My Vensin

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

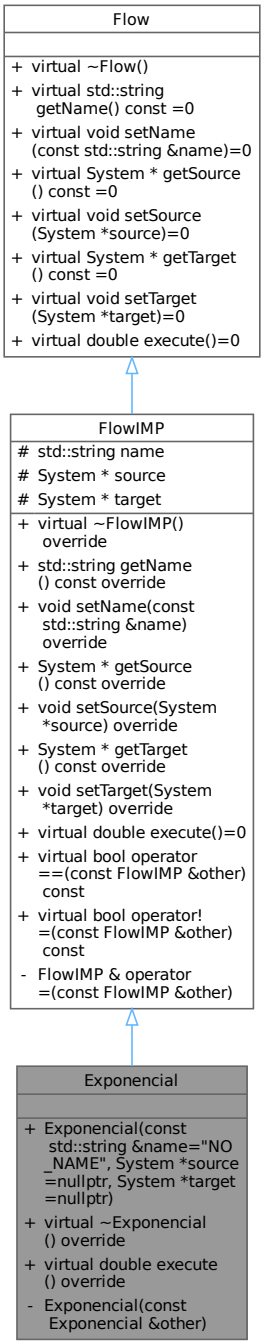
Chapter 4

Class Documentation

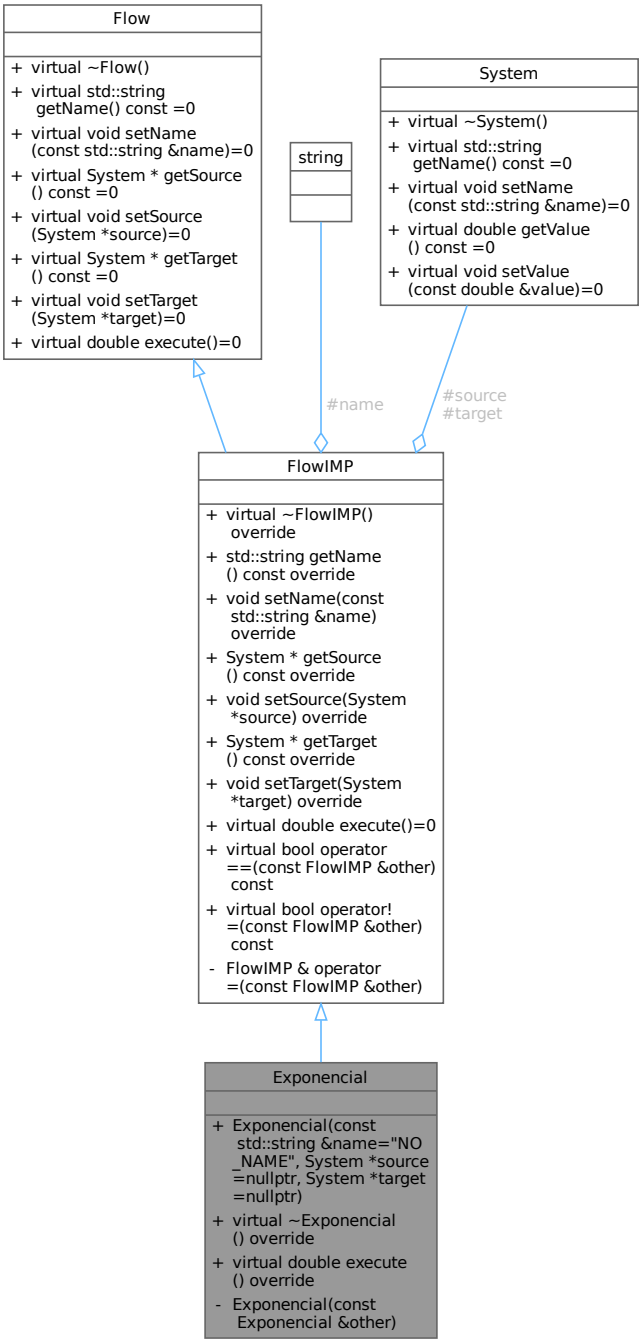
4.1 Exponencial Class Reference

`#include <Exponencial.hpp>`

Inheritance diagram for Exponencial:



Collaboration diagram for Exponencial:



Public Member Functions

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

  Construct a new Exponencial by name, source and target.
- virtual ~Exponencial () override

  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 () override

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

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

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

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

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

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

  This method assigns a system poiter 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

other	Exponencial 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 Exponencial() [2/2]

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

Construct a new Exponencial by name, source and target.

Parameters

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

```
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 ~Exponencial()

```
Exponencial::~Exponencial ( )  [override], [virtual]
```

This destructor is a virtual destructor of the Class.

```
00018 {}
```

4.1.2 Member Function Documentation

4.1.2.1 execute()

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

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

Returns

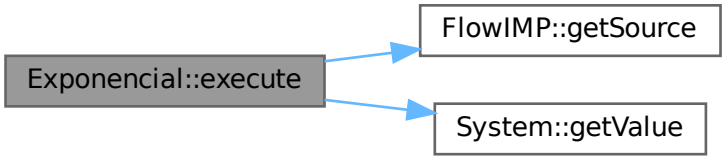
double

Implements FlowIMP.

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

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

Here is the call graph for this function:



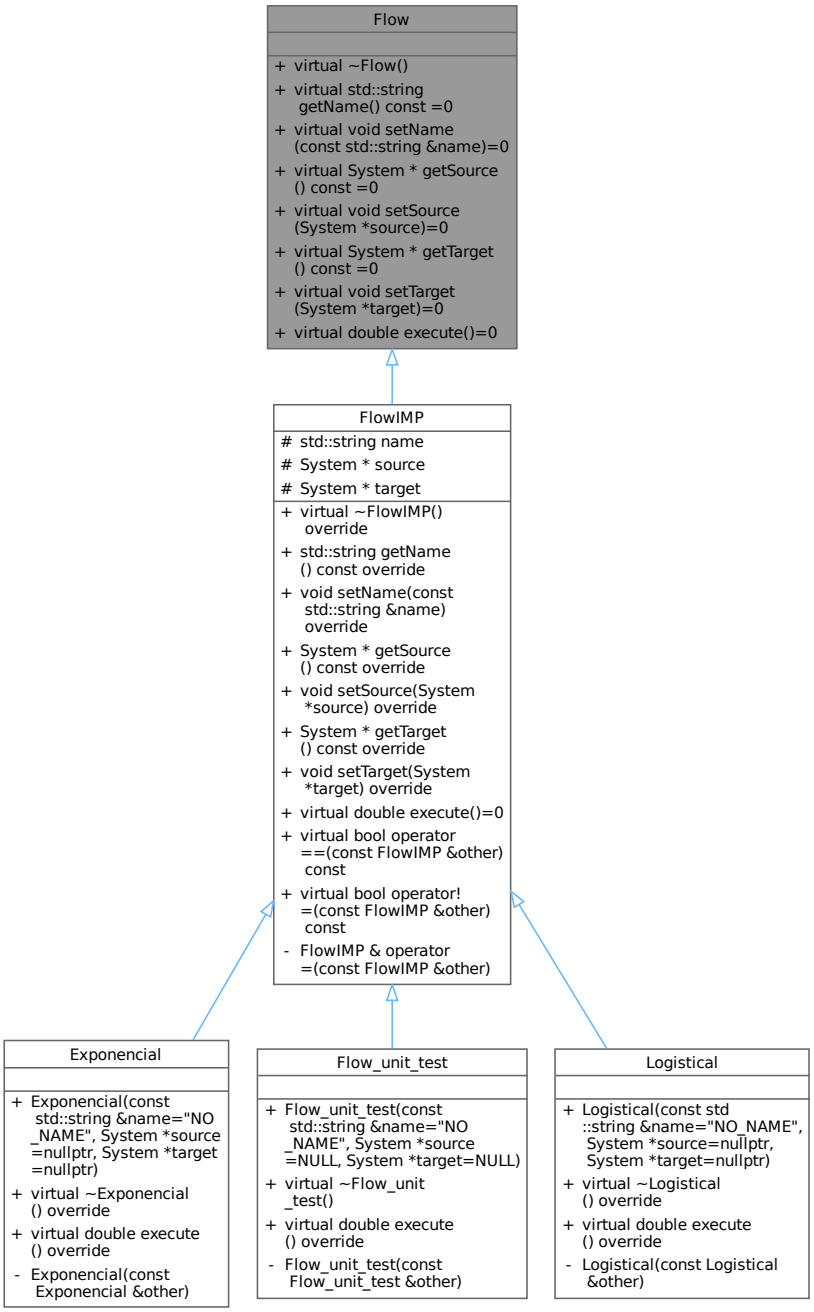
The documentation for this class was generated from the following files:

- /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

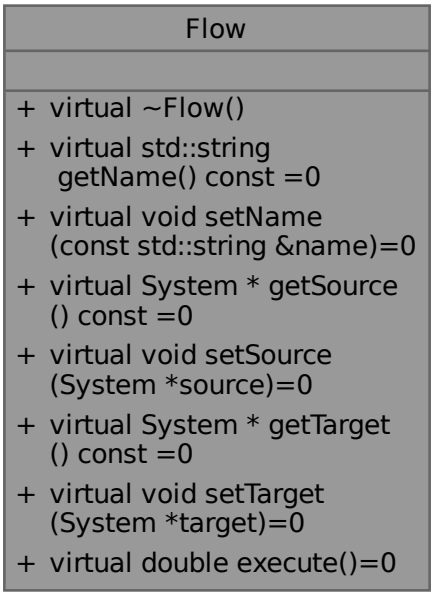
4.2 Flow Class Reference

`#include <Flow.hpp>`

Inheritance diagram for Flow:



Collaboration diagram for Flow:



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.

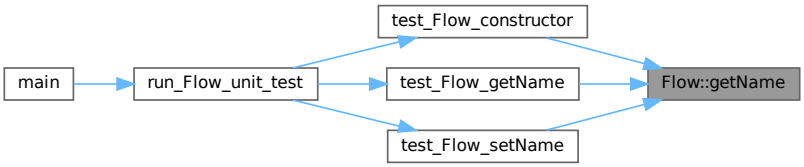
Returns

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 poiter.

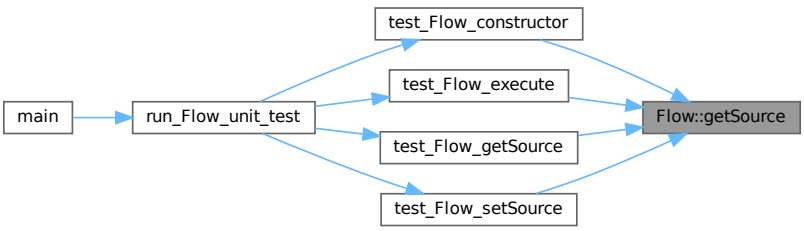
Returns

a system poiter 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 poiter.

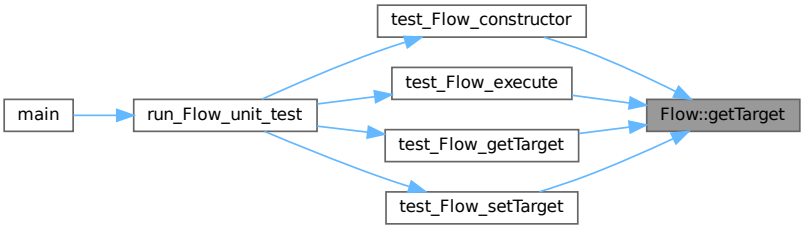
Returns

a system poiter 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

name	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 poiter to the source of a flow obj.

Parameters

source	system poiter 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 poiter to the target of a flow obj.

Parameters

target	system poiter must be passed to the method

Implemented in FlowIMP.

Referenced by test_Flow_setTarget().

Here is the caller graph for this function:

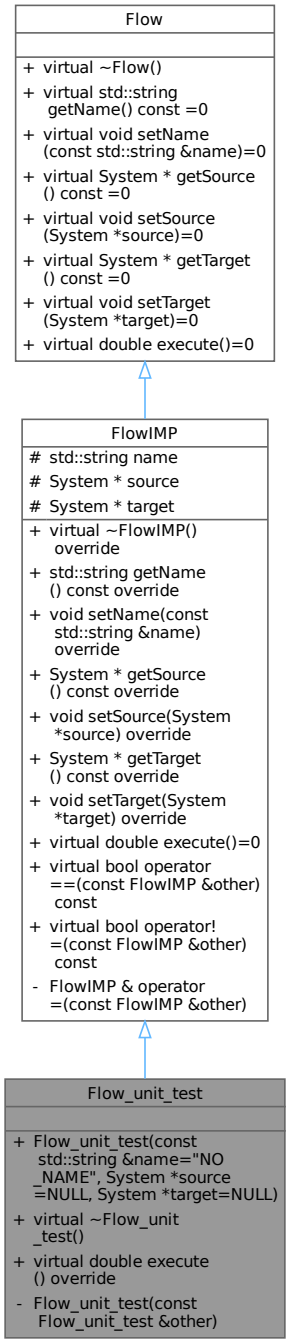
The documentation for this class was generated from the following file:

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

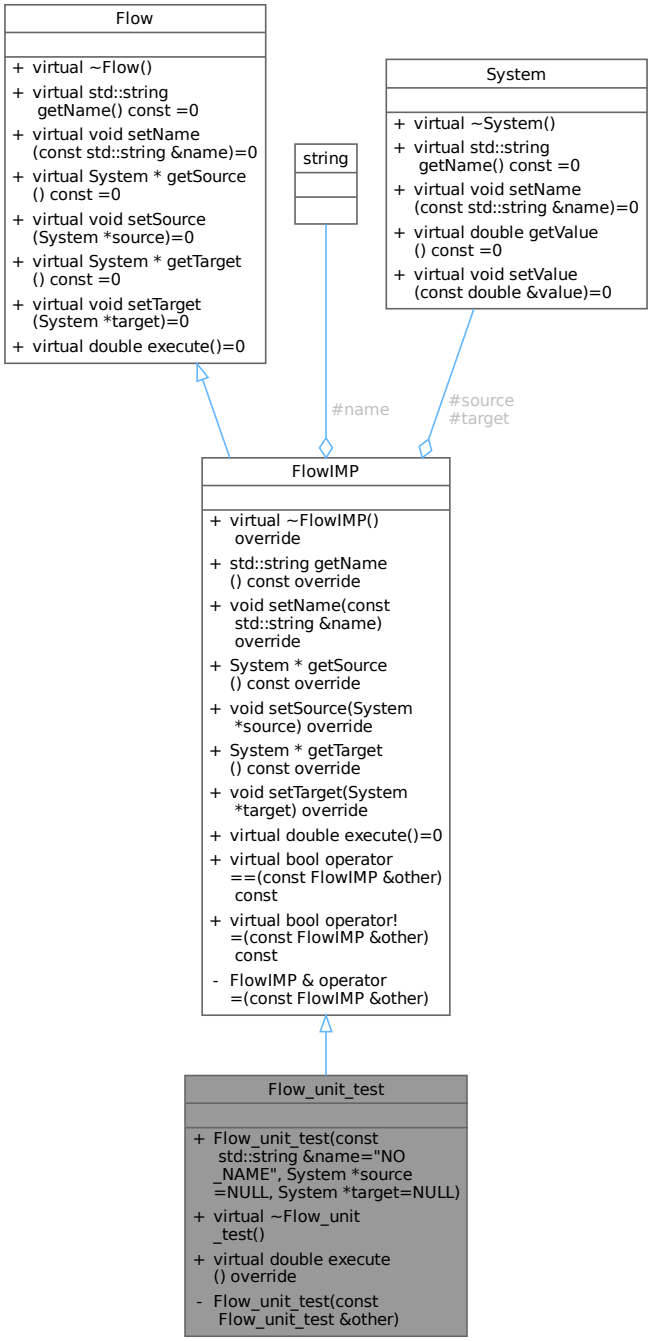
4.3 Flow_unit_test Class Reference

`#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 () override

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

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

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

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

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

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

  This method assigns a system poiter 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

other	Exponencial 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

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

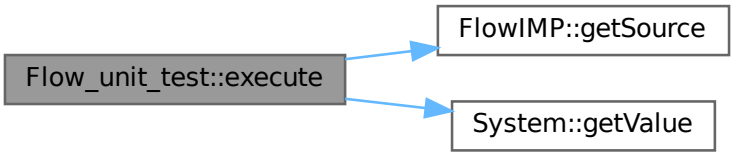
double

Implements FlowIMP.

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

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

Here is the call graph for this function:



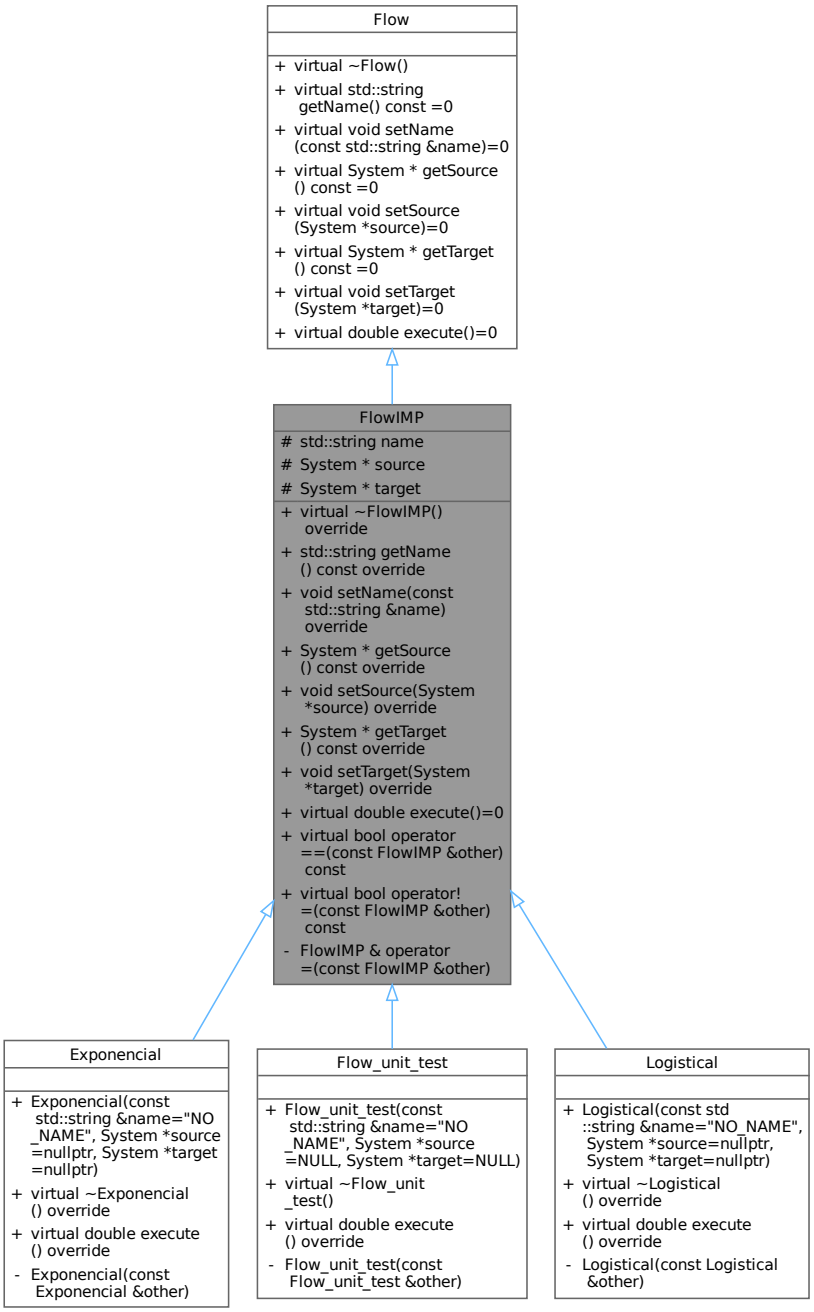
The documentation for this class was generated from the following files:

- /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

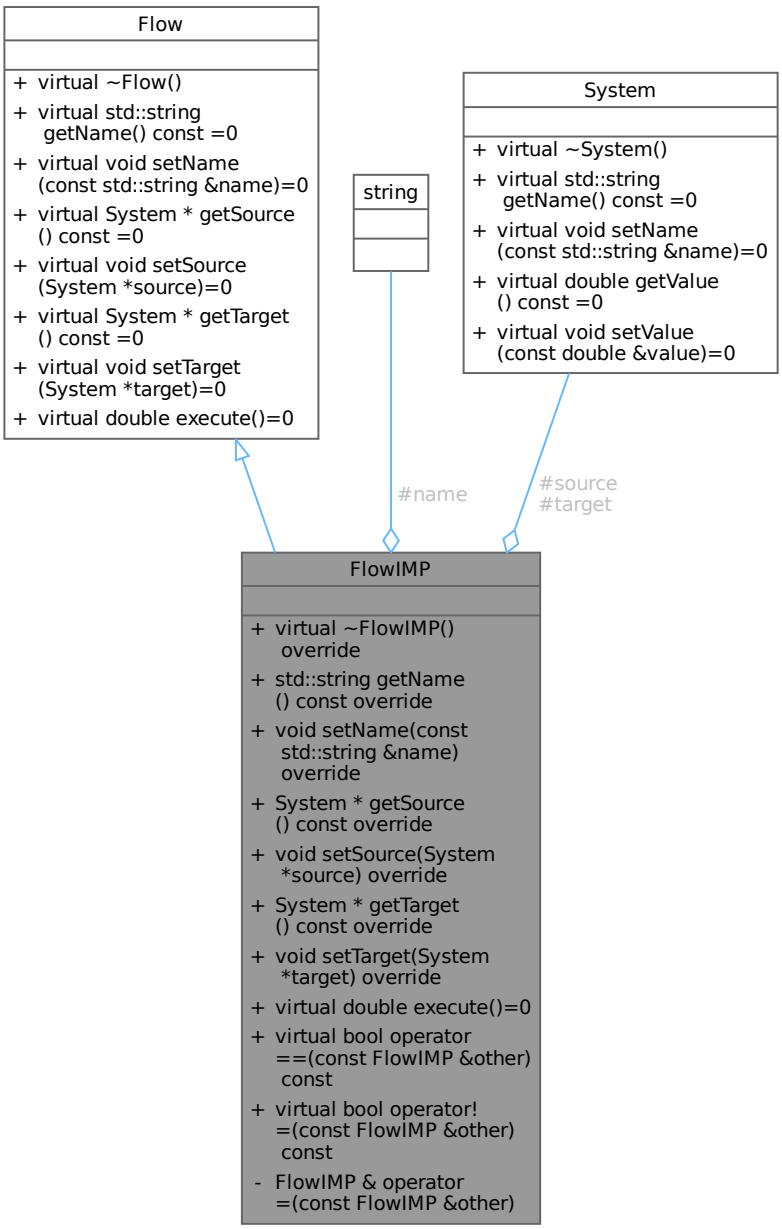
4.4 FlowIMP Class Reference

`#include <FlowIMP.hpp>`

Inheritance diagram for FlowIMP:



Collaboration diagram for FlowIMP:



Public Member Functions

- virtual ~FlowIMP () override

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

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

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

This method returns the source system poiter.

- void setSource (System *source) override

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

- System * getTarget () const override

  This method returns the target system poiter.

- void setTarget (System *target) override

  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.

- 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 ( )  [override], [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  [override], [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  [override], [virtual]
```

