarget(s3);
00148         assert(f2->getTarget() == s3);
00149
00150         delete s1;
00151         delete s2;
00152         delete s3;
00153         delete f1;

```

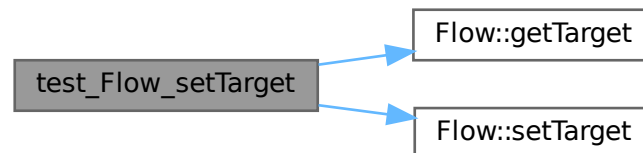


```
00154     delete f2;  
00155 }
```

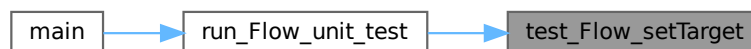
References [Flow::getTarget\(\)](#), and [Flow::setTarget\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



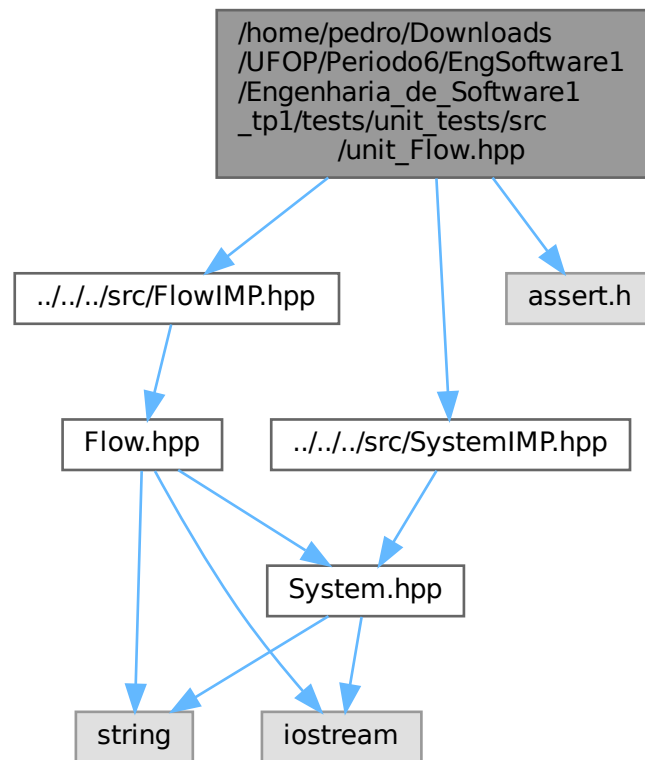
Here is the caller graph for this function:



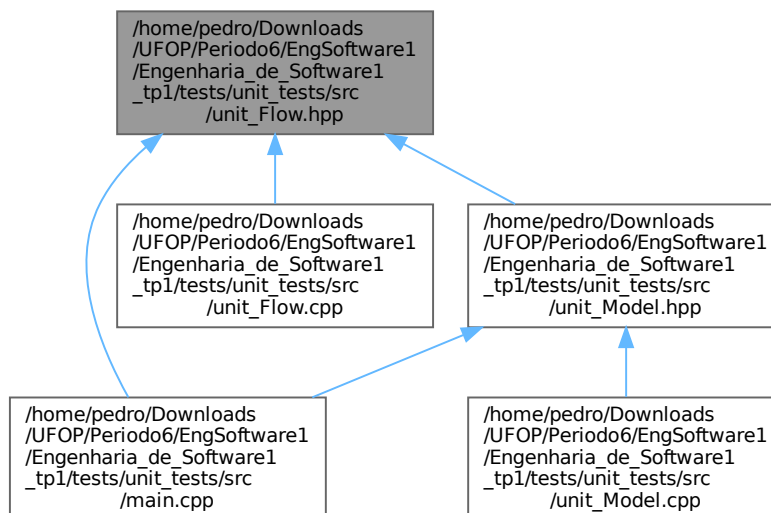
5.28 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_Flow.hpp File Reference

```
#include "../.../src/FlowIMP.hpp"  
#include "../.../src/SystemIMP.hpp"  
#include <assert.h>
```

Include dependency graph for unit_Flow.hpp:



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



Classes

- class [Flow_unit_test](#)

Functions

- void [test_Flow_constructor](#) ()
This function run the unit test of the flow constructor.
- void [test_Flow_destructor](#) ()
This function run the unit test of the flow destructor.
- void [test_Flow_getName](#) ()
This function run the unit test of the flow getName.
- void [test_Flow_getSource](#) ()
This function run the unit test of the flow getSource.
- void [test_Flow_getTarget](#) ()
This function run the unit test of the flow getTarge.
- void [test_Flow_setName](#) ()
This function run the unit test of the flow setName.
- void [test_Flow_setSource](#) ()
This function run the unit test of the flow setSource.
- void [test_Flow_setTarget](#) ()
This function run the unit test of the flow setTarge.
- void [test_Flow_execute](#) ()
This function run the unit test of the flow execute.
- void [test_Flow_equal](#) ()
This function run the unit test of the flow equal comparison.
- void [run_Flow_unit_test](#) ()
This function run the unit tests of the flow.

5.28.1 Function Documentation

5.28.1.1 run_Flow_unit_test()

```
void run_Flow_unit_test ( )
```

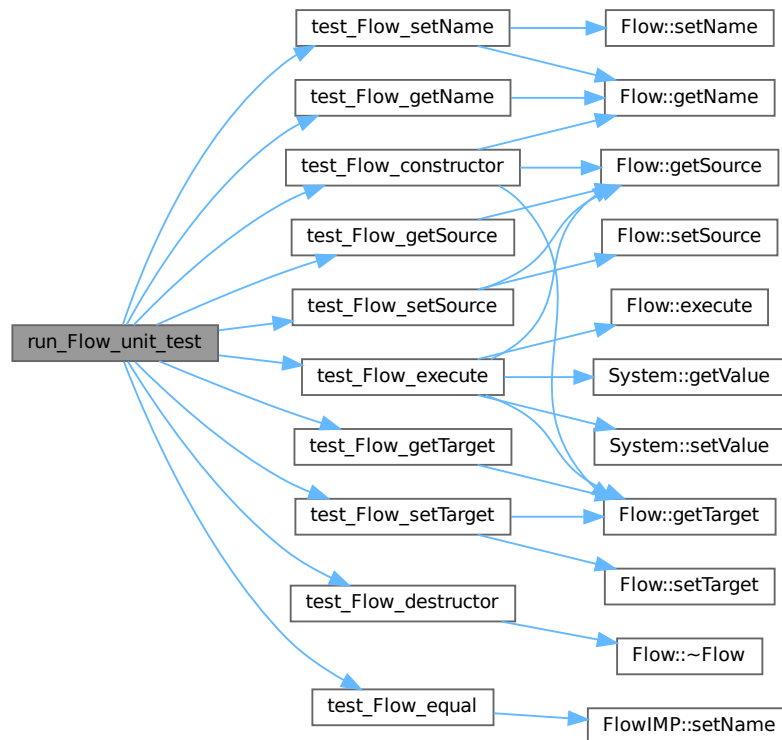
This function run the unit tests of the flow.

```
00189      {
00190          std::cout << "    Start Flow unit tests\n";
00191          test_Flow_constructor();
00192          test_Flow_destructor();
00193          test_Flow_getName();
00194          test_Flow_getSource();
00195          test_Flow_getTarget();
00196          test_Flow_setName();
00197          test_Flow_setSource();
00198          test_Flow_setTarget();
00199          test_Flow_execute();
00200          test_Flow_equal();
00201          std::cout << "    End Flow unit tests\n\n";
00202      }
```

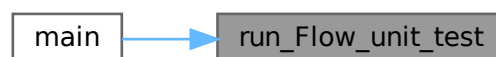
References [test_Flow_constructor\(\)](#), [test_Flow_destructor\(\)](#), [test_Flow_equal\(\)](#), [test_Flow_execute\(\)](#), [test_Flow_getName\(\)](#), [test_Flow_getSource\(\)](#), [test_Flow_getTarget\(\)](#), [test_Flow_setName\(\)](#), [test_Flow_setSource\(\)](#), and [test_Flow_setTarget\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.2 test_Flow_constructor()

```
void test_Flow_constructor ( )
```

This function run the unit test of the flow constructor.

```

00023         {
00024     std::cout << "          * Constructor tests\n";
00025     Flow* f1 = new Flow_unit_test();
00026     assert(f1->getName() == "NO_NAME");
00027     assert(f1->getSource() == NULL);
00028     assert(f1->getTarget() == NULL);
00029
00030     Flow* f2 = new Flow_unit_test("f2");
00031     assert(f2->getName() == "f2");
  
```

```

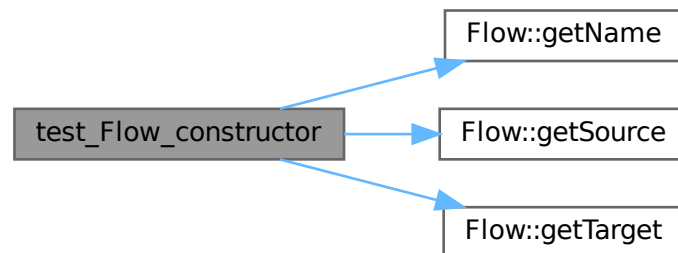
00032     assert(f2->getSource() == NULL);
00033     assert(f2->getTarget() == NULL);
00034
00035     System* s1 = new SystemIMP();
00036     Flow* f3 = new Flow_unit_test("f3", s1);
00037     assert(f3->getName() == "f3");
00038     assert(f3->getSource() == s1);
00039     assert(f3->getTarget() == NULL);
00040
00041     System* s2 = new SystemIMP();
00042     System* s3 = new SystemIMP();
00043     Flow* f4 = new Flow_unit_test("f4", s2, s3);
00044     assert(f4->getName() == "f4");
00045     assert(f4->getSource() == s2);
00046     assert(f4->getTarget() == s3);
00047
00048     delete f1;
00049     delete f2;
00050     delete s1;
00051     delete f3;
00052     delete s2;
00053     delete s3;
00054     delete f4;
00055 }

```

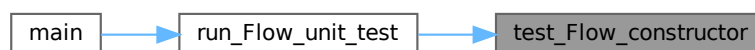
References [Flow::getName\(\)](#), [Flow::getSource\(\)](#), and [Flow::getTarget\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.3 test_Flow_destructor()

```

void test_Flow_destructor ( )

```

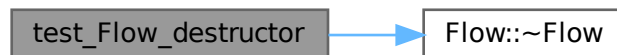
This function run the unit test of the flow destructor.

```
00057 {
00058     std::cout << "          * Destructor tests\n";
00059     Flow* f1 = new Flow_unit_test();
00060     f1->~Flow();
00061 }
```

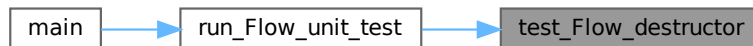
References [Flow::~~Flow\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.4 test_Flow_equal()

```
void test_Flow_equal ( )
```

This function run the unit test of the flow equal comparison.