This method returns the source system poiter.

Returns

a system poiter containing the source memory address is returned

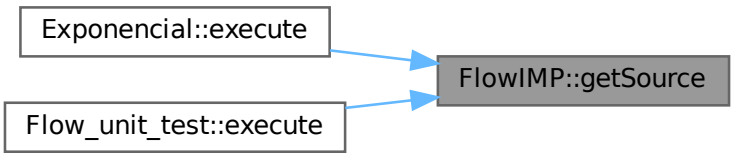
Implements Flow.

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

References source.

Referenced by Exponencial::execute(), and Flow_unit_test::execute().

Here is the caller graph for this function:



4.4.2.4 getTarget()

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

This method returns the target system poiter.

Returns

a system poiter containing the target memory address is returned

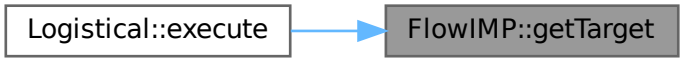
Implements Flow.

```
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

other	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

other	flow obj to be cloned must be passed

Returns

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

```
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

other	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 )  [override], [virtual]
```

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

Parameters

name	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 )  [override], [virtual]
```

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

Parameters

source	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 )  [override], [virtual]
```

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

Parameters

target	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().

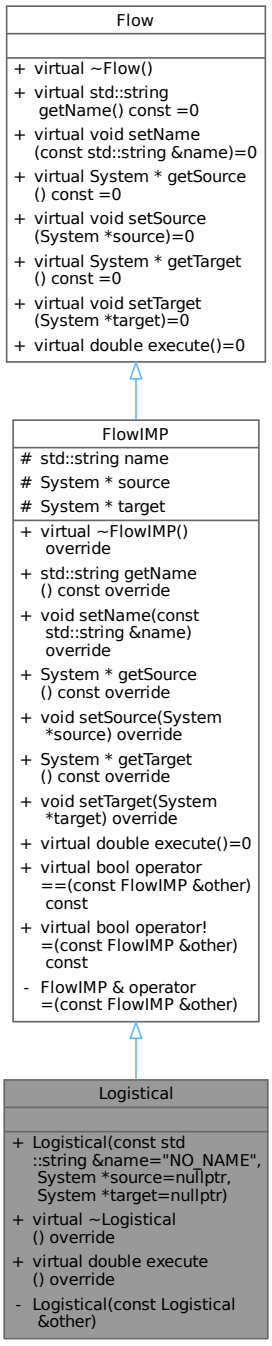
The documentation for this class was generated from the following files:

- /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

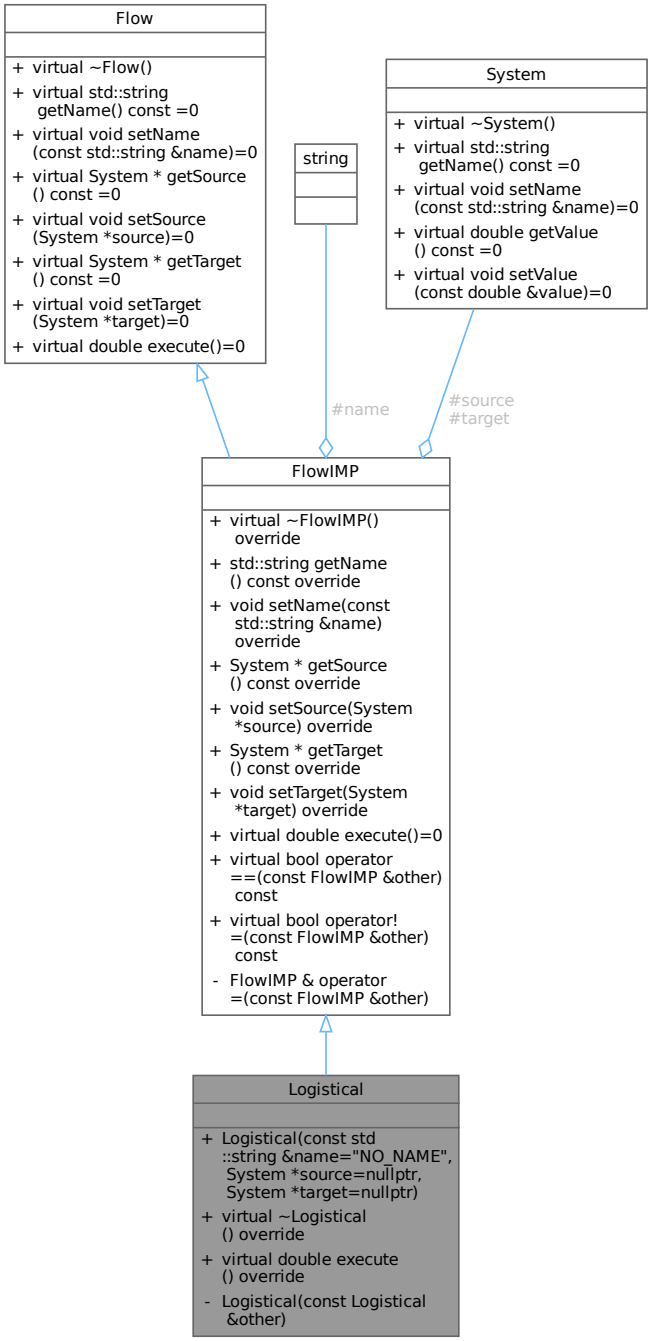
4.5 Logistical Class Reference

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

Inheritance diagram for Logistical:



Collaboration diagram for Logistical:



Public Member Functions

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

  Construct a new Logistical by name, source and target.
- virtual ~Logistical () override

  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 () override

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

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

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

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

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

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

  This method assigns a system poiter 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

other	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 = nullptr,
              System * target = nullptr )
```

Construct a new Logistical by name, source and target.