```
00175 {
00176     std::cout << "          * Equal tests\n";
00177
00178     FlowIMP* f1 = new Flow_unit_test();
00179     FlowIMP* f2 = new Flow_unit_test();
00180     assert(*f1 == *f2);
00181
00182     f1->setName("f1");
00183     assert(*f1 != *f2);
00184
00185     delete f1;
00186     delete f2;
00187 }
```

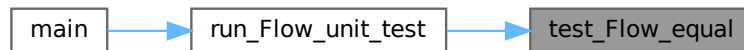
References [FlowIMP::setName\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.5 test_Flow_execute()

```
void test_Flow_execute ( )
```

This function run the unit test of the flow execute.

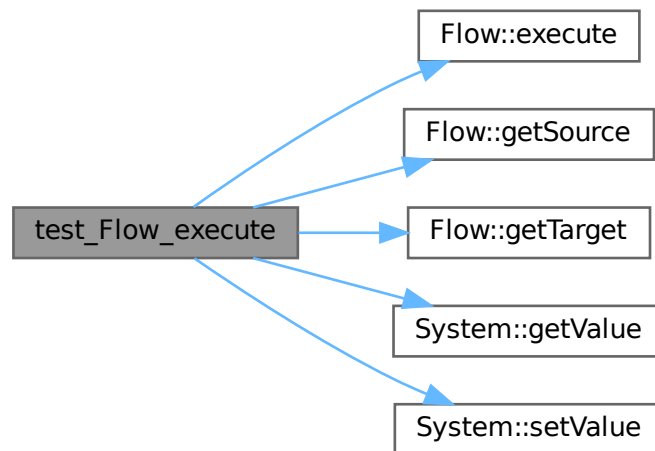
```

00157     {
00158         std::cout << "          * Execute tests\n";
00159
00160         System* s1 = new SystemIMP("s1", 100);
00161         System* s2 = new SystemIMP("s2", 0.0);
00162         Flow* f1 = new Flow_unit_test("f1", s1, s2);
00163         double result = f1->execute();
00164         f1->getTarget()->setValue(f1->getTarget()->getValue() + result);
00165         f1->getSource()->setValue(f1->getSource()->getValue() - result);
00166
00167         assert(s1->getValue() == 0);
00168         assert(s2->getValue() == 100);
00169
00170         delete s1;
00171         delete s2;
00172         delete f1;
00173     }
  
```

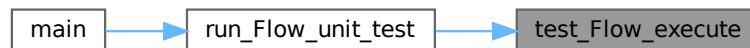
References [Flow::execute\(\)](#), [Flow::getSource\(\)](#), [Flow::getTarget\(\)](#), [System::getValue\(\)](#), and [System::setValue\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.6 test_Flow_getName()

```
void test_Flow_getName ( )
```

This function run the unit test of the flow getName.

```

00063     {
00064         std::cout << "          * getName tests\n";
00065         Flow* f1 = new Flow_unit_test();
00066         assert(f1->getName() == "NO_NAME");
00067
00068         Flow* f2 = new Flow_unit_test("f2");
00069         assert(f2->getName() == "f2");
00070
00071         delete f1;
00072         delete f2;
00073     }
  
```

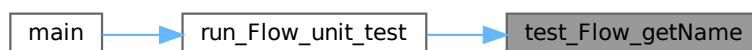
References [Flow::getName\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.7 test_Flow_getSource()

```
void test_Flow_getSource ( )
```

This function run the unit test of the flow getSource.

```
00075 {
00076     std::cout << "          * getSource tests\n";
00077     Flow* f1 = new Flow_unit_test();
00078     assert(f1->getSource() == NULL);
00079
00080     System* s1 = new SystemIMP();
00081     Flow* f2 = new Flow_unit_test("f2", s1);
00082     assert(f2->getSource() == s1);
00083
00084     delete f1;
00085     delete s1;
00086     delete f2;
00087 }
```

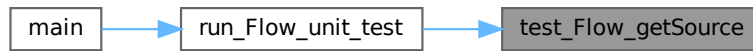
References [Flow::getSource\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.8 test_Flow_getTarget()

```
void test_Flow_getTarget ( )
```

This function run the unit test of the flow getTarge.

```

00089      {
00090          std::cout << "          * getTarget tests\n";
00091          Flow* f1 = new Flow_unit_test();
00092          assert(f1->getTarget() == NULL);
00093
00094          System* s1 = new SystemIMP();
00095          System* s2 = new SystemIMP();
00096          Flow* f2 = new Flow_unit_test("f2", s1, s2);
00097          assert(f2->getTarget() == s2);
00098
00099          delete f1;
00100          delete s1;
00101          delete s2;
00102          delete f2;
00103  }
```

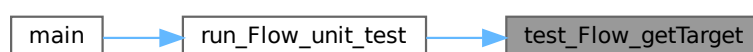
References [Flow::getTarget\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.9 test_Flow_setName()

```
void test_Flow_setName ( )
```

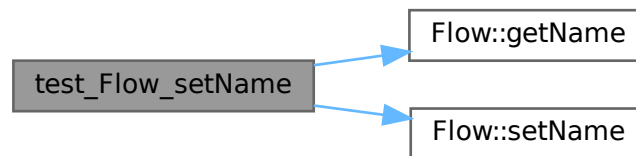
This function run the unit test of the flow setName.

```
00105         {
00106     std::cout << "          * setName tests\n";
00107     Flow* f1 = new Flow_unit_test();
00108     f1->setName("f1");
00109     assert(f1->getName() != "NO_NAME");
00110
00111     Flow* f2 = new Flow_unit_test("f");
00112     f2->setName("f2");
00113     assert(f2->getName() == "f2");
00114
00115     delete f1;
00116     delete f2;
00117 }
```

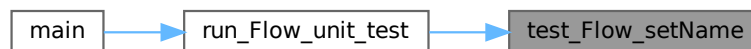
References [Flow::getName\(\)](#), and [Flow::setName\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.10 test_Flow_setSource()

```
void test_Flow_setSource ( )
```

This function run the unit test of the flow setSource.

```
00119         {
00120     std::cout << "          * setSource tests\n";
00121     System* s1 = new SystemIMP();
00122     Flow* f1 = new Flow_unit_test();
00123     f1->setSource(s1);
00124     assert(f1->getSource() != NULL);
00125 }
```

```

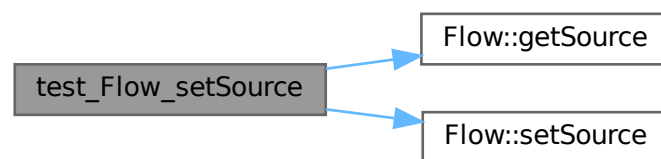
00126     System* s2 = new SystemIMP();
00127     Flow* f2 = new Flow_unit_test("f2", s1);
00128     f2->setSource(s2);
00129     assert(f2->getSource() == s2);
00130
00131     delete s1;
00132     delete s2;
00133     delete f1;
00134     delete f2;
00135 }

```

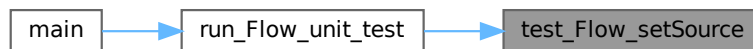
References [Flow::getSource\(\)](#), and [Flow::setSource\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.11 test_Flow_setTarget()

```
void test_Flow_setTarget ( )
```

This function run the unit test of the flow setTarge.

```

00137     {
00138         std::cout << "          * setTarget tests\n";
00139         System* s1 = new SystemIMP();
00140         Flow* f1 = new Flow_unit_test();
00141         f1->setTarget(s1);
00142         assert(f1->getTarget() != NULL);
00143
00144         System* s2 = new SystemIMP();
00145         System* s3 = new SystemIMP();
00146         Flow* f2 = new Flow_unit_test("f2", s1, s2);
00147         f2->setTarget(s3);
00148         assert(f2->getTarget() == s3);
00149
00150         delete s1;
00151         delete s2;
00152         delete s3;
00153         delete f1;

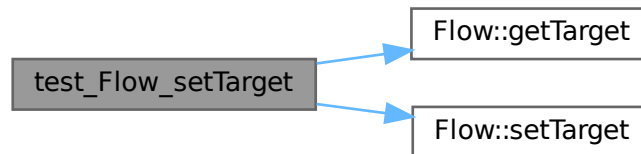
```

```
00154     delete f2;
00155 }
```

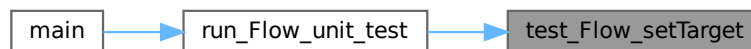
References [Flow::getTarget\(\)](#), and [Flow::setTarget\(\)](#).

Referenced by [run_Flow_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.29 unit_Flow.hpp

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * @file unit_Flow.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the flow units tests
00005  *****/
00006
00007 #ifndef UNIT_FLOW_HPP
00008 #define UNIT_FLOW_HPP
00009
00010 #include "../src/FlowIMP.hpp"
00011 #include "../src/SystemIMP.hpp"
00012
00013 #include <assert.h>
00014
00015
00016 * @brief This Flow class connects two systems and through the entered equation transfers values from
00017 one system to another
00018
00019 *****/
00018 class Flow_unit_test : public FlowIMP{
00019     private:
00024         Flow_unit_test(const Flow_unit_test& other);
00025
00026     public:
00027         //Constructor
00034         Flow_unit_test(const std::string& name = "NO_NAME", System* source = NULL, System* target =
            NULL);
```

```

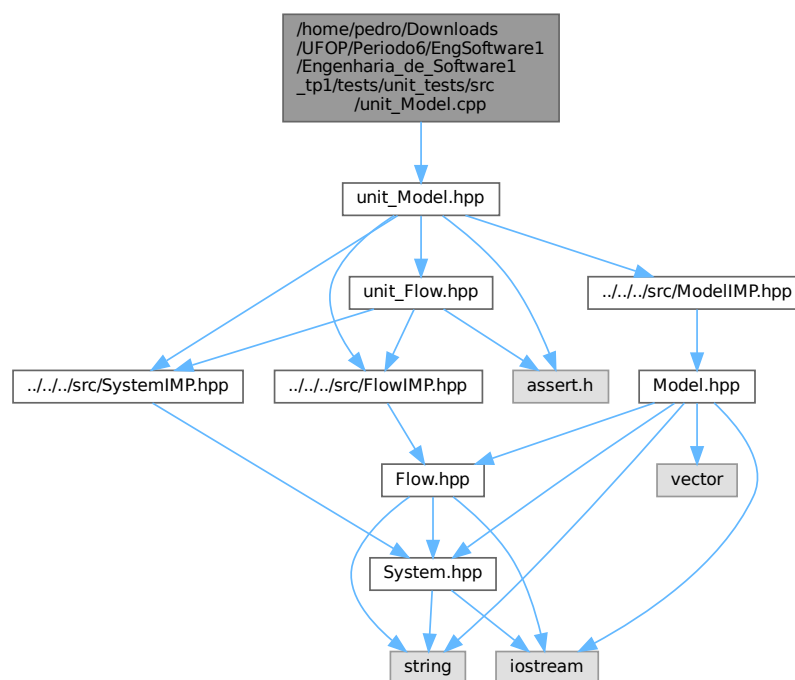
00035
00036     //Destructor
00040     virtual ~Flow_unit_test();
00041
00042     //Metodos
00047     virtual double execute() override;
00048 };
00049
00053 void test_Flow_constructor();
00057 void test_Flow_destructor();
00061 void test_Flow_getName();
00065 void test_Flow_getSource();
00069 void test_Flow_getTarget();
00073 void test_Flow_setName();
00077 void test_Flow_setSource();
00081 void test_Flow_setTarget();
00085 void test_Flow_execute();
00089 void test_Flow_equal();
00093 void run_Flow_unit_test();
00094
00095 #endif

```

5.30 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_Model.cpp File Reference

```
#include "unit_Model.hpp"
```

Include dependency graph for unit_Model.cpp:



Functions

- void `test_Model_constructor()`

This function run the unit test of the model constructor.

- void [test_Model_destructor](#) ()
This function run the unit test of the model destructor.
- void [test_Model_getName](#) ()
This function run the unit test of the model getName.
- void [test_Model_getSystems_and_add](#) ()
This function run the unit test of the model getSystems and add [System](#).
- void [test_Model_getFlows_and_add](#) ()
This function run the unit test of the model getFlows and add [Flow](#).
- void [test_Model_getStartTime](#) ()
This function run the unit test of the model getStartTime.
- void [test_Model_getEndTime](#) ()
This function run the unit test of the model getEndTime.
- void [test_Model_setName](#) ()
This function run the unit test of the model setName.
- void [test_Model_setSystems](#) ()
This function run the unit test of the model setSystems.
- void [test_Model_setFlows](#) ()
This function run the unit test of the model setFlows.
- void [test_Model_setStartTime](#) ()
This function run the unit test of the model setStartTime.
- void [test_Model_setEndTime](#) ()
This function run the unit test of the model setEndTime.
- void [test_Model_setTime](#) ()
This function run the unit test of the model setTime.
- void [test_Model_equal](#) ()
This function run the unit test of the model equal.
- void [test_Model_rmv_Sytem](#) ()
This function run the unit test of the model rmv [System](#).
- void [test_Model_rmv_Flow](#) ()
This function run the unit test of the model rmv [Flow](#).
- void [test_Model_run](#) ()
This function run the unit test of the model run.
- void [run_Model_unit_test](#) ()
This function run the unit tests of the model.

5.30.1 Function Documentation

5.30.1.1 run_Model_unit_test()

```
void run_Model_unit_test ( )
```

This function run the unit tests of the model.

```
00245 {  
00246     std::cout << "    Start Model unit tests\n";  
00247     test_Model_constructor();  
00248     test_Model_destructor();  
00249     test_Model_getName();  
00250     test_Model_getSystems_and_add();  
00251     test_Model_getFlows_and_add();  
00252     test_Model_getStartTime();  
00253     test_Model_getEndTime();  
00254     test_Model_setName();  
00255     test_Model_setSystems();  
00256     test_Model_setFlows();  
00257     test_Model_setStartTime();
```

```

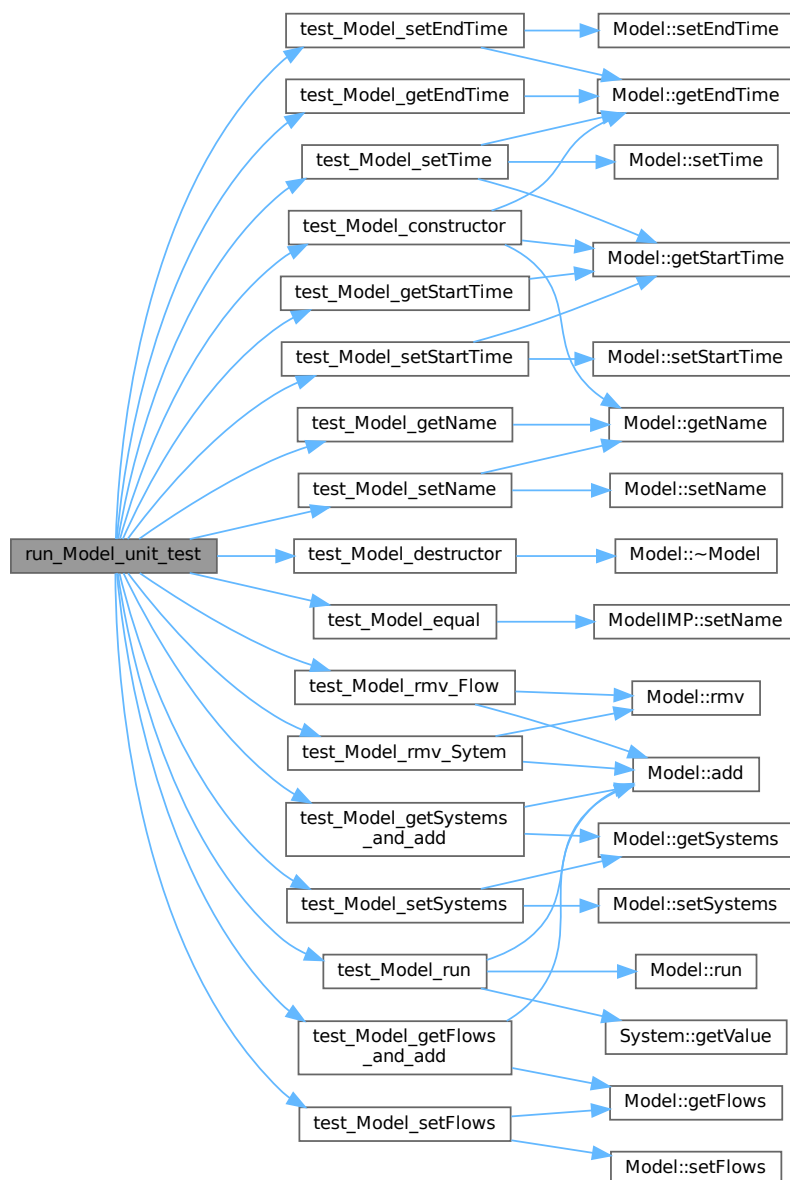
00258     test_Model_setEndTime();
00259     test_Model_setTime();
00260     test_Model_rmv_Sytem();
00261     test_Model_rmv_Flow();
00262     test_Model_run();
00263     test_Model_equal();
00264     std::cout << "      End Model unit tests\n";
00265 }

```

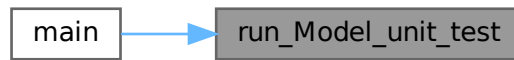
References [test_Model_constructor\(\)](#), [test_Model_destructor\(\)](#), [test_Model_equal\(\)](#), [test_Model_getEndTime\(\)](#), [test_Model_getFlows_and_add\(\)](#), [test_Model_getName\(\)](#), [test_Model_getStartTime\(\)](#), [test_Model_getSystems_and_add\(\)](#), [test_Model_rmv_Flow\(\)](#), [test_Model_rmv_Sytem\(\)](#), [test_Model_run\(\)](#), [test_Model_setEndTime\(\)](#), [test_Model_setFlows\(\)](#), [test_Model_setName\(\)](#), [test_Model_setStartTime\(\)](#), [test_Model_setSystems\(\)](#), and [test_Model_setTime\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.2 test_Model_constructor()

```
void test_Model_constructor ( )
```

This function run the unit test of the model constructor.

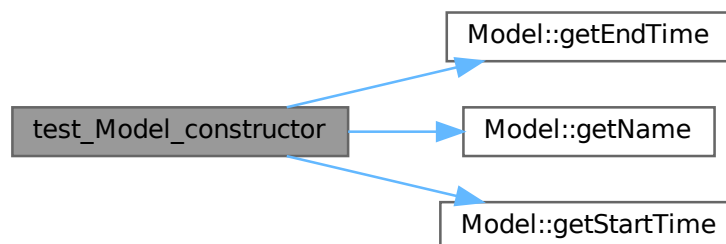
```

00003      {
00004          std::cout << "          * Constructor tests\n";
00005          Model* m1 = new ModelIMP();
00006          assert(m1->getName() == "NO_NAME");
00007          assert(m1->getStartTime() == 0);
00008          assert(m1->getEndTime() == 1);
00009
00010          Model* m2 = new ModelIMP("m2");
00011          assert(m2->getName() == "m2");
00012          assert(m2->getStartTime() == 0);
00013          assert(m2->getEndTime() == 1);
00014
00015          Model* m3 = new ModelIMP("m3", 2);
00016          assert(m3->getName() == "m3");
00017          assert(m3->getStartTime() == 2);
00018          assert(m3->getEndTime() == 1);
00019
00020          Model* m4 = new ModelIMP("m4", 2, 5);
00021          assert(m4->getName() == "m4");
00022          assert(m4->getStartTime() == 2);
00023          assert(m4->getEndTime() == 5);
00024
00025          delete m1;
00026          delete m2;
00027          delete m3;
00028          delete m4;
00029      }
  
```

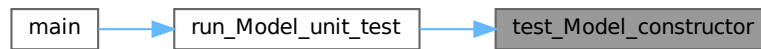
References [Model::getEndTime\(\)](#), [Model::getName\(\)](#), and [Model::getStartTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.3 test_Model_destructor()

```
void test_Model_destructor ( )
```

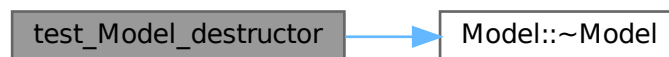
This function run the unit test of the model destructor.

```
00031      {
00032      std::cout << "          * Destructor tests\n";
00033      Model* m1 = new ModelIMP();
00034      m1->~Model();
00035 }
```

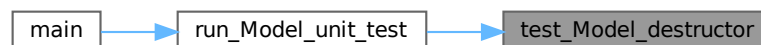
References [Model::~~Model\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.4 test_Model_equal()

```
void test_Model_equal ( )
```

This function run the unit test of the model equal.

```
00183     {
00184         std::cout << "          * Equal tests\n";
00185
00186         ModelIMP* m1 = new ModelIMP();
00187         ModelIMP* m2 = new ModelIMP();
00188         assert(*m1 == *m2);
00189
00190         m1->setName("m1");
00191         assert(m1 != m2);
00192
00193         delete m1;
00194         delete m2;
00195     }
```

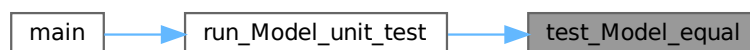
References [ModelIMP::setName\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.5 test_Model_getEndTime()

```
void test_Model_getEndTime ( )
```

This function run the unit test of the model getEndTime.

```
00087     {
00088         std::cout << "          * getEndTime tests\n";
00089         Model* m1 = new ModelIMP();
00090         assert(m1->getEndTime() == 1);
00091
00092         Model* m2 = new ModelIMP("m2", 0, 2);
00093         assert(m2->getEndTime() == 2);
00094
00095         delete m1;
00096         delete m2;
00097     }
```

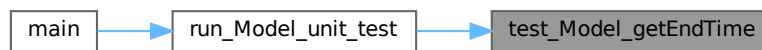
References [Model::getEndTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.6 test_Model_getFlows_and_add()

```
void test_Model_getFlows_and_add ( )
```

This function run the unit test of the model `getFlows` and add [Flow](#).

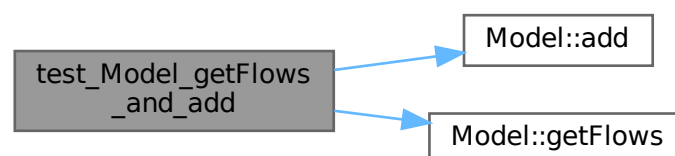
```

00062         {
00063     std::cout << "          * getFlows and add Flows tests\n";
00064     Flow* f1 = new Flow_unit_test("f1");
00065     Model* m1 = new ModelIMP("m1");
00066     std::vector<Flow*> flows;
00067     flows.push_back(f1);
00068     m1->add(f1);
00069     assert(m1->getFlows() == flows);
00070
00071     delete f1;
00072     delete m1;
00073 }
```

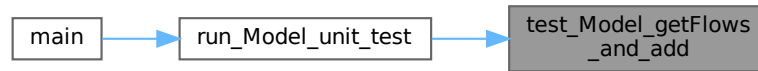
References [Model::add\(\)](#), and [Model::getFlows\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.7 test_Model_getName()

```
void test_Model_getName ( )
```

This function run the unit test of the model getName.

```

00037         {
00038     std::cout << "      * getName tests\n";
00039     Model* m1 = new ModelIMP();
00040     assert(m1->getName() == "NO_NAME");
00041
00042     Model* m2 = new ModelIMP("m2");
00043     assert(m2->getName() == "m2");
00044
00045     delete m1;
00046     delete m2;
00047 }

```

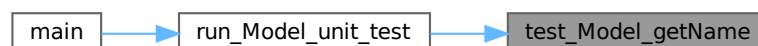
References [Model::getName\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.8 test_Model_getStartTime()

```
void test_Model_getStartTime ( )
```

This function run the unit test of the model getStartTime.

```
00075      {
00076      std::cout << "          * getStartTime tests\n";
00077      Model* m1 = new ModelIMP();
00078      assert(m1->getStartTime() == 0);
00079
00080      Model* m2 = new ModelIMP("m2", 1);
00081      assert(m2->getStartTime() == 1);
00082
00083      delete m1;
00084      delete m2;
00085  }
```

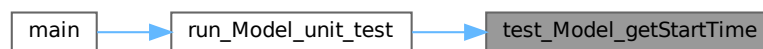
References [Model::getStartTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.9 test_Model_getSystems_and_add()

```
void test_Model_getSystems_and_add ( )
```

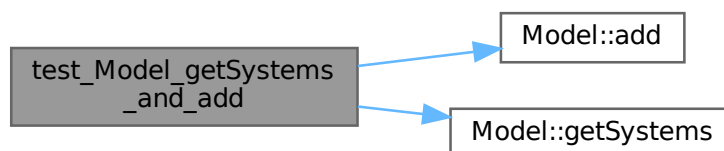
This function run the unit test of the model getSystems and add [System](#).

```
00049      {
00050      std::cout << "          * getSystems and add Systems tests\n";
00051      System* s1 = new SystemIMP("s1");
00052      Model* m1 = new ModelIMP("m1");
00053      std::vector<System*> systems;
00054      systems.push_back(s1);
00055      m1->add(s1);
00056      assert(m1->getSystems() == systems);
00057
00058      delete s1;
00059      delete m1;
00060  }
```

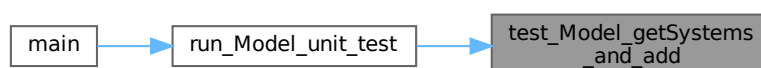
References [Model::add\(\)](#), and [Model::getSystems\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.10 test_Model_rmv_Flow()

```
void test_Model_rmv_Flow ( )
```

This function run the unit test of the model rmv [Flow](#).

```

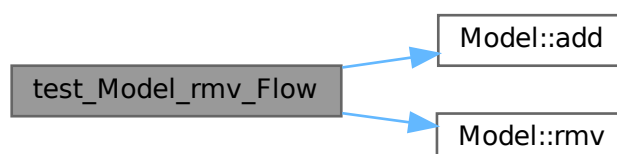
00210     {
00211         std::cout << "          * Remove Flow tests\n";
00212
00213         Model* model1 = new ModelIMP();
00214         FlowIMP* flow1 = new Flow_unit_test("flow1");
00215
00216         model1->add(flow1);
00217         assert(model1->rmv(flow1));
00218
00219         delete model1;
00220         delete flow1;
00221     }

```

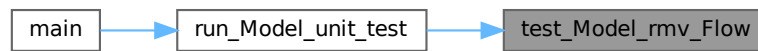
References [Model::add\(\)](#), and [Model::rmv\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.11 test_Model_rmv_Sytem()

```
void test_Model_rmv_Sytem ( )
```

This function run the unit test of the model rmv [System](#).