Parameters

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

```
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 ( )  [override], [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

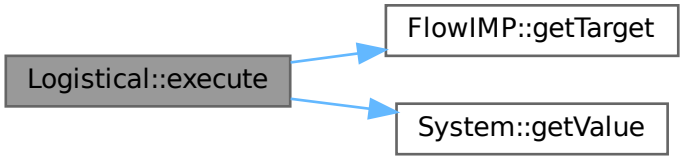
double

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:



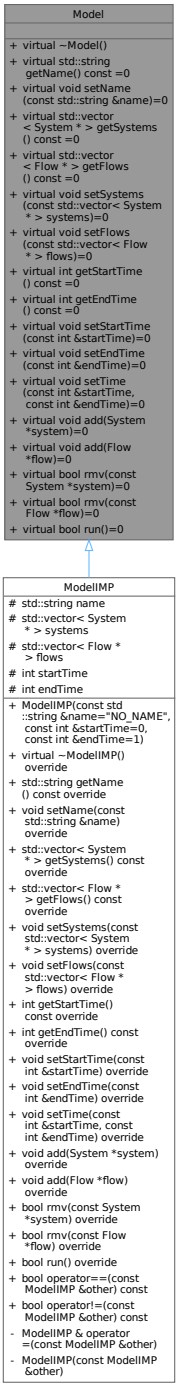
The documentation for this class was generated from the following files:

- /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

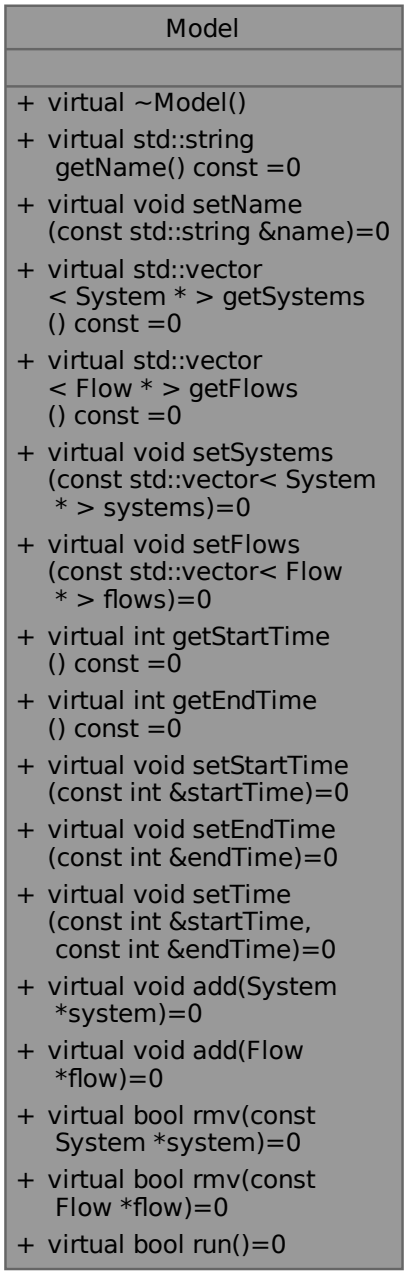
4.6 Model Class Reference

`#include <Model.hpp>`

Inheritance diagram for Model:



Collaboration diagram for Model:



Public Types

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

  typedef System vetor iterator
- typedef std::vector< Flow * >::iterator flowIterator

  typedef Flow vetor iterator

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
```

typedef Flow vetor iterator

4.6.1.2 systemIterator

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

typedef System vetor iterator

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.
```
00037 {};
```

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

flow	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.

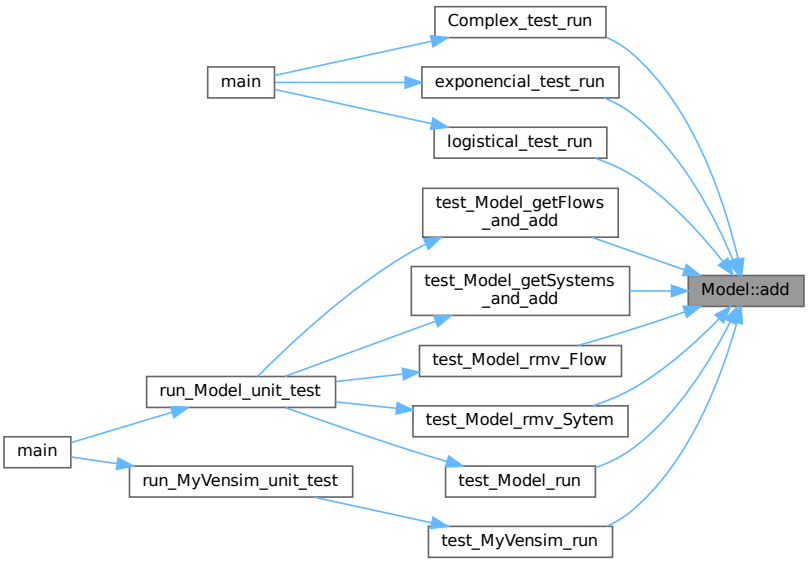
Parameters

system	System pointer must be passed to the method

Implemented in ModelIMP.

Referenced by Complex_test_run(), exponencial_test_run(), logistical_test_run(), test_Model_getFlows_and_add(), test_Model_getSystems_and_add(), test_Model_rmv_Flow(), test_Model_rmv_Sytem(), test_Model_run(), and test_MyVensim_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.

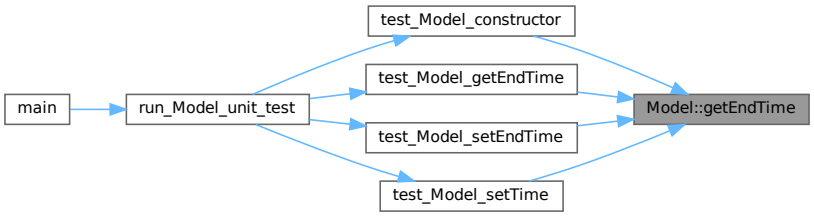
Returns

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.

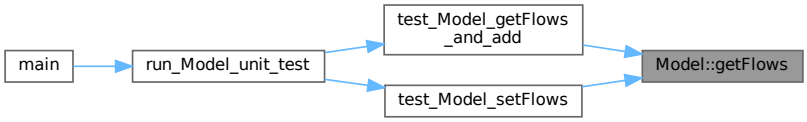
Returns

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.

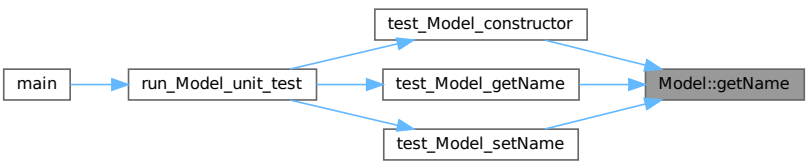
Returns

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.

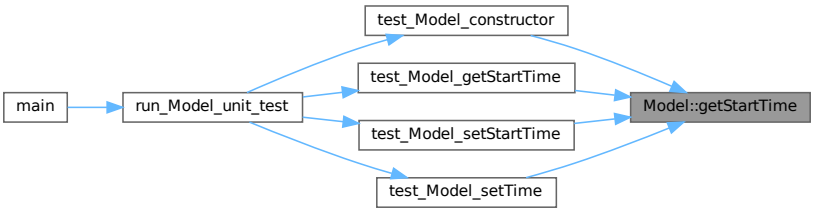
Returns

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.

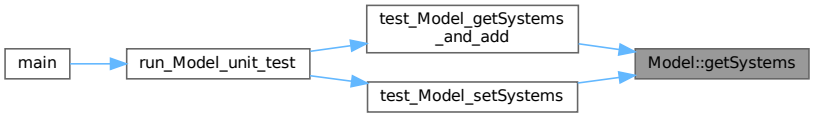
Returns

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

flow	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

system	System pointer iterator must be passed to the method

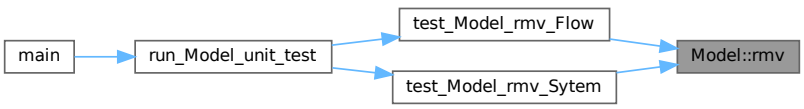
Returns

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.

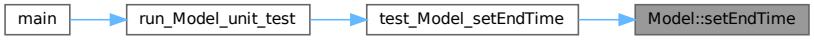
Parameters

endTime	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

flows	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

name	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.

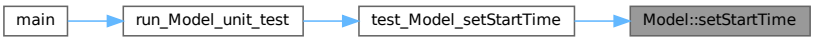
Parameters

startTime	int must be passed to the method

Implemented in ModelIMP.

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

systems	int must be passed to the method

Implemented in ModelIMP.

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

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

Implemented in ModelIMP.

Referenced by test_Model_setTime().

Here is the caller graph for this function:

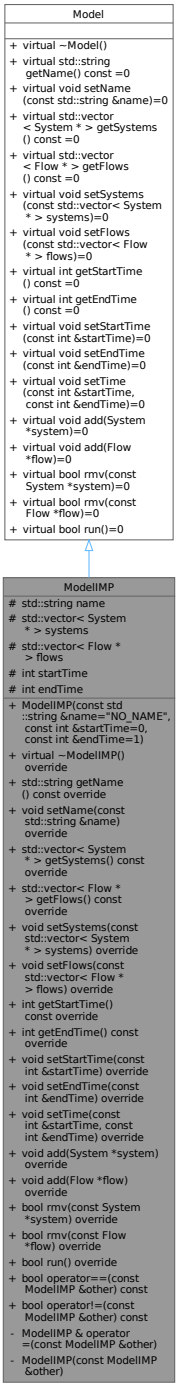
The documentation for this class was generated from the following file:

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

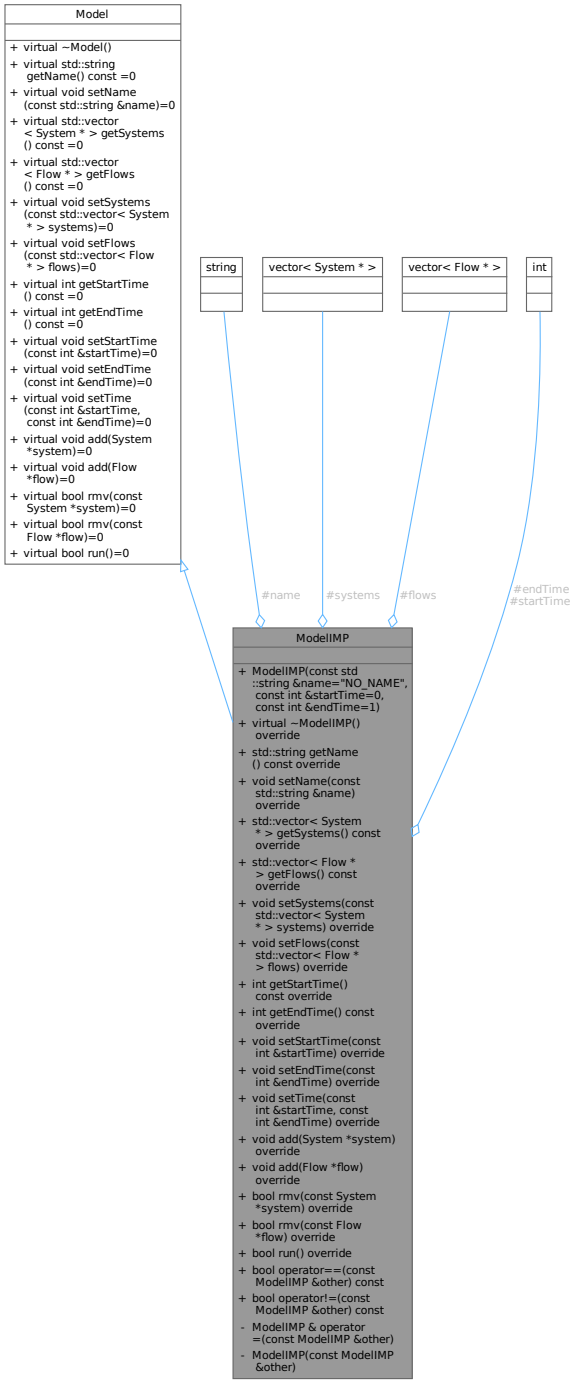
4.7 ModelIMP Class Reference

`#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 sart and end time.
- virtual ~ModelIMP () override

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

This method returns the name of a Model.

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

  This method assigns a string to the name of a Model.
- std::vector< System * > getSystems () const override

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

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

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

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

  This method returns the startTime of a Model.
- int getEndTime () const override

  This method returns the end of a Model.
- void setStartTime (const int &startTime) override

  This method assigns a int to the startTime of a Model.
- void setEndTime (const int &endTime) override

  This method assigns a int to the endTime of a Model.
- void setTime (const int &startTime, const int &endTime) override

  This method assigns a int to the startTime and endTime of a Model.
- void add (System *system) override

  This method add a System pointer to the vector of a Model.
- void add (Flow *flow) override

  This method add a Flow pointer to the vector of a Model.
- bool rmv (const System *system) override

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

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

  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 System vetor iterator
- typedef std::vector< Flow * >::iterator flowIterator

  typedef Flow vetor iterator

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

other	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

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

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

4.7.1.3 ~ModelIMP()

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

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

Parameters

flow	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 )  [override], [virtual]
```

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

Parameters

system	System pointer must be passed to the method

Implements Model.