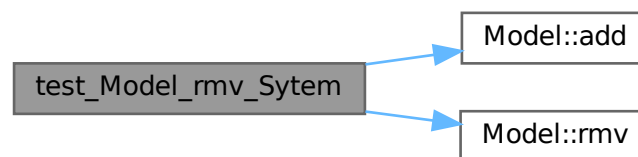
```

00197         {
00198     std::cout << "          * Remove System tests\n";
00199
00200     Model* modell = new ModelIMP();
00201     System* system1 = new SystemIMP("system1");
00202
00203     modell->add(system1);
00204     assert(modell->rmv(system1));
00205
00206     delete modell;
00207     delete system1;
00208 }
  
```

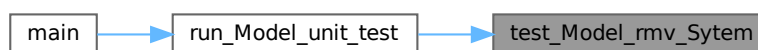
References [Model::add\(\)](#), and [Model::rmv\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.12 test_Model_run()

```
void test_Model_run ( )
```

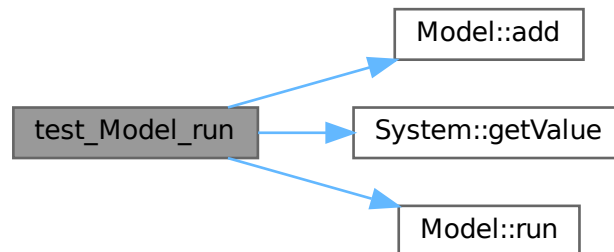
This function run the unit test of the model run.

```
00223     {
00224         std::cout << "          * Run tests\n";
00225
00226         System* s1 = new SystemIMP("s1", 100);
00227         System* s2 = new SystemIMP("s2", 0.0);
00228         Flow* f1 = new Flow_unit_test("f1", s1, s2);
00229         Model* m1 = new ModelIMP("m1", 0, 1);
00230
00231         m1->add(s1);
00232         m1->add(s2);
00233         m1->add(f1);
00234
00235         m1->run();
00236
00237         assert(s1->getValue() == 0);
00238         assert(s2->getValue() == 100);
00239
00240         delete s1;
00241         delete s2;
00242         delete f1;
00243     }
```

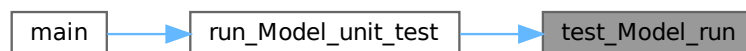
References [Model::add\(\)](#), [System::getValue\(\)](#), and [Model::run\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.13 test_Model_setEndTime()

```
void test_Model_setEndTime ( )
```

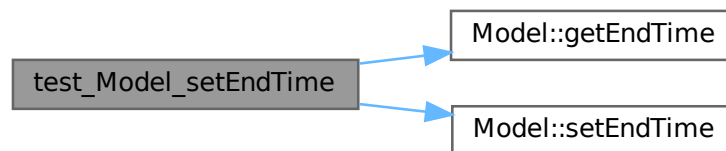
This function run the unit test of the model setEndTime.

```
00153     {
00154         std::cout << "          * setEndTime tests\n";
00155         Model* m1 = new ModelIMP();
00156         m1->setEndTime(3);
00157         assert(m1->getEndTime() != 1);
00158
00159         Model* m2 = new ModelIMP("m2", 0, 1);
00160         m2->setEndTime(2);
00161         assert(m2->getEndTime() == 2);
00162
00163         delete m1;
00164         delete m2;
00165     }
```

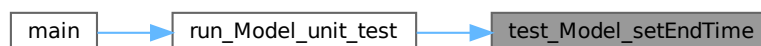
References [Model::getEndTime\(\)](#), and [Model::setEndTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.14 test_Model_setFlows()

```
void test_Model_setFlows ( )
```

This function run the unit test of the model setFlows.

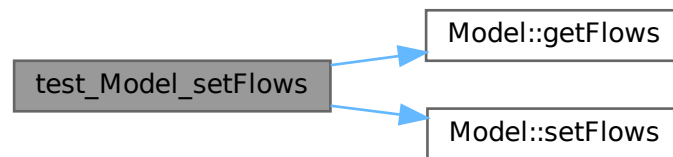
```
00126     {
00127         std::cout << "          * setFlows tests\n";
00128         Flow* f1 = new Flow_unit_test("f1");
00129         Model* m1 = new ModelIMP("m1");
00130         std::vector<Flow*> flows;
00131         flows.push_back(f1);
00132         m1->setFlows(flows);
00133         assert(m1->getFlows() == flows);
00134     }
```

```
00134
00135     delete f1;
00136     delete m1;
00137 }
```

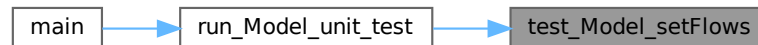
References [Model::getFlows\(\)](#), and [Model::setFlows\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.15 test_Model_setName()

```
void test_Model_setName ( )
```

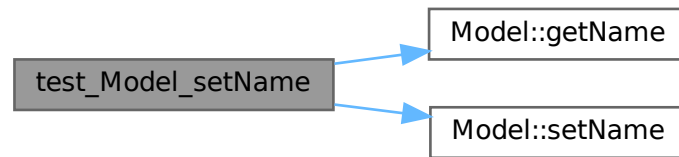
This function run the unit test of the model setName.

```
00099     {
00100         std::cout << "          * setName tests\n";
00101         Model* m1 = new ModelIMP();
00102         m1->setName("m1");
00103         assert(m1->getName() != "NO_NAME");
00104
00105         Model* m2 = new ModelIMP("m");
00106         m2->setName("m2");
00107         assert(m2->getName() == "m2");
00108
00109         delete m1;
00110         delete m2;
00111     }
```

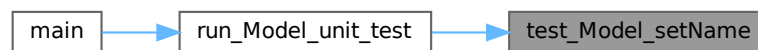
References [Model::getName\(\)](#), and [Model::setName\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.16 test_Model_setStartTime()

```
void test_Model_setStartTime ( )
```

This function run the unit test of the model `setStartTime`.

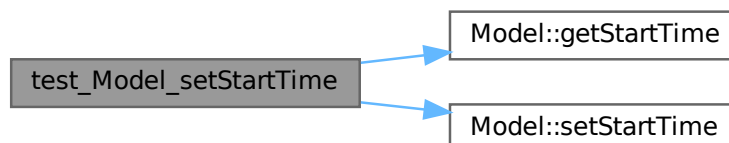
```

00139         {
00140     std::cout << "          * setStartTime tests\n";
00141     Model* m1 = new ModelIMP();
00142     m1->setStartTime(3);
00143     assert(m1->getStartTime() != 0);
00144
00145     Model* m2 = new ModelIMP("m2", 3);
00146     m2->setStartTime(1);
00147     assert(m2->getStartTime() == 1);
00148
00149     delete m1;
00150     delete m2;
00151 }
```

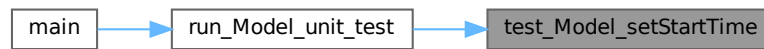
References [Model::getStartTime\(\)](#), and [Model::setStartTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.17 test_Model_setSystems()

```
void test_Model_setSystems ( )
```

This function run the unit test of the model setSystems.

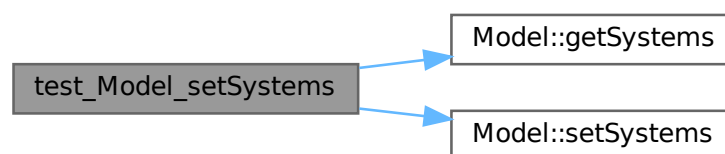
```

00113     {
00114     std::cout << "          * setSystems tests\n";
00115     System* s1 = new SystemIMP("s1");
00116     Model* m1 = new ModelIMP("m1");
00117     std::vector<System*> systems;
00118     systems.push_back(s1);
00119     m1->setSystems(systems);
00120     assert(m1->getSystems() == systems);
00121
00122     delete s1;
00123     delete m1;
00124 }
  
```

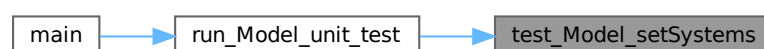
References [Model::getSystems\(\)](#), and [Model::setSystems\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.18 test_Model_setTime()

```
void test_Model_setTime ( )
```

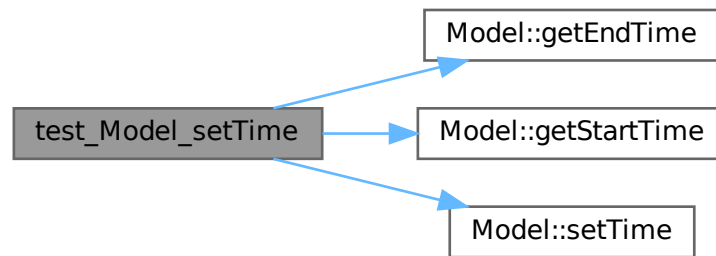
This function run the unit test of the model setTime.

```
00167     {
00168         std::cout << "          * setTime tests\n";
00169         Model* m1 = new ModelIMP();
00170         m1->setTime(1, 3);
00171         assert(m1->getStartTime() != 0);
00172         assert(m1->getEndTime() != 1);
00173
00174         Model* m2 = new ModelIMP("m2", 0, 1);
00175         m2->setTime(3, 4);
00176         assert(m2->getStartTime() == 3);
00177         assert(m2->getEndTime() == 4);
00178
00179         delete m1;
00180         delete m2;
00181     }
```

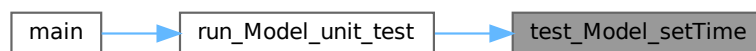
References [Model::getEndTime\(\)](#), [Model::getStartTime\(\)](#), and [Model::setTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

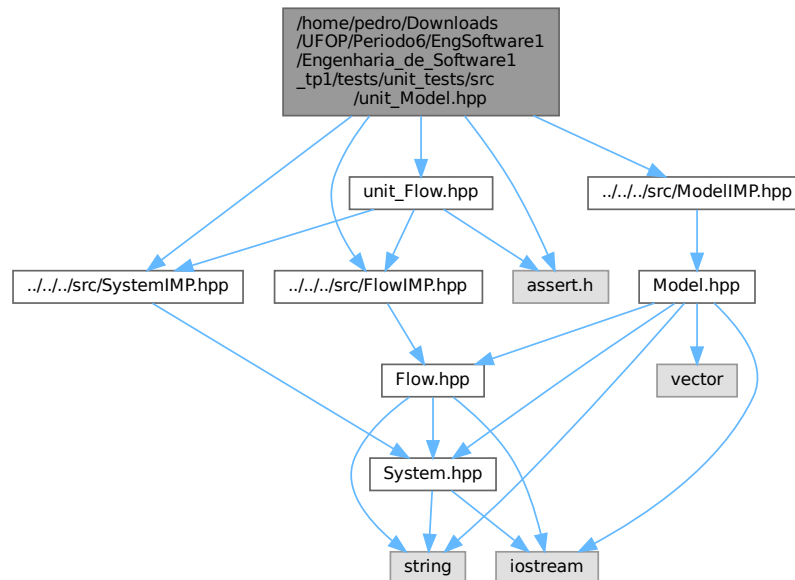


5.31 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_Model.hpp File Reference

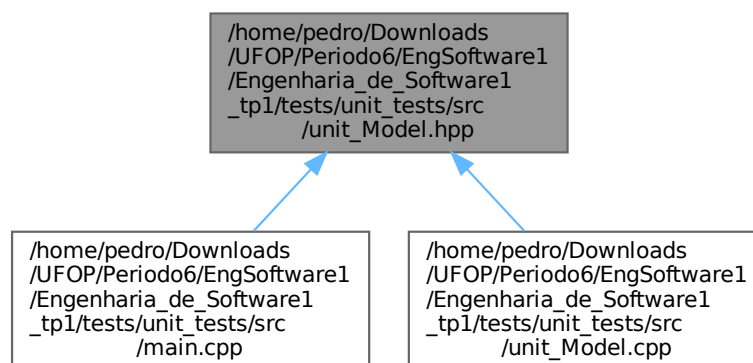
```
#include "../src/FlowIMP.hpp"
#include "../src/SystemIMP.hpp"
```

```
#include "../.../src/ModelIMP.hpp"  
#include "unit_Flow.hpp"  
#include <assert.h>
```

Include dependency graph for unit_Model.hpp:



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



Functions

- void `test_Model_constructor()`
This function run the unit test of the model constructor.
- void `test_Model_destructor()`
This function run the unit test of the model destructor.