```
00034 { systems.push_back(system); }
```

References systems.

4.7.2.3 getEndTime()

```
int ModelIMP::getEndTime ( ) const  [override], [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  [override], [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  [override], [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  [override], [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  [override], [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

other	model obj to be compare must be passed

Returns

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

```
00108                                                 {
00109     return (name != other.name || systems != other.systems || flows != other.flows || startTime !=
      other.startTime || endTime != other.endTime);
00110 }
```

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

other	Model obj to be cloned must be passed

Returns

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

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

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

other	model obj to be compare must be passed

Returns

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

```
00104                                                       {
00105     return (name == other.name && systems == other.systems && flows == other.flows && startTime ==
      other.startTime && endTime == other.endTime);
00106 }
```

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

4.7.2.11 rmv() [1/2]

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

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

Parameters

flow	Flow pointer iterator must be passed to the method

Returns

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

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 )  [override], [virtual]
```

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

Parameters

system	System pointer iterator must be passed to the method

Returns

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

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 ( )  [override], [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.

4.7.2.14 setEndTime()

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

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

Parameters

endTime	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 )  [override], [virtual]
```

This method assigns a vector to the flows of a Model.

Parameters

flows	int must be passed to the method

Implements Model.

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

References flows.

4.7.2.16 setName()

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

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

Parameters

name	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 )  [override], [virtual]
```

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

Parameters

startTime	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 )  [override], [virtual]
```

This method assigns a vector to the systems of a Model.

Parameters

systems	int must be passed to the method

Implements Model.

```
00023 { this->systems.clear(); 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 )  [override], [virtual]
```

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

Parameters

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

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().

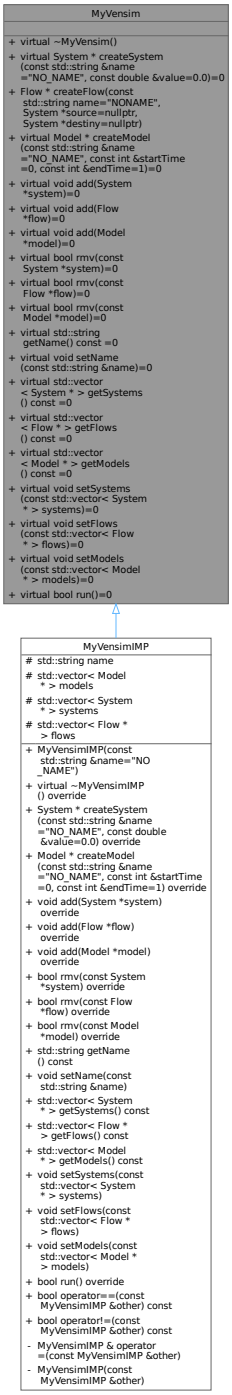
The documentation for this class was generated from the following files:

- /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

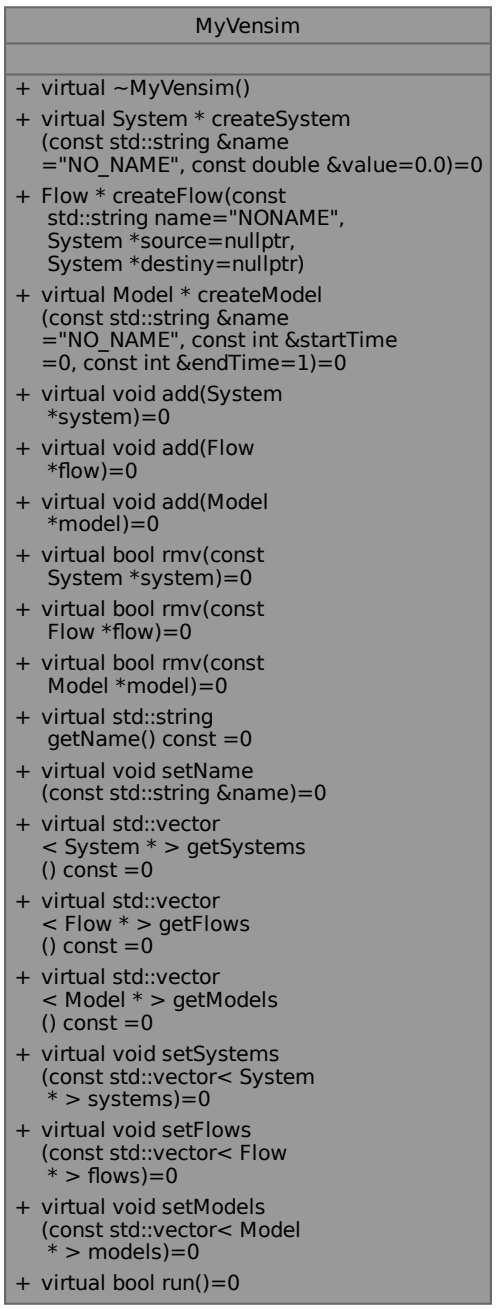
4.8 MyVensim Class Reference

```
#include <MyVensim.hpp>
```

Inheritance diagram for MyVensim:



Collaboration diagram for MyVensim:



Public Types

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

  typedef System vetor iterator
- typedef std::vector< Flow * >::iterator flowIterator

  typedef Flow vetor iterator
- typedef std::vector< Model * >::iterator modelIterator

  typedef Model vetor iterator

Public Member Functions

- virtual ~MyVensim ()

  This destructor is a virtual destructor of the class.
- virtual System * createSystem (const std::string &name="NO_NAME", const double &value=0.0)=0

  Create a instance of System.
- template<typename FlowType >
  Flow * createFlow (const std::string name="NONAME", System *source=nullptr, System *destiny=nullptr)

  Create a instance of Flow.
- virtual Model * createModel (const std::string &name="NO_NAME", const int &startTime=0, const int &end↩
  Time=1)=0

  Create a instance of Model.
- virtual void add (System *system)=0

  This method add a System pointer to the vector.
- virtual void add (Flow *flow)=0

  This method add a Flow pointer to the vector.
- virtual void add (Model *model)=0

  This method add a Model pointer to the vector.
- virtual bool rmv (const System *system)=0

  This method remove a System pointer of the vector.
- virtual bool rmv (const Flow *flow)=0

  This method remove a Flow pointer of the vector.
- virtual bool rmv (const Model *model)=0

  This method remove a Model pointer of the vector.
- virtual std::string getName () const =0

  This method returns the name of a MyVensim.
- virtual void setName (const std::string &name)=0

  This method assigns a string to the name of a MyVensim.
- 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 std::vector< Model * > getModels () const =0

  This method returns the vector of Models.
- virtual void setSystems (const std::vector< System * > systems)=0

  This method assigns a vector to the systems of a MyVensim.
- virtual void setFlows (const std::vector< Flow * > flows)=0

  This method assigns a vector to the flows of a MyVensim.
- virtual void setModels (const std::vector< Model * > models)=0

  This method assigns a vector to the models of a MyVensim.
- virtual bool run ()=0

  This method run all model.

4.8.1 Member Typedef Documentation

4.8.1.1 flowIterator

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

typedef Flow vetor iterator

4.8.1.2 modelIterator

```
typedef std::vector<Model*>::iterator MyVensim::modelIterator
```

typedef Model vetor iterator

4.8.1.3 systemIterator

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

typedef System vetor iterator

4.8.2 Constructor & Destructor Documentation

4.8.2.1 ~MyVensim()

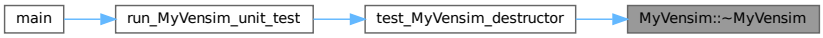
```
virtual MyVensim::~MyVensim ( )  [inline], [virtual]
```

This destructor is a virtual destructor of the class.

```
00042 {}
```

Referenced by test_MyVensim_destructor().

Here is the caller graph for this function:



4.8.3 Member Function Documentation

4.8.3.1 add() [1/3]

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

This method add a Flow pointer to the vector.

Parameters

flow	Flow pointer must be passed to the method

Implemented in MyVensimIMP.

4.8.3.2 add() [2/3]

```
virtual void MyVensim::add (
            Model * model )  [pure virtual]
```

This method add a Model pointer to the vector.

Parameters

model	Model pointer must be passed to the method

Implemented in MyVensimIMP.

4.8.3.3 add() [3/3]

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

This method add a System pointer to the vector.

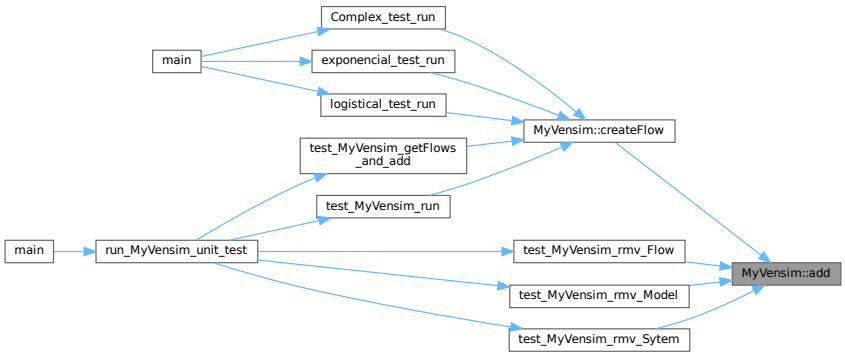
Parameters

system	System pointer must be passed to the method

Implemented in MyVensimIMP.

Referenced by createFlow(), test_MyVensim_rmv_Flow(), test_MyVensim_rmv_Model(), and test_MyVensim_rmv_Sytem().

Here is the caller graph for this function:



4.8.3.4 createFlow()

```
template<typename FlowType >
Flow * MyVensim::createFlow (
```

```
const std::string name = "NONAME",
System * source = nullptr,
System * destiny = nullptr )  [inline]
```

Create a instance of Flow.

Parameters

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

Returns

A Flow pointer to the new instance of Flow

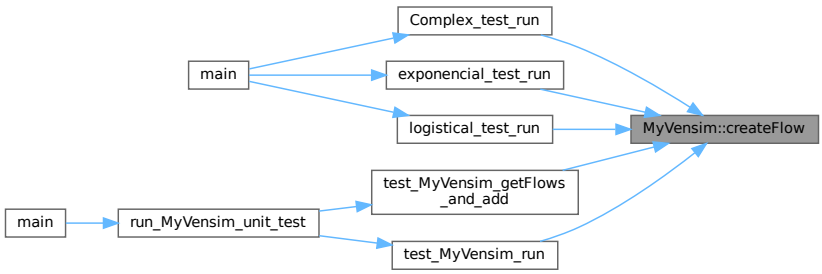
```
00060
    {
00061             Flow* f = new FlowType(name, source, destiny);
00062             add(f);
00063             return f;
00064         }
```

References add().

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

Here is the call graph for this function:

Here is the caller graph for this function:



4.8.3.5 createModel()

```
virtual Model * MyVensim::createModel (
            const std::string & name = "NO_NAME",
            const int & startTime = 0,
            const int & endTime = 1 )  [pure virtual]
```

Create a instance of Model.

Parameters

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

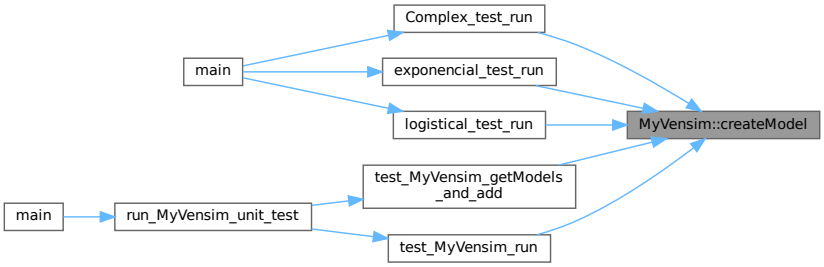
Returns

A Model pointer to the new instance of Model

Implemented in MyVensimIMP.

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

Here is the caller graph for this function:



4.8.3.6 createSystem()

```
virtual System * MyVensim::createSystem (
            const std::string & name = "NO_NAME",
            const double & value = 0.0 )  [pure virtual]
```

Create a instance of System.

Parameters

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

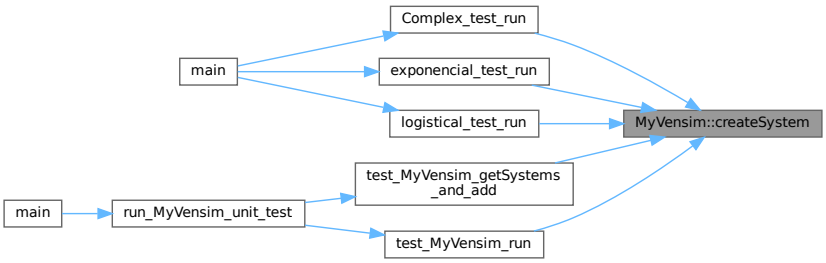
Returns

A System pointer to the new instance of System

Implemented in MyVensimIMP.

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

Here is the caller graph for this function:



4.8.3.7 getFlows()

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

This method returns the vector of flows.

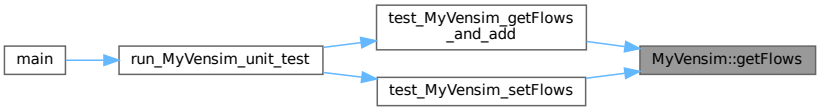
Returns

a vector containing Flows is returned

Implemented in MyVensimIMP.

Referenced by test_MyVensim_getFlows_and_add(), and test_MyVensim_setFlows().

Here is the caller graph for this function:



4.8.3.8 getModels()

```
virtual std::vector< Model * > MyVensim::getModels ( ) const  [pure virtual]
```

This method returns the vector of Models.