- void [test_Model_getName](#) ()
This function run the unit test of the model getName.
- void [test_Model_getSystems_and_add](#) ()
This function run the unit test of the model getSystems and add [System](#).
- void [test_Model_getFlows_and_add](#) ()
This function run the unit test of the model getFlows and add [Flow](#).
- void [test_Model_getStartTime](#) ()
This function run the unit test of the model getStartTime.
- void [test_Model_getEndTime](#) ()
This function run the unit test of the model getEndTime.
- void [test_Model_setName](#) ()
This function run the unit test of the model setName.
- void [test_Model_setSystems](#) ()
This function run the unit test of the model setSystems.
- void [test_Model_setFlows](#) ()
This function run the unit test of the model setFlows.
- void [test_Model_setStartTime](#) ()
This function run the unit test of the model setStartTime.
- void [test_Model_setEndTime](#) ()
This function run the unit test of the model setEndTime.
- void [test_Model_setTime](#) ()
This function run the unit test of the model setTime.
- void [test_Model_rmv_Sytem](#) ()
This function run the unit test of the model rmv [System](#).
- void [test_Model_rmv_Flow](#) ()
This function run the unit test of the model rmv [Flow](#).
- void [test_Model_equal](#) ()
This function run the unit test of the model equal.
- void [test_Model_run](#) ()
This function run the unit test of the model run.
- void [run_Model_unit_test](#) ()
This function run the unit tests of the model.

5.31.1 Function Documentation

5.31.1.1 run_Model_unit_test()

```
void run_Model_unit_test ( )
```

This function run the unit tests of the model.

```
00245         {
00246         std::cout << "      Start Model unit tests\n";
00247         test_Model_constructor();
00248         test_Model_destructor();
00249         test_Model_getName();
00250         test_Model_getSystems_and_add();
00251         test_Model_getFlows_and_add();
00252         test_Model_getStartTime();
00253         test_Model_getEndTime();
00254         test_Model_setName();
00255         test_Model_setSystems();
00256         test_Model_setFlows();
00257         test_Model_setStartTime();
00258         test_Model_setEndTime();
00259         test_Model_setTime();
00260         test_Model_rmv_Sytem();
00261         test_Model_rmv_Flow();
```

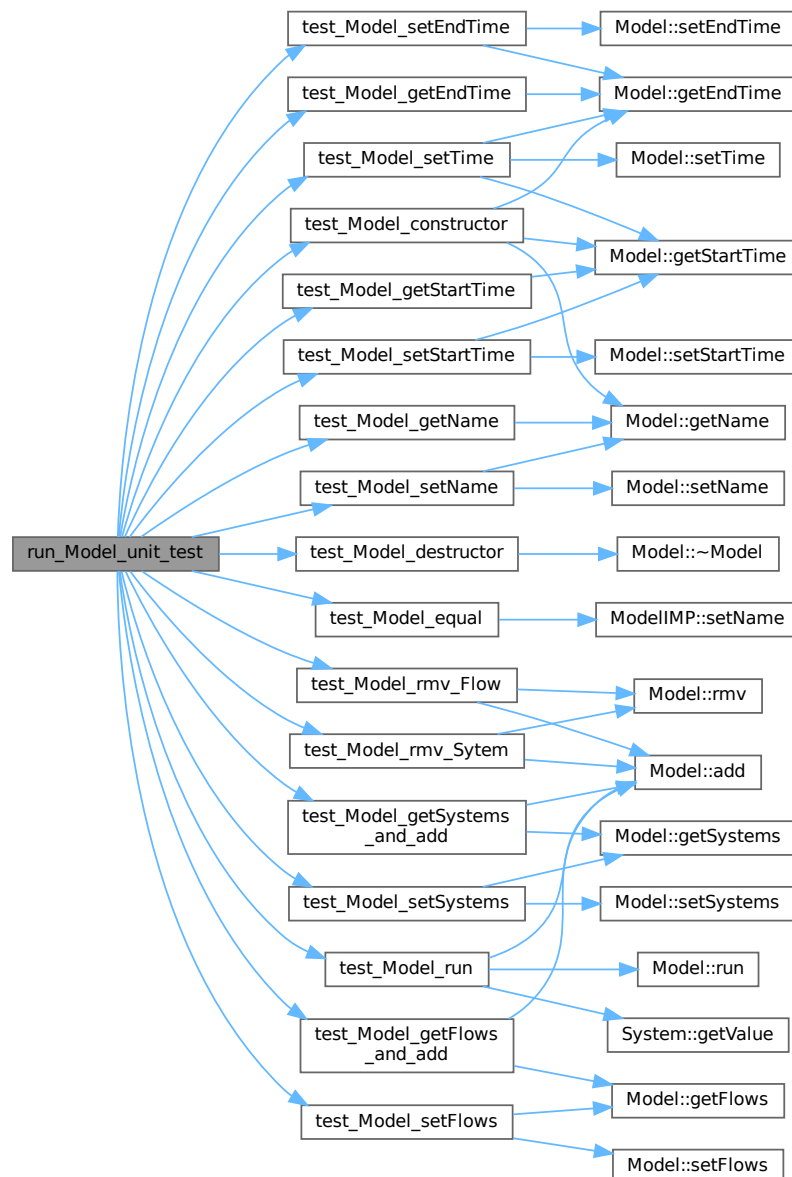


```
00262     test_Model_run();
00263     test_Model_equal();
00264     std::cout << "      End Model unit tests\n";
00265 }
```

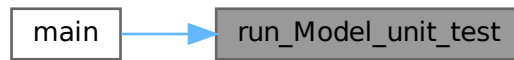
References [test_Model_constructor\(\)](#), [test_Model_destructor\(\)](#), [test_Model_equal\(\)](#), [test_Model_getEndTime\(\)](#), [test_Model_getFlows_and_add\(\)](#), [test_Model_getName\(\)](#), [test_Model_getStartTime\(\)](#), [test_Model_getSystems_and_add\(\)](#), [test_Model_rmv_Flow\(\)](#), [test_Model_rmv_Sytem\(\)](#), [test_Model_run\(\)](#), [test_Model_setEndTime\(\)](#), [test_Model_setFlows\(\)](#), [test_Model_setName\(\)](#), [test_Model_setStartTime\(\)](#), [test_Model_setSystems\(\)](#), and [test_Model_setTime\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.2 test_Model_constructor()

```
void test_Model_constructor ( )
```

This function run the unit test of the model constructor.

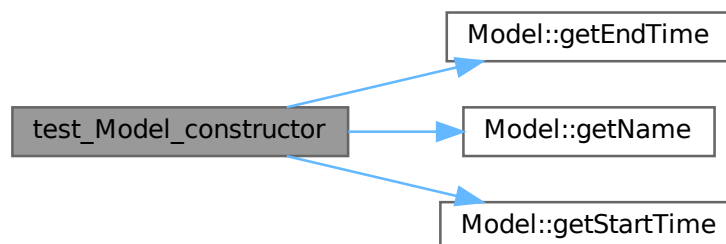
```

00003      {
00004          std::cout << "          * Constructor tests\n";
00005          Model* m1 = new ModelIMP();
00006          assert(m1->getName() == "NO_NAME");
00007          assert(m1->getStartTime() == 0);
00008          assert(m1->getEndTime() == 1);
00009
00010          Model* m2 = new ModelIMP("m2");
00011          assert(m2->getName() == "m2");
00012          assert(m2->getStartTime() == 0);
00013          assert(m2->getEndTime() == 1);
00014
00015          Model* m3 = new ModelIMP("m3", 2);
00016          assert(m3->getName() == "m3");
00017          assert(m3->getStartTime() == 2);
00018          assert(m3->getEndTime() == 1);
00019
00020          Model* m4 = new ModelIMP("m4", 2, 5);
00021          assert(m4->getName() == "m4");
00022          assert(m4->getStartTime() == 2);
00023          assert(m4->getEndTime() == 5);
00024
00025          delete m1;
00026          delete m2;
00027          delete m3;
00028          delete m4;
00029      }
  
```

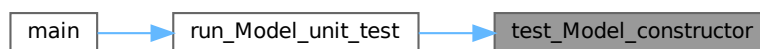
References [Model::getEndTime\(\)](#), [Model::getName\(\)](#), and [Model::getStartTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.3 test_Model_destructor()

```
void test_Model_destructor ( )
```

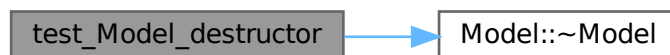
This function run the unit test of the model destructor.

```
00031     {  
00032     std::cout << "          * Destructor tests\n";  
00033     Model* m1 = new ModelIMP();  
00034     m1->~Model();  
00035 }
```

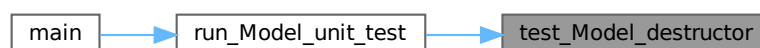
References [Model::~~Model\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.4 test_Model_equal()

```
void test_Model_equal ( )
```

This function run the unit test of the model equal.

```
00183         {
00184             std::cout << "          * Equal tests\n";
00185
00186             ModelIMP* m1 = new ModelIMP();
00187             ModelIMP* m2 = new ModelIMP();
00188             assert(*m1 == *m2);
00189
00190             m1->setName("m1");
00191             assert(m1 != m2);
00192
00193             delete m1;
00194             delete m2;
00195     }
```

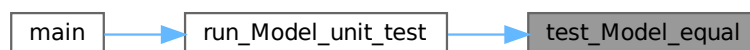
References [ModelIMP::setName\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.5 test_Model_getEndTime()

```
void test_Model_getEndTime ( )
```

This function run the unit test of the model getEndTime.

```
00087         {
00088             std::cout << "          * getEndTime tests\n";
00089             Model* m1 = new ModelIMP();
00090             assert(m1->getEndTime() == 1);
00091
00092             Model* m2 = new ModelIMP("m2", 0, 2);
00093             assert(m2->getEndTime() == 2);
00094
00095             delete m1;
00096             delete m2;
00097     }
```

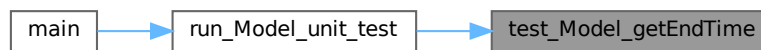
References [Model::getEndTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.6 test_Model_getFlows_and_add()

```
void test_Model_getFlows_and_add ( )
```

This function run the unit test of the model `getFlows` and add [Flow](#).

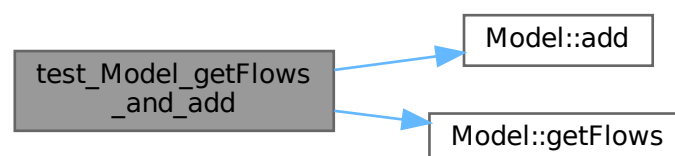
```

00062         {
00063     std::cout << "          * getFlows and add Flows tests\n";
00064     Flow* f1 = new Flow_unit_test("f1");
00065     Model* m1 = new ModelIMP("m1");
00066     std::vector<Flow*> flows;
00067     flows.push_back(f1);
00068     m1->add(f1);
00069     assert(m1->getFlows() == flows);
00070
00071     delete f1;
00072     delete m1;
00073 }
```

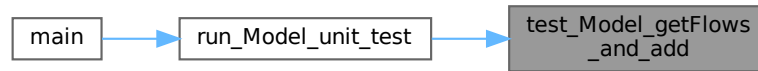
References [Model::add\(\)](#), and [Model::getFlows\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.7 test_Model_getName()

```
void test_Model_getName ( )
```

This function run the unit test of the model getName.

```

00037         {
00038     std::cout << "      * getName tests\n";
00039     Model* m1 = new ModelIMP ();
00040     assert (m1->getName () == "NO_NAME");
00041
00042     Model* m2 = new ModelIMP ("m2");
00043     assert (m2->getName () == "m2");
00044
00045     delete m1;
00046     delete m2;
00047 }
```

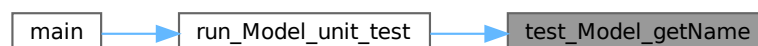
References [Model::getName\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.8 test_Model_getStartTime()

```
void test_Model_getStartTime ( )
```

This function run the unit test of the model getStartTime.

```
00075     {
00076     std::cout << "          * getStartTime tests\n";
00077     Model* m1 = new ModelIMP();
00078     assert(m1->getStartTime() == 0);
00079
00080     Model* m2 = new ModelIMP("m2", 1);
00081     assert(m2->getStartTime() == 1);
00082
00083     delete m1;
00084     delete m2;
00085 }
```

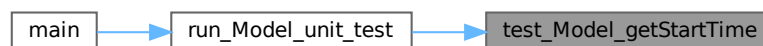
References [Model::getStartTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.9 test_Model_getSystems_and_add()

```
void test_Model_getSystems_and_add ( )
```

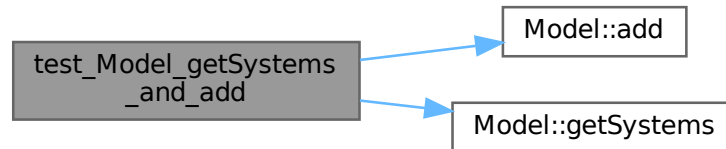
This function run the unit test of the model getSystems and add [System](#).

```
00049     {
00050     std::cout << "          * getSystems and add Systems tests\n";
00051     System* s1 = new SystemIMP("s1");
00052     Model* m1 = new ModelIMP("m1");
00053     std::vector<System*> systems;
00054     systems.push_back(s1);
00055     m1->add(s1);
00056     assert(m1->getSystems() == systems);
00057
00058     delete s1;
00059     delete m1;
00060 }
```

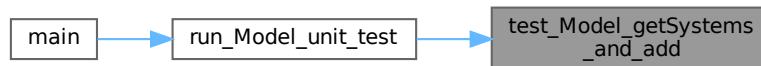
References [Model::add\(\)](#), and [Model::getSystems\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.10 test_Model_rmv_Flow()

```
void test_Model_rmv_Flow ( )
```

This function run the unit test of the model rmv [Flow](#).

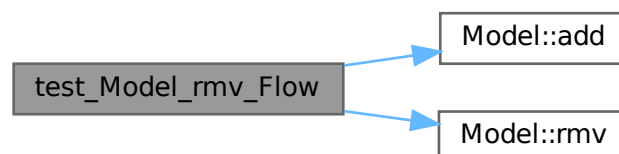
```

00210     {
00211         std::cout << "          * Remove Flow tests\n";
00212
00213         Model* model1 = new ModelIMP();
00214         FlowIMP* flow1 = new Flow_unit_test("flow1");
00215
00216         model1->add(flow1);
00217         assert(model1->rmv(flow1));
00218
00219         delete model1;
00220         delete flow1;
00221     }
  
```

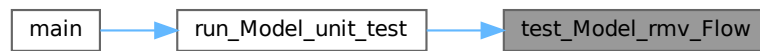
References [Model::add\(\)](#), and [Model::rmv\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.11 test_Model_rmv_Sytem()

```
void test_Model_rmv_Sytem ( )
```

This function run the unit test of the model rmv [System](#).

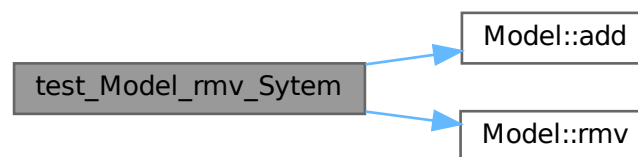
```

00197         {
00198     std::cout << "          * Remove System tests\n";
00199
00200     Model* model1 = new ModelIMP();
00201     System* system1 = new SystemIMP("system1");
00202
00203     model1->add(system1);
00204     assert(model1->rmv(system1));
00205
00206     delete model1;
00207     delete system1;
00208 }
    
```

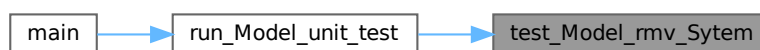
References [Model::add\(\)](#), and [Model::rmv\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.12 test_Model_run()

```
void test_Model_run ( )
```

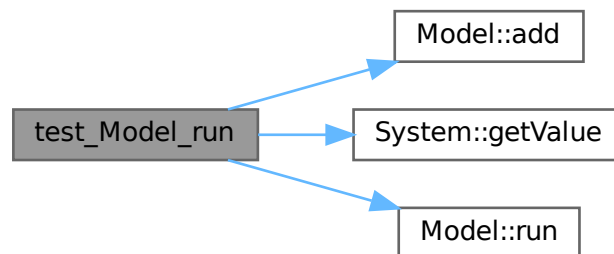
This function run the unit test of the model run.

```
00223     {
00224         std::cout << "          * Run tests\n";
00225
00226         System* s1 = new SystemIMP("s1", 100);
00227         System* s2 = new SystemIMP("s2", 0.0);
00228         Flow* f1 = new Flow_unit_test("f1", s1, s2);
00229         Model* m1 = new ModelIMP("m1", 0, 1);
00230
00231         m1->add(s1);
00232         m1->add(s2);
00233         m1->add(f1);
00234
00235         m1->run();
00236
00237         assert(s1->getValue() == 0);
00238         assert(s2->getValue() == 100);
00239
00240         delete s1;
00241         delete s2;
00242         delete f1;
00243     }
```

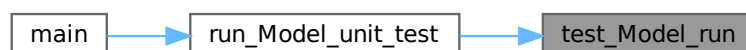
References [Model::add\(\)](#), [System::getValue\(\)](#), and [Model::run\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.13 test_Model_setEndTime()

```
void test_Model_setEndTime ( )
```

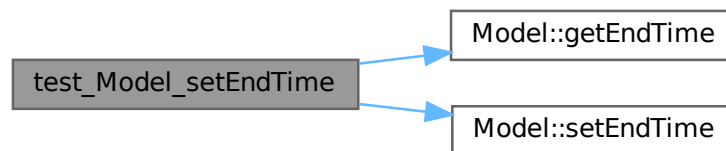
This function run the unit test of the model setEndTime.

```
00153     {
00154         std::cout << "          * setEndTime tests\n";
00155         Model* m1 = new ModelIMP();
00156         m1->setEndTime(3);
00157         assert(m1->getEndTime() != 1);
00158
00159         Model* m2 = new ModelIMP("m2", 0, 1);
00160         m2->setEndTime(2);
00161         assert(m2->getEndTime() == 2);
00162
00163         delete m1;
00164         delete m2;
00165     }
```

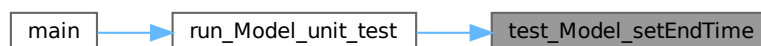
References [Model::getEndTime\(\)](#), and [Model::setEndTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.14 test_Model_setFlows()

```
void test_Model_setFlows ( )
```

This function run the unit test of the model setFlows.

```
00126     {
00127         std::cout << "          * setFlows tests\n";
00128         Flow* f1 = new Flow_unit_test("f1");
00129         Model* m1 = new ModelIMP("m1");
00130         std::vector<Flow*> flows;
00131         flows.push_back(f1);
00132         m1->setFlows(flows);
00133         assert(m1->getFlows() == flows);

```

```

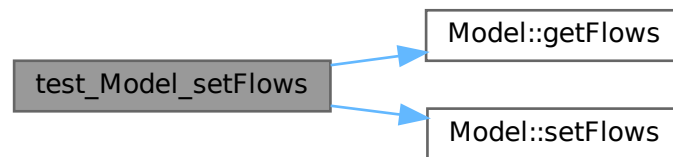
00134
00135     delete f1;
00136     delete m1;
00137 }

```

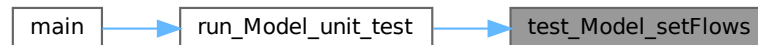
References [Model::getFlows\(\)](#), and [Model::setFlows\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.15 test_Model_setName()

```
void test_Model_setName ( )
```

This function run the unit test of the model setName.

```

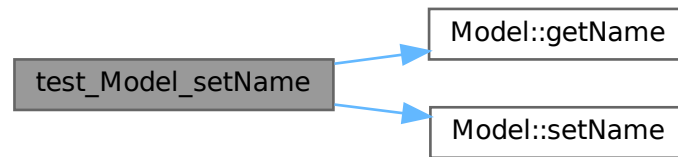
00099     {
00100         std::cout << "          * setName tests\n";
00101         Model* m1 = new ModelIMP();
00102         m1->setName("m1");
00103         assert(m1->getName() != "NO_NAME");
00104
00105         Model* m2 = new ModelIMP("m");
00106         m2->setName("m2");
00107         assert(m2->getName() == "m2");
00108
00109         delete m1;
00110         delete m2;
00111     }

```

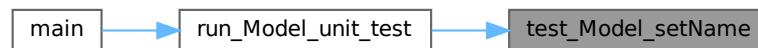
References [Model::getName\(\)](#), and [Model::setName\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.16 test_Model_setStartTime()

```
void test_Model_setStartTime ( )
```

This function run the unit test of the model `setStartTime`.

```

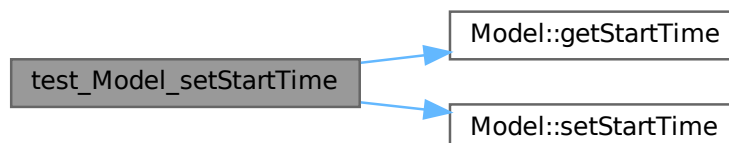
00139         {
00140         std::cout << "          * setStartTime tests\n";
00141         Model* m1 = new ModelIMP();
00142         m1->setStartTime(3);
00143         assert(m1->getStartTime() != 0);
00144
00145         Model* m2 = new ModelIMP("m2", 3);
00146         m2->setStartTime(1);
00147         assert(m2->getStartTime() == 1);
00148
00149         delete m1;
00150         delete m2;
00151     }

```

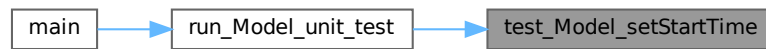
References [Model::getStartTime\(\)](#), and [Model::setStartTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.17 test_Model_setSystems()

```
void test_Model_setSystems ( )
```

This function run the unit test of the model setSystems.