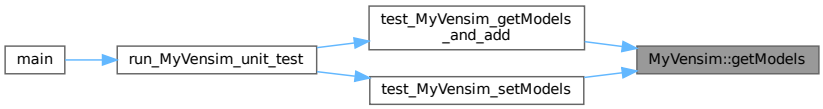
Returns

a vector containing Models is returned

Implemented in MyVensimIMP.

Referenced by test_MyVensim_getModels_and_add(), and test_MyVensim_setModels().

Here is the caller graph for this function:



4.8.3.9 getName()

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

This method returns the name of a MyVensim.

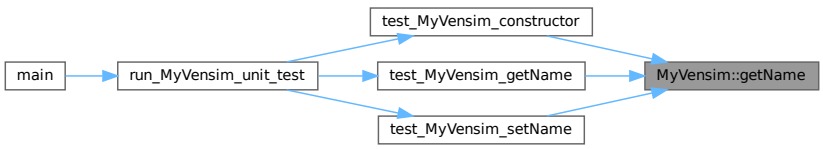
Returns

a string containing the name is returned

Implemented in MyVensimIMP.

Referenced by test_MyVensim_constructor(), test_MyVensim_getName(), and test_MyVensim_setName().

Here is the caller graph for this function:



4.8.3.10 getSystems()

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

This method returns the vector of Systems.

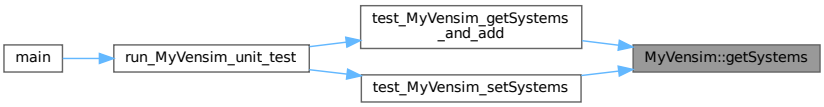
Returns

a vector containing Systems is returned

Implemented in MyVensimIMP.

Referenced by test_MyVensim_getSystems_and_add(), and test_MyVensim_setSystems().

Here is the caller graph for this function:



4.8.3.11 rmv() [1/3]

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

This method remove a Flow pointer of the vector.

Parameters

flow	Flow pointer iterator must be passed to the method

Returns

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

Implemented in MyVensimIMP.

4.8.3.12 rmv() [2/3]

```
virtual bool MyVensim::rmv (
            const Model * model )  [pure virtual]
```

This method remove a Model pointer of the vector.

Parameters

model	Model pointer iterator must be passed to the method

Returns

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

Implemented in MyVensimIMP.

4.8.3.13 rmv() [3/3]

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

This method remove a System pointer of the vector.

Parameters

system	System pointer iterator must be passed to the method

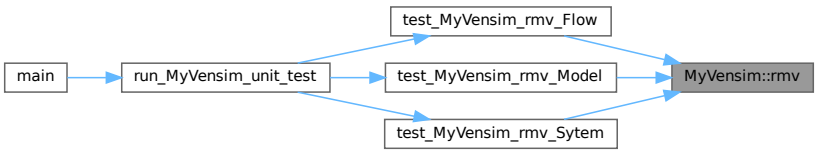
Returns

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

Implemented in MyVensimIMP.

Referenced by test_MyVensim_rmv_Flow(), test_MyVensim_rmv_Model(), and test_MyVensim_rmv_Sytem().

Here is the caller graph for this function:



4.8.3.14 run()

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

This method run all model.

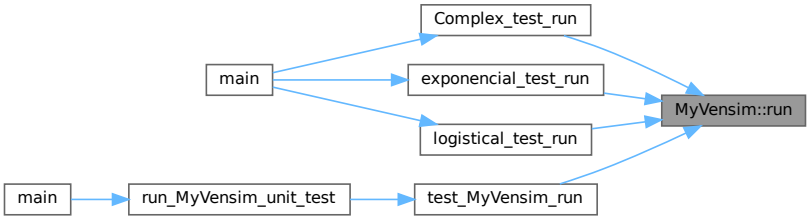
Returns

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

Implemented in MyVensimIMP.

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

Here is the caller graph for this function:



4.8.3.15 setFlows()

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

This method assigns a vector to the flows of a MyVensim.

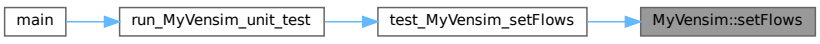
Parameters

flows	int must be passed to the method

Implemented in MyVensimIMP.

Referenced by test_MyVensim_setFlows().

Here is the caller graph for this function:



4.8.3.16 setModels()

```
virtual void MyVensim::setModels (
            const std::vector< Model * > models )  [pure virtual]
```

This method assigns a vector to the models of a MyVensim.

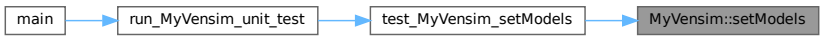
Parameters

models	int must be passed to the method

Implemented in MyVensimIMP.

Referenced by test_MyVensim_setModels().

Here is the caller graph for this function:



4.8.3.17 setName()

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

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

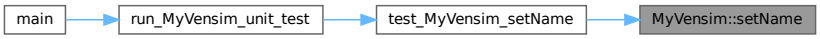
Parameters

name	string must be passed to the method

Implemented in MyVensimIMP.

Referenced by test_MyVensim_setName().

Here is the caller graph for this function:



4.8.3.18 setSystems()

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

This method assigns a vector to the systems of a MyVensim.

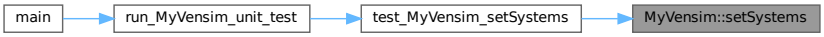
Parameters

systems	int must be passed to the method

Implemented in MyVensimIMP.

Referenced by test_MyVensim_setSystems().

Here is the caller graph for this function:



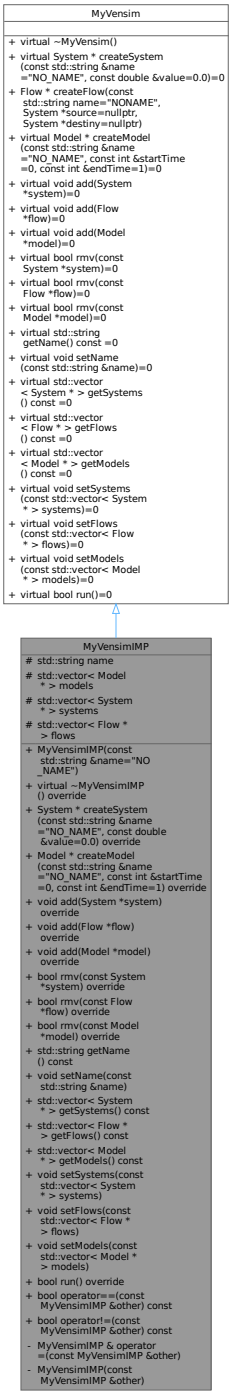
The documentation for this class was generated from the following file:

- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/MyVensim.hpp

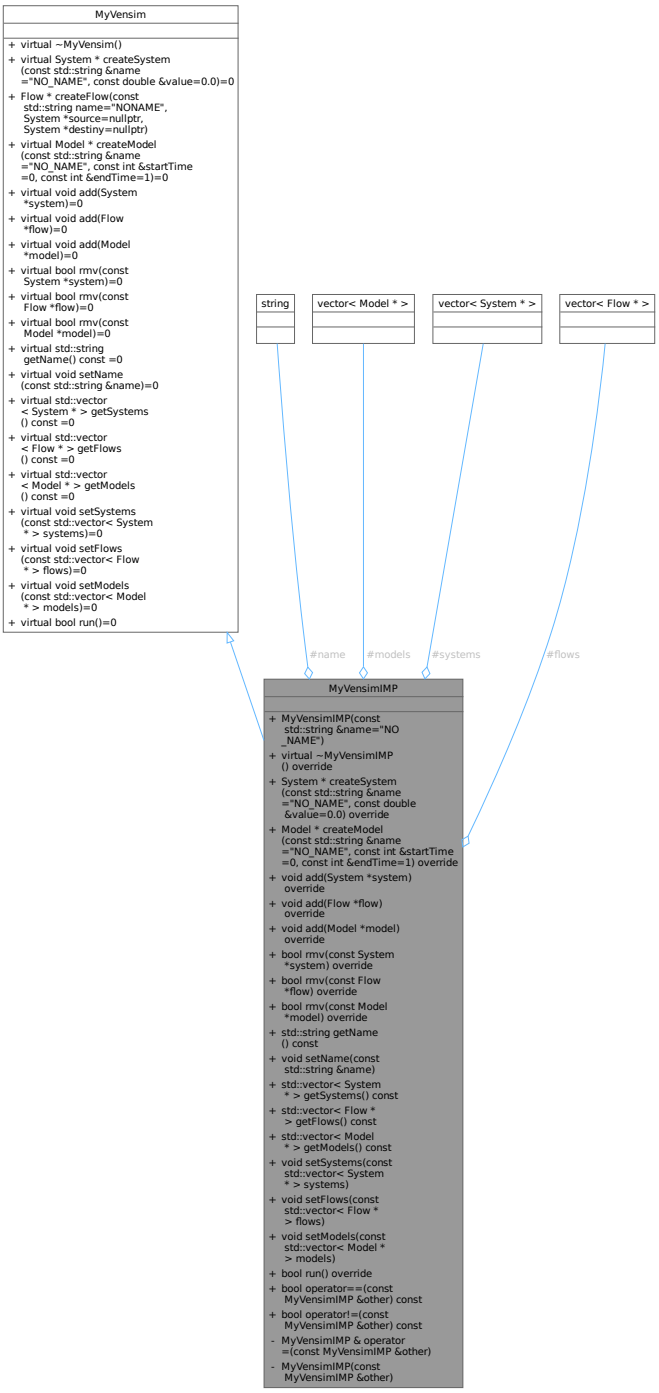
4.9 MyVensimIMP Class Reference

```
#include <MyVensimIMP.hpp>
```

Inheritance diagram for MyVensimIMP:



Collaboration diagram for MyVensimIMP:



Public Member Functions

- MyVensimIMP (const std::string &name="NO_NAME")

  Construct a new Model by name and sart and end time.
- virtual ~MyVensimIMP () override

  This destructor is a virtual destructor of the class.
- System * createSystem (const std::string &name="NO_NAME", const double &value=0.0) override

Create a instance of System.

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

  Create a instance of Model.
- void add (System *system) override

  This method add a System pointer to the vector.
- void add (Flow *flow) override

  This method add a Flow pointer to the vector.
- void add (Model *model) override

  This method add a Model pointer to the vector.
- bool rmv (const System *system) override

  This method remove a System pointer of the vector.
- bool rmv (const Flow *flow) override

  This method remove a Flow pointer of the vector.
- bool rmv (const Model *model) override

  This method remove a Model pointer of the vector.
- std::string getName () const

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

  This method assigns a string to the name of a MyVensim.
- std::vector< System * > getSystems () const

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

  This method returns the vector of flows.
- std::vector< Model * > getModels () const

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

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

  This method assigns a vector to the flows of a MyVensim.
- void setModels (const std::vector< Model * > models)

  This method assigns a vector to the models of a MyVensim.
- bool run () override

  This method run all model.
- bool operator== (const MyVensimIMP &other) const

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

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

Public Member Functions inherited from MyVensim

- virtual ~MyVensim ()

  This destructor is a virtual destructor of the class.
- template<typename FlowType >
  Flow * createFlow (const std::string name="NONAME", System *source=nullptr, System *destiny=nullptr)

  Create a instance of Flow.

Protected Attributes

- std::string name
- std::vector< Model * > models
- std::vector< System * > systems
- std::vector< Flow * > flows

Private Member Functions

- MyVensimIMP & operator= (const MyVensimIMP &other)

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

  Construct a new MyVensim by a obj.

Additional Inherited Members

Public Types inherited from MyVensim

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

  typedef System vetor iterator
- typedef std::vector< Flow * >::iterator flowIterator

  typedef Flow vetor iterator
- typedef std::vector< Model * >::iterator modelIterator

  typedef Model vetor iterator

4.9.1 Constructor & Destructor Documentation

4.9.1.1 MyVensimIMP() [1/2]

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

Construct a new MyVensim by a obj.

Parameters

other	MyVensim obj

```
00005                                                {
00006     models.clear();
00007     flows.clear();
00008     systems.clear();
00009     for (auto i : other.models) models.push_back(i);
00010     for (auto i : other.flows) flows.push_back(i);
00011     for (auto i : other.systems) systems.push_back(i);
00012 }
```

References flows, models, and systems.

4.9.1.2 MyVensimIMP() [2/2]

```
MyVensimIMP::MyVensimIMP (
            const std::string & name = "NO_NAME" )
```

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

Parameters

name	string with default value "NO_NAME"

```
00003 : name(name) {}
```

4.9.1.3 ~MyVensimIMP()

```
MyVensimIMP::~MyVensimIMP ( )  [override], [virtual]
```

This destructor is a virtual destructor of the class.

```
00014                           {
00015     models.clear();
00016     flows.clear();
00017     systems.clear();
00018 }
```

References flows, models, and systems.

4.9.2 Member Function Documentation

4.9.2.1 add() [1/3]

```
void MyVensimIMP::add (
            Flow * flow )  [override], [virtual]
```

This method add a Flow pointer to the vector.

Parameters

flow	Flow pointer must be passed to the method

Implements MyVensim.

```
00046 { flows.push_back(flow); }
```

References flows.

4.9.2.2 add() [2/3]

```
void MyVensimIMP::add (
            Model * model )  [override], [virtual]
```

This method add a Model pointer to the vector.

Parameters

model	Model pointer must be passed to the method

Implements MyVensim.

```
00047 { models.push_back(model); };
```

References models.

4.9.2.3 add() [3/3]

```
void MyVensimIMP::add (
            System * system )  [override], [virtual]
```

This method add a System pointer to the vector.

Parameters

system	System pointer must be passed to the method

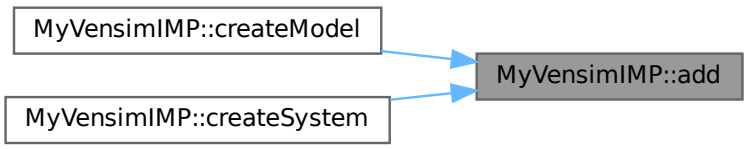
Implements MyVensim.

```
00045 { systems.push_back(system); }
```

References systems.

Referenced by createModel(), and createSystem().

Here is the caller graph for this function:



4.9.2.4 createModel()

```
Model * MyVensimIMP::createModel (
            const std::string & name = "NO_NAME",
            const int & startTime = 0,
            const int & endTime = 1 )  [override], [virtual]
```

Create a instance of Model.

Parameters

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

Returns

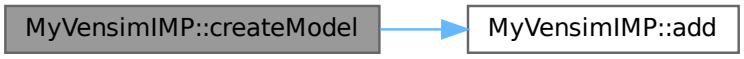
A Model pointer to the new instance of Model

Implements MyVensim.

```
00026                                                                                {
00027     Model* m = new ModelIMP(name, startTime, endTime);
00028     add(m);
00029     return m;
00030 }
```

References add(), and name.

Here is the call graph for this function:



4.9.2.5 createSystem()

```
System * MyVensimIMP::createSystem (
            const std::string & name = "NO_NAME",
            const double & value = 0.0 )  [override], [virtual]
```

Create a instance of System.

Parameters

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

Returns

A System pointer to the new instance of System

Implements MyVensim.

```
00020                                                                          {
00021     System* s = new SystemIMP(name, value);
00022     add(s);
00023     return s;
00024 }
```

References add(), and name.

Here is the call graph for this function:

4.9.2.6 getFlows()

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

This method returns the vector of flows.

Returns

a vector containing Flows is returned

Implements MyVensim.

```
00038 { return flows; }
```

References flows.

4.9.2.7 getModels()

```
std::vector< Model * > MyVensimIMP::getModels ( ) const  [virtual]
```

This method returns the vector of Models.

Returns

a vector containing Models is returned

Implements MyVensim.

```
00039 { return models; }
```

References models.

4.9.2.8 getName()

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

This method returns the name of a MyVensim.

Returns

a string containing the name is returned

Implements MyVensim.

```
00034 { return name; }
```

References name.

4.9.2.9 getSystems()

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

This method returns the vector of Systems.

Returns

a vector containing Systems is returned

Implements MyVensim.

```
00037 { return systems; }
```

References systems.

4.9.2.10 operator"!=()

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

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

Parameters

other	model obj to be compare must be passed

Returns

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

```
00099                                                       {
00100     return (name != other.name || systems != other.systems || flows != other.flows || models !=
      other.models );
00101 }
```

References flows, models, name, and systems.

4.9.2.11 operator=()

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

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

Parameters

other	MyVensim obj to be cloned must be passed

Returns

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

```
00083                                                                    {
00084     if(other == *this) return *this;
00085     name = other.name;
00086     flows.clear();
00087     systems.clear();
00088     models.clear();
00089     for (auto i : other.flows) flows.push_back(i);
00090     for (auto i : other.systems) systems.push_back(i);
00091     for (auto i : other.models) models.push_back(i);
00092     return *this;
00093 }
```

References flows, models, name, and systems.

4.9.2.12 operator==()

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

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

Parameters

other	model obj to be compare must be passed

Returns

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

```
00095                                                                {
00096     return (name == other.name && systems == other.systems && flows == other.flows && models ==
      other.models );
00097 }
```

References flows, models, name, and systems.

4.9.2.13 rmv() [1/3]

```
bool MyVensimIMP::rmv (
            const Flow * flow )  [override], [virtual]
```

This method remove a Flow pointer of the vector.

Parameters

flow	Flow pointer iterator must be passed to the method

Returns

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

Implements MyVensim.

```
00058                                    {
00059     for(flowIterator i = flows.begin(); i < flows.end(); i++)
00060         if(*i == flow){
00061             flows.erase(i);
00062             return true;
```

```
00063         }
00064     return false;
00065 }
```

References flows.

4.9.2.14 rmv() [2/3]

```
bool MyVensimIMP::rmv (
            const Model * model )  [override], [virtual]
```

This method remove a Model pointer of the vector.

Parameters

model	Model pointer iterator must be passed to the method

Returns

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

Implements MyVensim.

```
00066                                       {
00067     for(modelIterator i = models.begin(); i < models.end(); i++)
00068         if(*i == model){
00069             models.erase(i);
00070             return true;
00071         }
00072     return false;
00073 }
```

References models.

4.9.2.15 rmv() [3/3]

```
bool MyVensimIMP::rmv (
            const System * system )  [override], [virtual]
```

This method remove a System pointer of the vector.

Parameters

system	System pointer iterator must be passed to the method

Returns

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

Implements MyVensim.

```
00050                                         {
00051     for(systemIterator i = systems.begin(); i < systems.end(); i++)
00052         if(*i == system){
00053             systems.erase(i);
00054             return true;
00055         }
00056     return false;
00057 }
```

References systems.

4.9.2.16 run()

```
bool MyVensimIMP::run ( )  [override], [virtual]
```

This method run all model.

Returns

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

Implements MyVensim.

```
00075                       {
00076     if(models.empty()) return false;
00077     for(auto i : models) if(!(i->run())) return false;
00078     return true;
00079 }
```

References models.

4.9.2.17 setFlows()

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

This method assigns a vector to the flows of a MyVensim.

Parameters

flows	int must be passed to the method

Implements MyVensim.

```
00041 { this->flows.clear(); for(auto i : flows) this->flows.push_back(i); }
```

References flows.

4.9.2.18 setModels()

```
void MyVensimIMP::setModels (
            const std::vector< Model * > models )  [virtual]
```

This method assigns a vector to the models of a MyVensim.

Parameters

models	int must be passed to the method

Implements MyVensim.

```
00042 { { this->models.clear(); for(auto i : models) this->models.push_back(i); } }
```

References models.

4.9.2.19 setName()

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

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

Parameters

name	string must be passed to the method

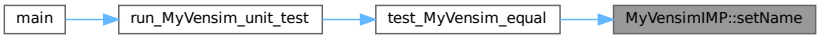
Implements MyVensim.

```
00035 { this->name = name; }
```

References name.

Referenced by test_MyVensim_equal().

Here is the caller graph for this function:



4.9.2.20 setSystems()

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

This method assigns a vector to the systems of a MyVensim.

Parameters

systems	int must be passed to the method

Implements MyVensim.

```
00040 { this->systems.clear(); for(auto i : systems) this->systems.push_back(i); }
```

References systems.

4.9.3 Member Data Documentation

4.9.3.1 flows

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

Flow pointers vector.

Referenced by add(), getFlows(), MyVensimIMP(), operator!=(), operator=(), operator==(), rmv(), setFlows(), and ~MyVensimIMP().

4.9.3.2 models

```
std::vector<Model*> MyVensimIMP::models  [protected]
```

Model pointers vector.

Referenced by add(), getModels(), MyVensimIMP(), operator!=(), operator=(), operator==(), rmv(), run(), setModels(), and ~MyVensimIMP().

4.9.3.3 name

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

Name string attribute.

Referenced by createModel(), createSystem(), getName(), operator!=(), operator=(), operator==(), and setName().

4.9.3.4 systems

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

System pointers vector.

Referenced by add(), getSystems(), MyVensimIMP(), operator!=(), operator=(), operator==(), rmv(), setSystems(), and ~MyVensimIMP().

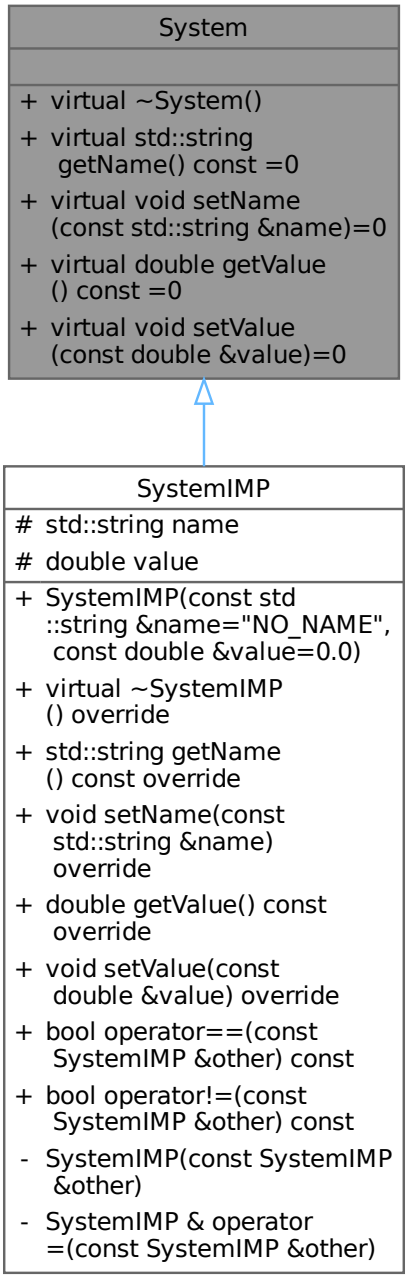
The documentation for this class was generated from the following files:

- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/MyVensimIMP.hpp
- /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/MyVensimIMP.cpp

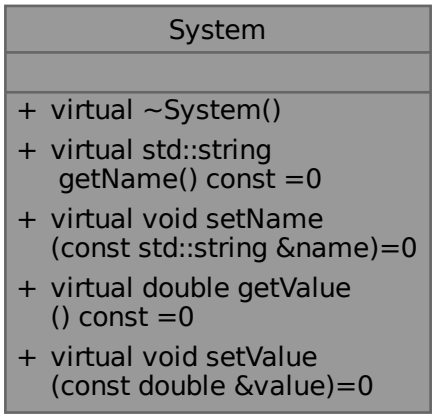
4.10 System Class Reference

```
#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.

4.10.1 Constructor & Destructor Documentation

4.10.1.1 ~System()

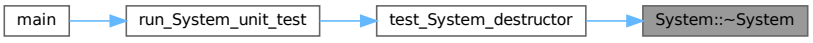
```
virtual System::~System ( )  [inline], [virtual]
```

This destructor is a virtual destructor of the Class.