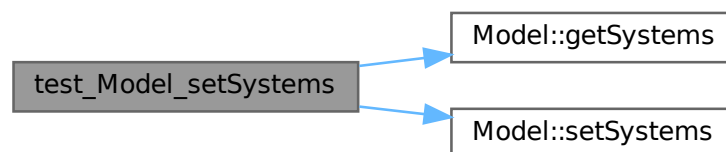
```

00113         {
00114     std::cout << "          * setSystems tests\n";
00115     System* s1 = new SystemIMP("s1");
00116     Model* m1 = new ModelIMP("m1");
00117     std::vector<System*> systems;
00118     systems.push_back(s1);
00119     m1->setSystems(systems);
00120     assert(m1->getSystems() == systems);
00121
00122     delete s1;
00123     delete m1;
00124 }
```

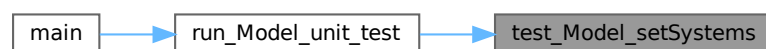
References [Model::getSystems\(\)](#), and [Model::setSystems\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.18 test_Model_setTime()

```
void test_Model_setTime ( )
```

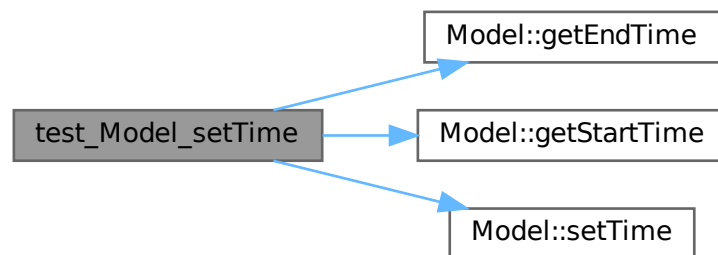
This function run the unit test of the model setTime.

```
00167         {
00168     std::cout << "          * setTime tests\n";
00169     Model* m1 = new ModelIMP();
00170     m1->setTime(1, 3);
00171     assert(m1->getStartTime() != 0);
00172     assert(m1->getEndTime() != 1);
00173
00174     Model* m2 = new ModelIMP("m2", 0, 1);
00175     m2->setTime(3, 4);
00176     assert(m2->getStartTime() == 3);
00177     assert(m2->getEndTime() == 4);
00178
00179     delete m1;
00180     delete m2;
00181 }
```

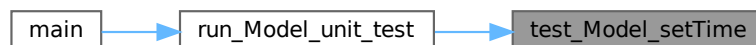
References [Model::getEndTime\(\)](#), [Model::getStartTime\(\)](#), and [Model::setTime\(\)](#).

Referenced by [run_Model_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.32 unit_Model.hpp

[Go to the documentation of this file.](#)

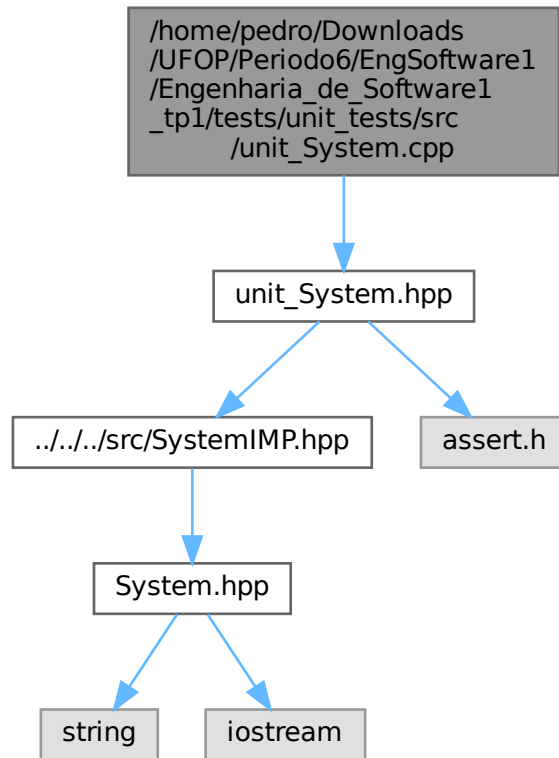
```
00001 /*****
00002  * @file unit_Model.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
```

```
00004  * @brief This file represents the model units tests
00005  *****/
00006
00007 #ifndef UNIT_MODEL_HPP
00008 #define UNIT_MODEL_HPP
00009
00010 #include "../src/FlowIMP.hpp"
00011 #include "../src/SystemIMP.hpp"
00012 #include "../src/ModelIMP.hpp"
00013 #include "unit_Flow.hpp"
00014
00015 #include <assert.h>
00016
00020 void test_Model_constructor();
00024 void test_Model_destructor();
00028 void test_Model_getName();
00032 void test_Model_getSystems_and_add();
00036 void test_Model_getFlows_and_add();
00040 void test_Model_getStartTime();
00044 void test_Model_getEndTime();
00048 void test_Model_setName();
00052 void test_Model_setSystems();
00056 void test_Model_setFlows();
00060 void test_Model_setStartTime();
00064 void test_Model_setEndTime();
00068 void test_Model_setTime();
00072 void test_Model_rmv_Sytem();
00076 void test_Model_rmv_Flow();
00080 void test_Model_equal();
00084 void test_Model_run();
00088 void run_Model_unit_test();
00089
00090
00091 #endif
```


5.33 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↵ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_↵ System.cpp File Reference

```
#include "unit_System.hpp"
```

Include dependency graph for unit_System.cpp:



Functions

- void `test_System_constructor()`
This function run the unit test of the system constructor.
- void `test_System_destructor()`
This function run the unit test of the system destructor.
- void `test_System_getName()`
This function run the unit test of the system getName.
- void `test_System_getValue()`
This function run the unit test of the system getValue.
- void `test_System_setName()`
This function run the unit test of the system setName.
- void `test_System_setValue()`
This function run the unit test of the system setValeu.

- void [test_System_equal\(\)](#)
This function run the unit test of the system equal comparison.
- void [run_System_unit_test\(\)](#)
This function run the unit tests of the system.

5.33.1 Function Documentation

5.33.1.1 run_System_unit_test()

```
void run_System_unit_test ( )
```

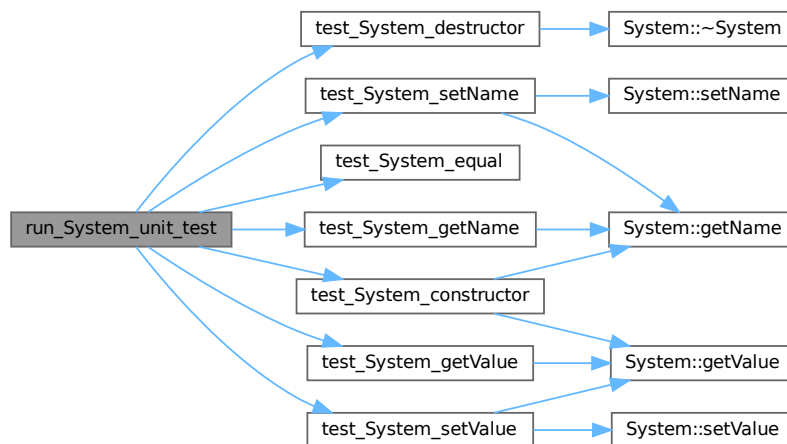
This function run the unit tests of the system.

```
00095         {
00096         std::cout << "    Start System unit tests\n";
00097         test_System_constructor();
00098         test_System_destructor();
00099         test_System_getName();
00100         test_System_getValue();
00101         test_System_setName();
00102         test_System_setValue();
00103         test_System_equal();
00104         std::cout << "    End System unit tests\n\n";
00105     }
```

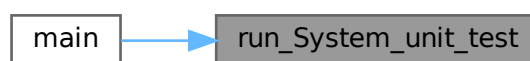
References [test_System_constructor\(\)](#), [test_System_destructor\(\)](#), [test_System_equal\(\)](#), [test_System_getName\(\)](#), [test_System_getValue\(\)](#), [test_System_setName\(\)](#), and [test_System_setValue\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.2 test_System_constructor()

```
void test_System_constructor ( )
```

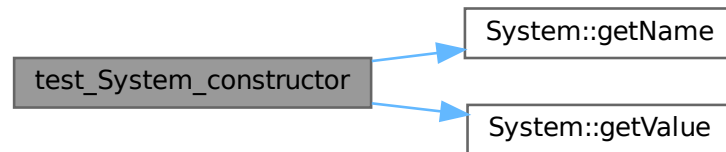
This function run the unit test of the system constructor.

```
00003      {
00004      std::cout << "          * Constructor tests\n";
00005      System* s1 = new SystemIMP();
00006      assert(s1->getName() == "NO_NAME");
00007      assert(s1->getValue() == 0.0);
00008
00009      System* s2 = new SystemIMP("s2");
00010      assert(s2->getName() == "s2");
00011      assert(s2->getValue() == 0.0);
00012
00013      System* s3 = new SystemIMP("s3", 2.0);
00014      assert(s3->getName() == "s3");
00015      assert(s3->getValue() == 2.0);
00016
00017      delete s1;
00018      delete s2;
00019      delete s3;
00020 }
```

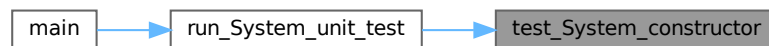
References [System::getName\(\)](#), and [System::getValue\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.3 test_System_destructor()

```
void test_System_destructor ( )
```

This function run the unit test of the system destructor.

```
00022      {
00023      std::cout << "          * Destructor tests\n";
00024      System* s1 = new SystemIMP();
```

```
00025     s1->~System();
00026 }
```

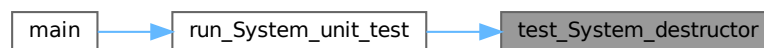
References [System::~~System\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.4 test_System_equal()

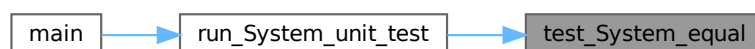
```
void test_System_equal ( )
```

This function run the unit test of the system equal comparison.

```
00079     {
00080         std::cout << "          * Equal tests\n";
00081         SystemIMP* s1 = new SystemIMP ("s1");
00082         SystemIMP* s2 = new SystemIMP ("s2");
00083         assert (*s1 != *s2);
00084
00085         SystemIMP* s3 = new SystemIMP ();
00086         SystemIMP* s4 = new SystemIMP ();
00087         assert (*s3 == *s4);
00088
00089         delete s1;
00090         delete s2;
00091         delete s3;
00092         delete s4;
00093     }
```

Referenced by [run_System_unit_test\(\)](#).

Here is the caller graph for this function:



5.33.1.5 test_System_getName()

```
void test_System_getName ( )
```

This function run the unit test of the system getName.

```
00028      {
00029      std::cout << "      * getName tests\n";
00030      System* s1 = new SystemIMP();
00031      assert(s1->getName() == "NO_NAME");
00032
00033      System* s2 = new SystemIMP("s2");
00034      assert(s2->getName() == "s2");
00035
00036      delete s1;
00037      delete s2;
00038  }
```

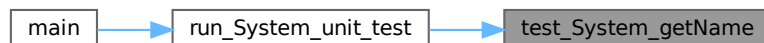
References [System::getName\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.6 test_System_getValue()

```
void test_System_getValue ( )
```

This function run the unit test of the system getValue.

```
00040      {
00041      std::cout << "      * getValue tests\n";
00042      System* s1 = new SystemIMP();
00043      assert(s1->getValue() == 0);
00044
00045      System* s2 = new SystemIMP("s2", 22);
00046      assert(s2->getValue() == 22);
00047
00048      delete s1;
00049      delete s2;
00050  }
```

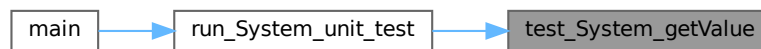
References [System::getValue\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.7 test_System_setName()

```
void test_System_setName ( )
```

This function run the unit test of the system setName.