```
00023 {};
```

Referenced by test_System_destructor().

Here is the caller graph for this function:



4.10.2 Member Function Documentation

4.10.2.1 getName()

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

This method returns the name of a system.

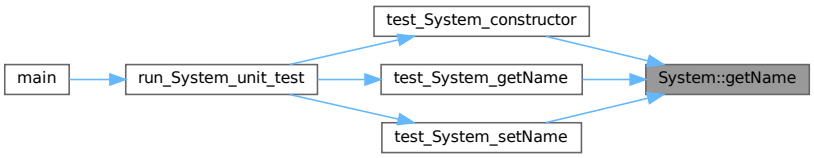
Returns

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:



4.10.2.2 getValue()

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

This method returns the value of a system.

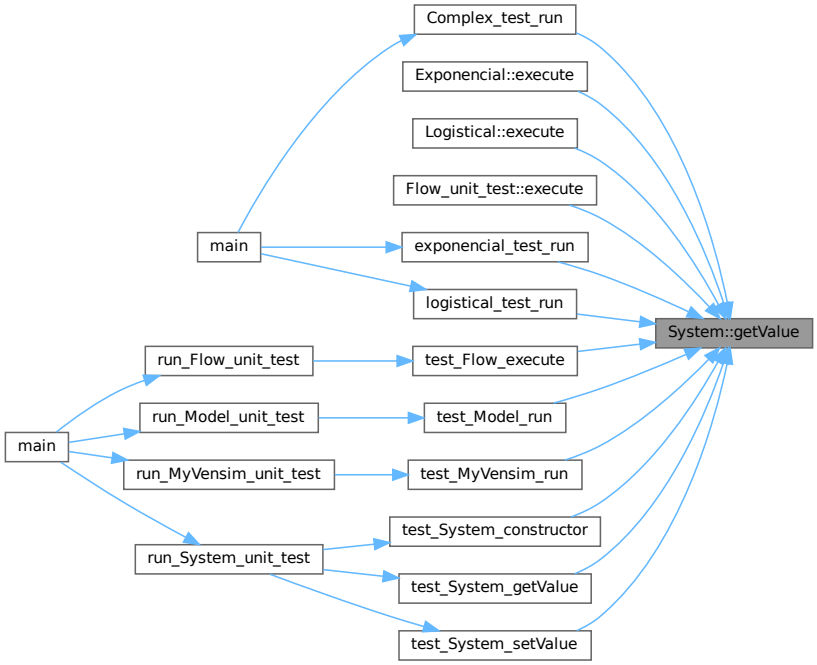
Returns

a double containing the value is returned

Implemented in SystemIMP.

Referenced by Complex_test_run(), Exponencial::execute(), Logistical::execute(), Flow_unit_test::execute(), exponencial_test_run(), logistical_test_run(), test_Flow_execute(), test_Model_run(), test_MyVensim_run(), test_System_constructor(), test_System_getValue(), and test_System_setValue().

Here is the caller graph for this function:



4.10.2.3 setName()

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

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

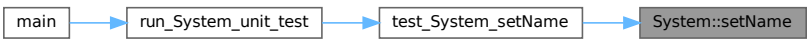
Parameters

name	string must be passed to the method

Implemented in SystemIMP.

Referenced by test_System_setName().

Here is the caller graph for this function:



4.10.2.4 setValue()

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

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

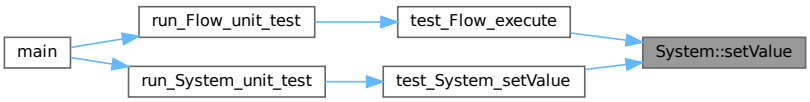
Parameters

value	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:



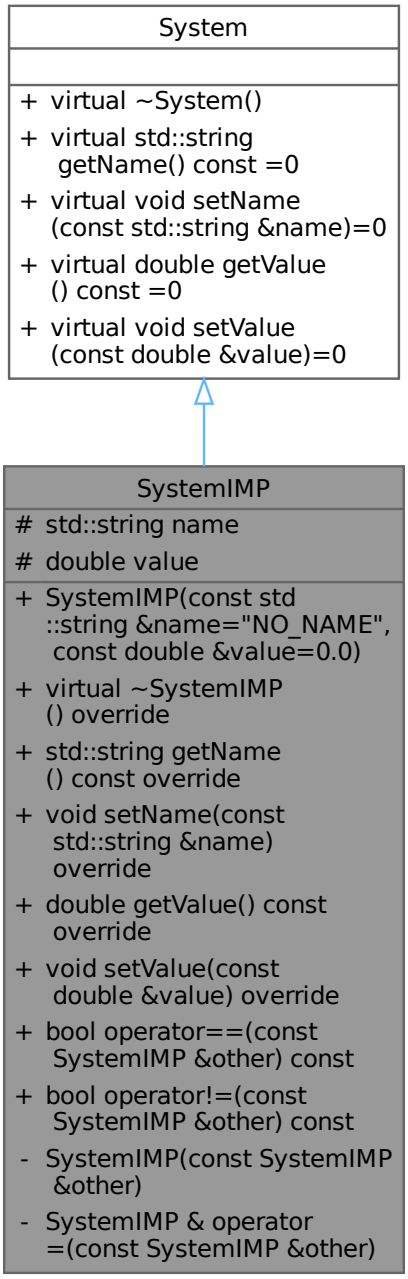
The documentation for this class was generated from the following file:

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

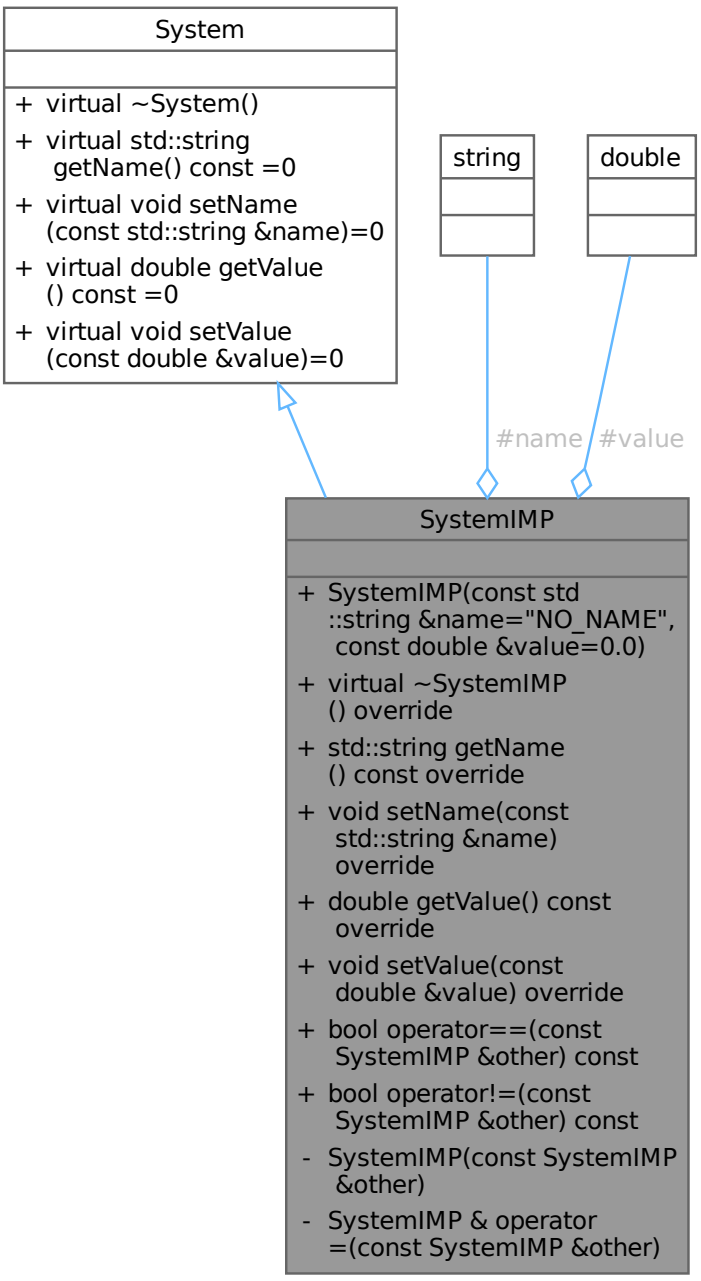
4.11 SystemIMP Class Reference

```
#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 () override

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

This method returns the name of a system.

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

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

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

  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.

4.11.1 Constructor & Destructor Documentation

4.11.1.1 SystemIMP() [1/2]

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

Construct a new System by a obj.

Parameters

other	System obj

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

4.11.1.2 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

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

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

4.11.1.3 ~SystemIMP()

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

This destructor is a virtual destructor of the Class.

```
00009 {};
```

4.11.2 Member Function Documentation

4.11.2.1 getName()

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

This method returns the name of a system.

Returns

a string containing the name is returned

Implements System.

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

References name.

4.11.2.2 getValue()

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

This method returns the value of a system.

Returns

a double containing the value is returned

Implements System.

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

References value.

4.11.2.3 operator"!=()

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

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

Parameters

other	system obj to be compare must be passed

Returns

A bool is returned, 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.

4.11.2.4 operator=()

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

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

Parameters

other	system obj to be cloned must be passed

Returns

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

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

References name, and value.

4.11.2.5 operator==()

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

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

Parameters

other	system obj to be compare must be passed

Returns

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

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

References name, and value.

4.11.2.6 setName()

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

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

Parameters

name	string must be passed to the method

Implements System.

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

References name.

4.11.2.7 setValue()

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

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

Parameters

value	double must be passed to the method

Implements System.

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

References value.

4.11.3 Member Data Documentation

4.11.3.1 name

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

Name string attribute.

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

4.11.3.2 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

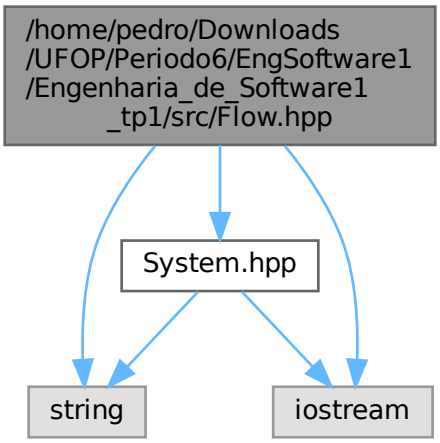
Chapter 5

File Documentation

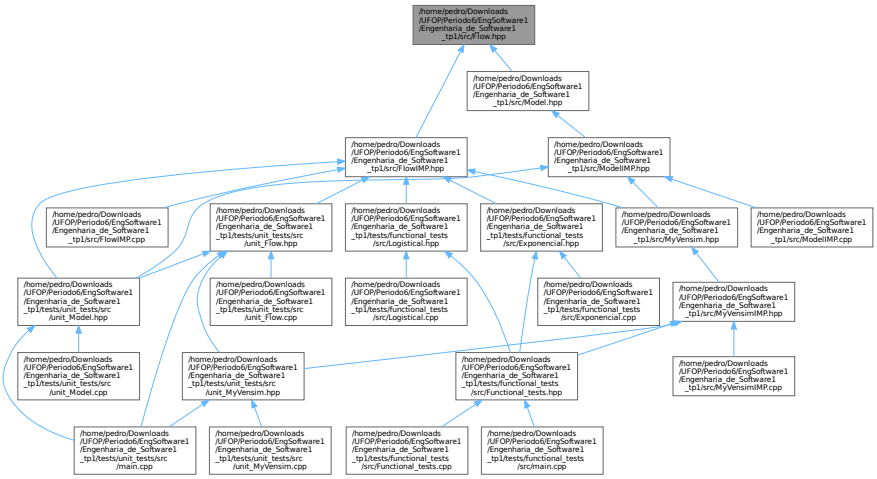
5.1 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Flow.hpp File Reference

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

Include dependency graph for Flow.hpp:



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



Classes

- class Flow

5.2 Flow.hpp

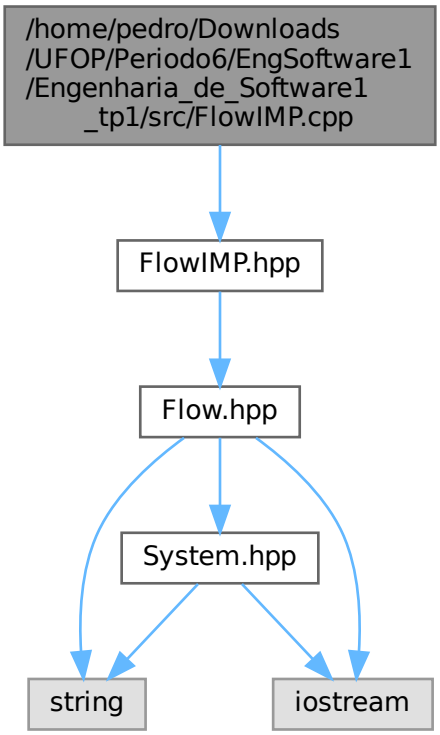
Go to the documentation of this file.

```
00001 /*************************************************
00002  * @file Flow.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the flow Interface
00005 *************************************************/
00006
00007 #ifndef FLOW_HPP
00008 #define FLOW_HPP
00009
00010 #include "System.hpp"
00011 #include <string>
00012 #include <iostream>
00013
00014 /*****************************************************************************************
00015  *@brief The Flow Interface is the Interface that defines the methods to be implemented
00016 *****************************************************************************************/
00017
00018 #include <string>
00019 #include <iostream>
00020
00021 class Flow{
00022     public:
00023         //Destructor
00027         virtual ~Flow() {};
00028
00029         //Geters e seters
00030         //Name
00035         virtual std::string getName() const = 0;
00040         virtual void setName(const std::string& name) = 0;
00041         //Source
00046         virtual System* getSource() const = 0;
00051         virtual void setSource(System* source) = 0;
00052         //Target
00057         virtual System* getTarget() const = 0;
00062         virtual void setTarget(System* target) = 0;
00063
00064         //Metodos
00069         virtual double execute() = 0;
00070 };
00071
00072
00073 #endif
```

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

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

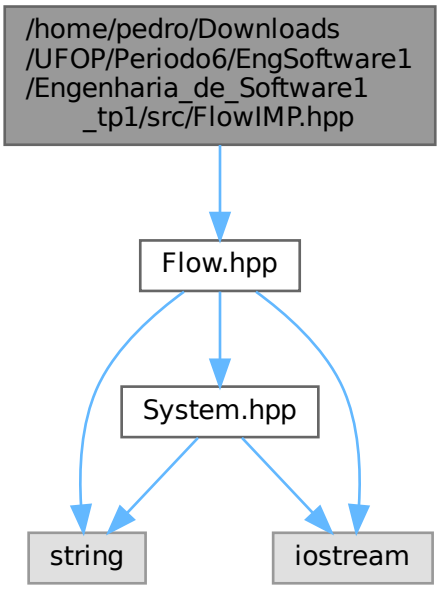
Include dependency graph for FlowIMP.cpp:



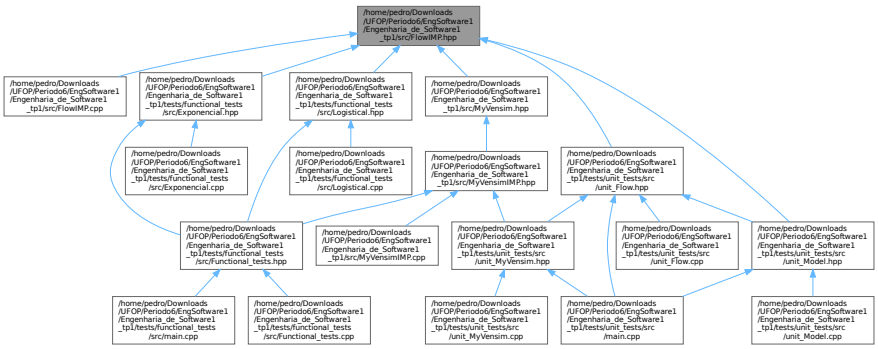
5.4 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/FlowIMP.hpp File Reference

```
#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

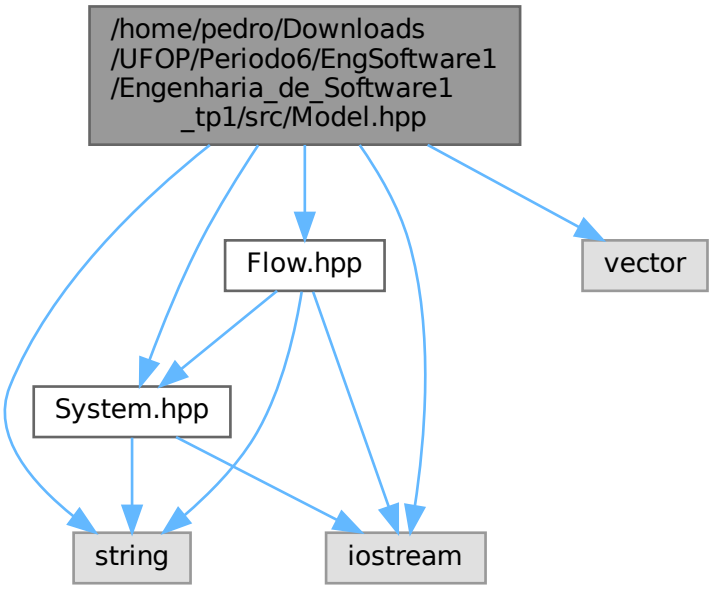
Go to the documentation of this file.