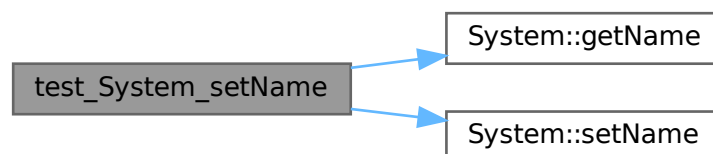
```

00052     {
00053         std::cout << "          * setName tests\n";
00054         System* s1 = new SystemIMP();
00055         s1->setName("s1");
00056         assert(s1->getName() != "NO_NAME");
00057
00058         System* s2 = new SystemIMP();
00059         s2->setName("s2");
00060         assert(s2->getName() == "s2");
00061         delete s1;
00062         delete s2;
00063     }
  
```

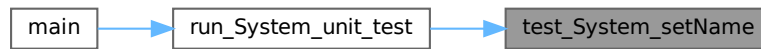
References [System::getName\(\)](#), and [System::setName\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.8 test_System_setValue()

```
void test_System_setValue ( )
```

This function run the unit test of the system setValeu.

```

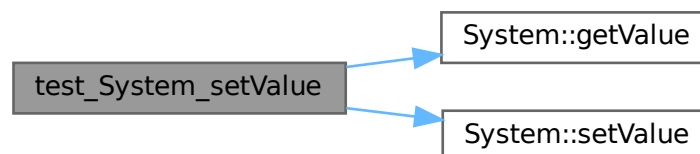
00065         {
00066         std::cout << "          * setValue tests\n";
00067         System* s1 = new SystemIMP();
00068         s1->setValue(21);
00069         assert(s1->getValue() != 0);
00070
00071         System* s2 = new SystemIMP("s2", 22);
00072         s2->setValue(45);
00073         assert(s2->getValue() == 45);
00074
00075         delete s1;
00076         delete s2;
00077     }

```

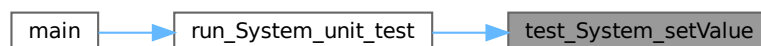
References [System::getValue\(\)](#), and [System::setValue\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:

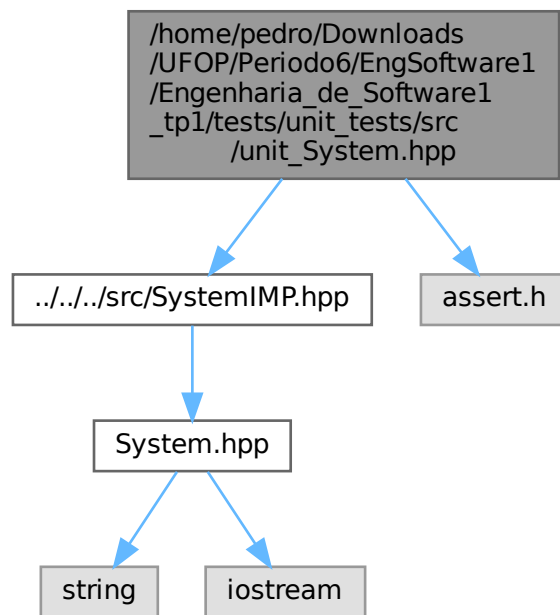


Here is the caller graph for this function:

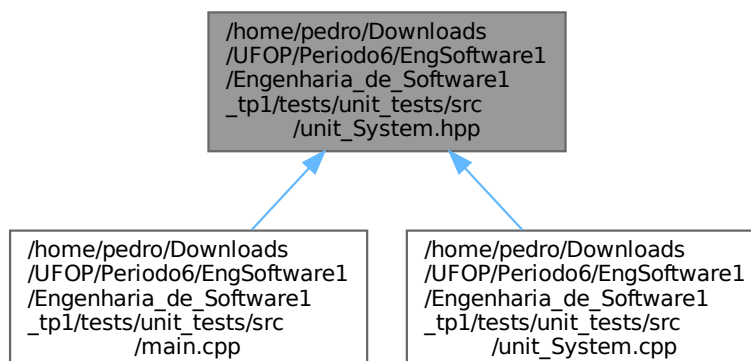


5.34 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_System.hpp File Reference

```
#include "../../src/SystemIMP.hpp"
#include <assert.h>
Include dependency graph for unit_System.hpp:
```



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



Functions

- void [test_System_constructor](#) ()
This function run the unit test of the system constructor.
- void [test_System_destructor](#) ()
This function run the unit test of the system destructor.
- void [test_System_getName](#) ()
This function run the unit test of the system getName.
- void [test_System_getValue](#) ()
This function run the unit test of the system getValue.
- void [test_System_setName](#) ()
This function run the unit test of the system setName.
- void [test_System_setValue](#) ()
This function run the unit test of the system setValeu.
- void [test_System_equal](#) ()
This function run the unit test of the system equal comparison.
- void [run_System_unit_test](#) ()
This function run the unit tests of the system.

5.34.1 Function Documentation

5.34.1.1 run_System_unit_test()

```
void run_System_unit_test ( )
```

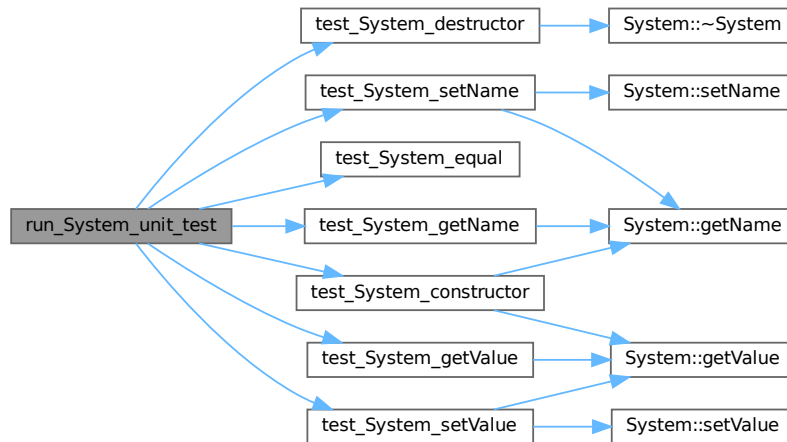
This function run the unit tests of the system.

```
00095      {  
00096          std::cout << "      Start System unit tests\n";  
00097          test\_System\_constructor();  
00098          test\_System\_destructor();  
00099          test\_System\_getName();  
00100          test\_System\_getValue();  
00101          test\_System\_setName();  
00102          test\_System\_setValue();  
00103          test\_System\_equal();  
00104          std::cout << "      End System unit tests\n\n";  
00105      }
```

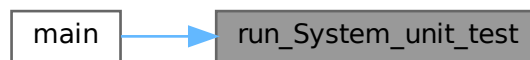
References [test_System_constructor\(\)](#), [test_System_destructor\(\)](#), [test_System_equal\(\)](#), [test_System_getName\(\)](#), [test_System_getValue\(\)](#), [test_System_setName\(\)](#), and [test_System_setValue\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.1.2 test_System_constructor()

```
void test_System_constructor ( )
```

This function run the unit test of the system constructor.

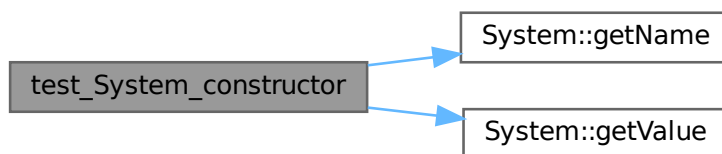
```

00003      {
00004      std::cout << "          * Constructor tests\n";
00005      System* s1 = new SystemIMP();
00006      assert(s1->getName() == "NO_NAME");
00007      assert(s1->getValue() == 0.0);
00008
00009      System* s2 = new SystemIMP("s2");
00010      assert(s2->getName() == "s2");
00011      assert(s2->getValue() == 0.0);
00012
00013      System* s3 = new SystemIMP("s3", 2.0);
00014      assert(s3->getName() == "s3");
00015      assert(s3->getValue() == 2.0);
00016
00017      delete s1;
00018      delete s2;
00019      delete s3;
00020  }
```

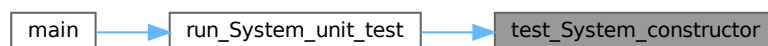
References [System::getName\(\)](#), and [System::getValue\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.1.3 test_System_destructor()

```
void test_System_destructor ( )
```

This function run the unit test of the system destructor.

```

00022     {
00023     std::cout << "          * Destructor tests\n";
00024     System* s1 = new SystemIMP();
00025     s1->~System();
00026 }
```

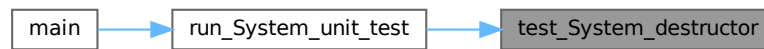
References [System::~~System\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.1.4 test_System_equal()

```
void test_System_equal ( )
```

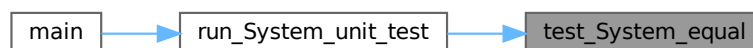
This function run the unit test of the system equal comparison.

```

00079         {
00080         std::cout << "          * Equal tests\n";
00081         SystemIMP* s1 = new SystemIMP("s1");
00082         SystemIMP* s2 = new SystemIMP("s2");
00083         assert(*s1 != *s2);
00084
00085         SystemIMP* s3 = new SystemIMP();
00086         SystemIMP* s4 = new SystemIMP();
00087         assert(*s3 == *s4);
00088
00089         delete s1;
00090         delete s2;
00091         delete s3;
00092         delete s4;
00093     }
  
```

Referenced by [run_System_unit_test\(\)](#).

Here is the caller graph for this function:



5.34.1.5 test_System_getName()

```
void test_System_getName ( )
```

This function run the unit test of the system getName.

```

00028         {
00029         std::cout << "          * getName tests\n";
00030         System* s1 = new SystemIMP();
00031         assert(s1->getName() == "NO_NAME");
00032
00033         System* s2 = new SystemIMP("s2");
00034         assert(s2->getName() == "s2");
00035
00036         delete s1;
00037         delete s2;
00038     }
  
```

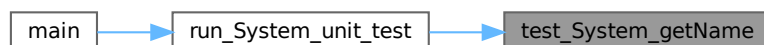
References [System::getName\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.1.6 test_System_getValue()

```
void test_System_getValue ( )
```

This function run the unit test of the system `getValue`.

```

00040      {
00041          std::cout << "          * getValue tests\n";
00042          System* s1 = new SystemIMP();
00043          assert(s1->getValue() == 0);
00044
00045          System* s2 = new SystemIMP("s2", 22);
00046          assert(s2->getValue() == 22);
00047
00048          delete s1;
00049          delete s2;
00050      }

```

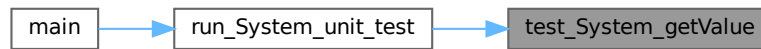
References [System::getValue\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.1.7 test_System_setName()

```
void test_System_setName ( )
```

This function run the unit test of the system setName.

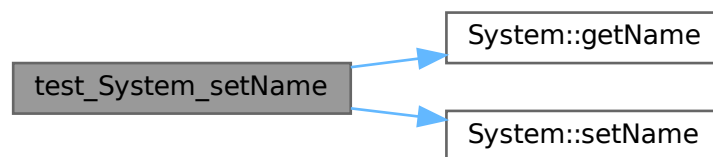
```

00052         {
00053     std::cout << "          * setName tests\n";
00054     System* s1 = new SystemIMP();
00055     s1->setName("s1");
00056     assert(s1->getName() != "NO_NAME");
00057
00058     System* s2 = new SystemIMP();
00059     s2->setName("s2");
00060     assert(s2->getName() == "s2");
00061     delete s1;
00062     delete s2;
00063 }
```

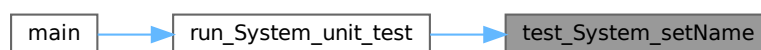
References [System::getName\(\)](#), and [System::setName\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.1.8 test_System_setValue()

```
void test_System_setValue ( )
```

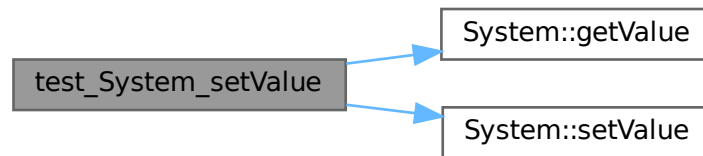
This function run the unit test of the system setValeu.

```
00065     {
00066         std::cout << "          * setValue tests\n";
00067         System* s1 = new SystemIMP();
00068         s1->setValue(21);
00069         assert(s1->getValue() != 0);
00070
00071         System* s2 = new SystemIMP("s2", 22);
00072         s2->setValue(45);
00073         assert(s2->getValue() == 45);
00074
00075         delete s1;
00076         delete s2;
00077     }
```

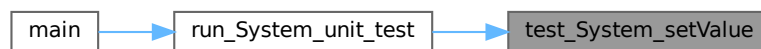
References [System::getValue\(\)](#), and [System::setValue\(\)](#).

Referenced by [run_System_unit_test\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.35 unit_System.hpp

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * @file unit_System.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the system units tests
00005  *****/
00006
00007 #ifndef UNIT_SYSTEM_HPP
00008 #define UNIT_SYSTEM_HPP
00009
00010 #include "../src/SystemIMP.hpp"
```

```
00011
00012 #include <assert.h>
00013
00017 void test_System_constructor();
00021 void test_System_destructor();
00025 void test_System_getName();
00029 void test_System_getValue();
00033 void test_System_setName();
00037 void test_System_setValue();
00041 void test_System_equal();
00045 void run_System_unit_test();
00046
00047 #endif
```