```
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() override;
00036
00037         //Geters e seters
00038         //Name
00043         std::string getName() const override;
00048         void setName(const std::string& name) override;
00049         //Source
00054         System* getSource() const override;
00059         void setSource(System* source) override;
00060         //Target
00065         System* getTarget() const override;
00070         void setTarget(System* target) override;
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
```

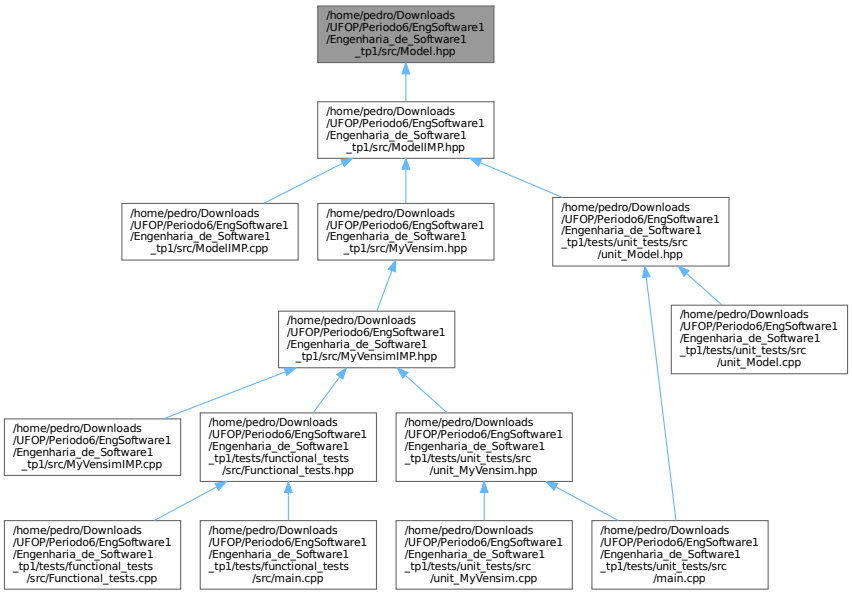
5.6 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/Model.hpp File Reference

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

Include dependency graph for Model.hpp:



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



Classes

- class Model

5.7 Model.hpp

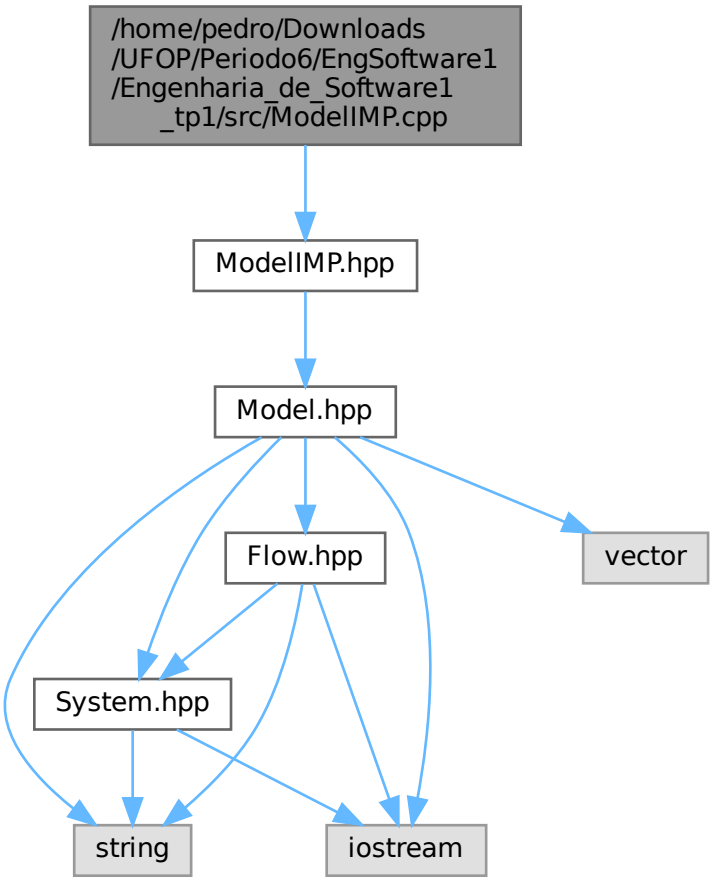
Go to the documentation of this file.

```
00001 /**************************************************
00002  * @file Model.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the model interface
00005 **************************************************/
00006
00007 #ifndef MODEL_HPP
00008 #define MODEL_HPP
00009
00010 #include "System.hpp"
00011 #include "Flow.hpp"
00012
00013 #include <string>
00014 #include <iostream>
00015 #include <vector>
00016
00017 /*****************************************************************************************
00018  *@brief he Model Interface is the Interface that defines the methods to be implemented
00019 *****************************************************************************************/
00020
00021 class Model{
00022     public:
00023         //Iteradores
00027         typedef std::vector<System*>::iterator systemIterator;
00031         typedef std::vector<Flow*>::iterator flowIterator;
00032
00033         //Destructor
00037         virtual ~Model() {};
00038
00039         //Geters e seters
00040         //Name
00045         virtual std::string getName() const = 0;
00050         virtual void setName(const std::string& name) = 0;
00051         //Vector
00056         virtual std::vector<System*> getSystems() const = 0;
00061         virtual std::vector<Flow*> getFlows() const = 0;
00066         virtual void setSystems(const std::vector<System*> systems) = 0;
00071         virtual void setFlows(const std::vector<Flow*> flows) = 0;
00072         //Time
00077         virtual int getStartTime() const = 0;
00082         virtual int getEndTime() const = 0;
00087         virtual void setStartTime(const int& startTime) = 0;
00092         virtual void setEndTime(const int& endTime) = 0;
00098         virtual void setTime(const int& startTime, const int& endTime) = 0;
00099
00100         //Metodos
00101         //add
00106         virtual void add(System* system) = 0;
00111         virtual void add(Flow* flow) = 0;
00112         //remove
00118         virtual bool rmv(const System* system) = 0;
00124         virtual bool rmv(const Flow* flow) = 0;
00125         //Others
00130         virtual bool run() = 0;
00131
00132 };
00133
00134 #endif
```

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

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

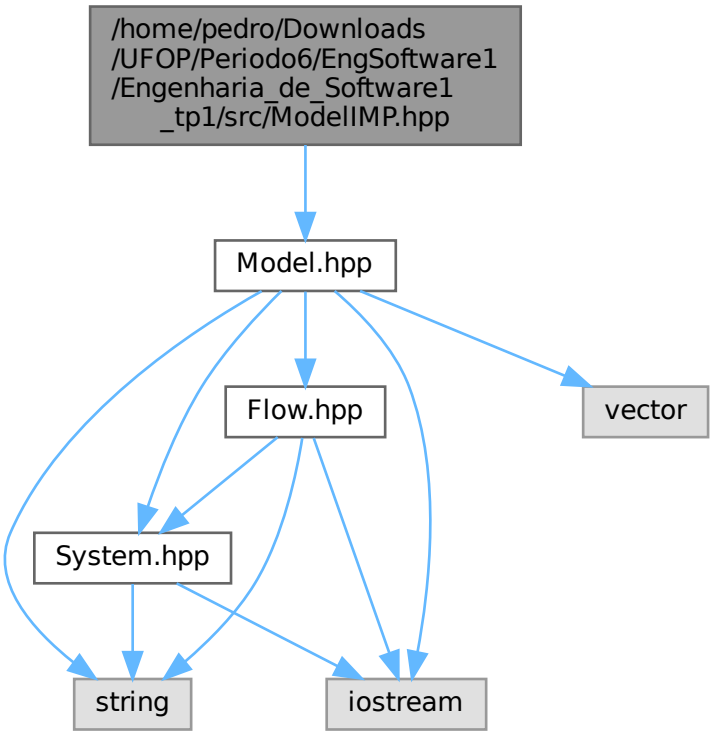
Include dependency graph for ModelIMP.cpp:



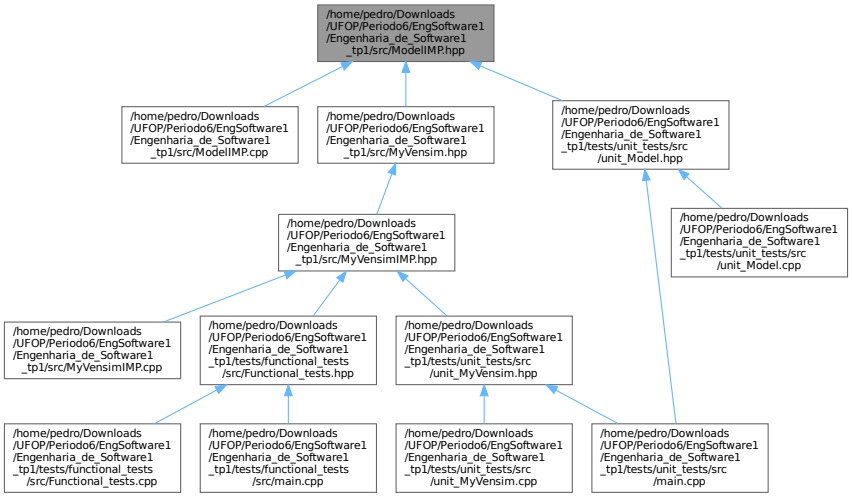
5.9 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/ModelIMP.hpp File Reference

```
#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.

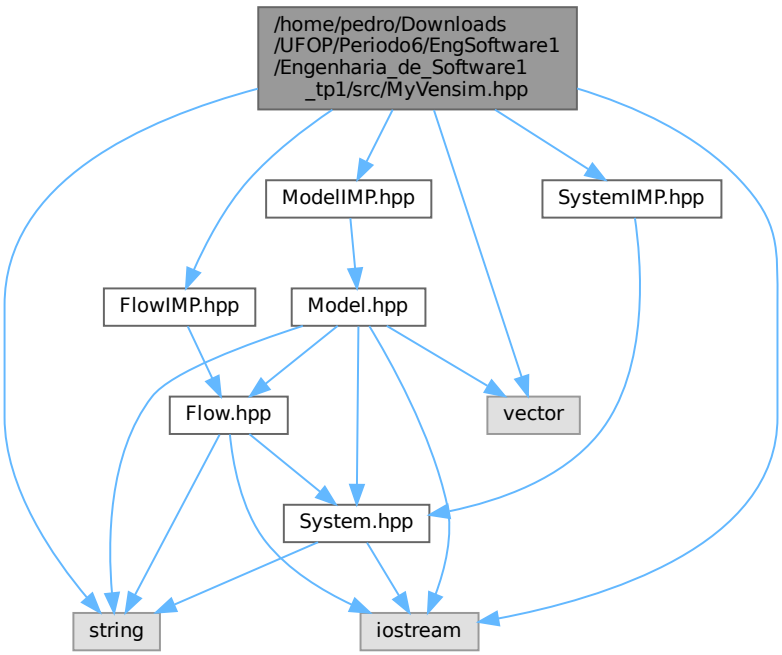
```
00001 /*******************************************************
00002  * @file ModelIMP.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the model implementation
00005 *******************************************************/
00006
00007 #ifndef MODELIMP_HPP
00008 #define MODELIMP_HPP
00009
00010 #include "Model.hpp"
00011
00012 /*****************************************************************************
00013  *@brief The Model implementation defines the attributes and implements the methods
00014 *****************************************************************************/
00015
00016 class ModelIMP : public Model{
00017     private:
00023         ModelIMP& operator=(const ModelIMP& other); // Operador de atribuição
00028         ModelIMP(const ModelIMP& other); //Copia outro flow
00029
00030     protected:
00031         std::string name;
00032         std::vector<System*> systems;
00033         std::vector<Flow*> flows;
00034         int startTime;
00035         int endTime;
00037     public:
00038         //Contructors
00045         ModelIMP(const std::string& name = "NO_NAME", const int& startTime = 0, const int& endTime =
      1);
00046
00047         //Destructor
00051         virtual ~ModelIMP() override;
00052
00053         //Geters e seters
00054         //Name
00059         std::string getName() const override;
00064         void setName(const std::string& name) override;
```

```
00065         //Vector
00070         std::vector<System*> getSystems() const override;
00075         std::vector<Flow*> getFlows() const override;
00080         void setSystems(const std::vector<System*> systems) override;
00085         void setFlows(const std::vector<Flow*> flows) override;
00086         //Time
00091         int getStartTime() const override;
00096         int getEndTime() const override;
00101         void setStartTime(const int& startTime) override;
00106         void setEndTime(const int& endTime) override;
00112         void setTime(const int& startTime, const int& endTime) override;
00113
00114         //Metodos
00115         //add
00120         void add(System* system) override;
00125         void add(Flow* flow) override;
00126         //remove
00132         bool rmv(const System* system) override;
00138         bool rmv(const Flow* flow) override;
00139         //Others
00144         bool run() override;
00145
00146         //Sobrecarga de operadores
00152         bool operator==(const ModelIMP& other) const; // Operador de igualdade
00158         bool operator!=(const ModelIMP& other) const; // Operador de igualdade
00159 };
00160
00161 #endif
```

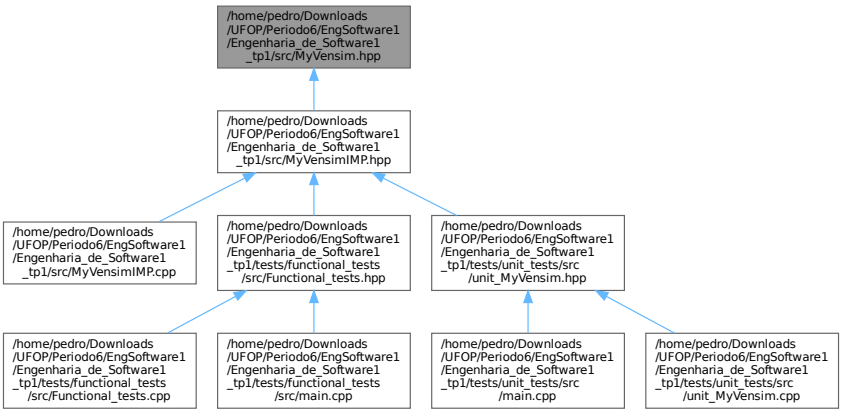
5.11 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/src/MyVensim.hpp File Reference

```
#include "ModelIMP.hpp"
#include "FlowIMP.hpp"
#include "SystemIMP.hpp"
#include <vector>
#include <iostream>
#include <string>
```

Include dependency graph for MyVensim.hpp:



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



Classes

- class MyVensim

5.12 MyVensim.hpp

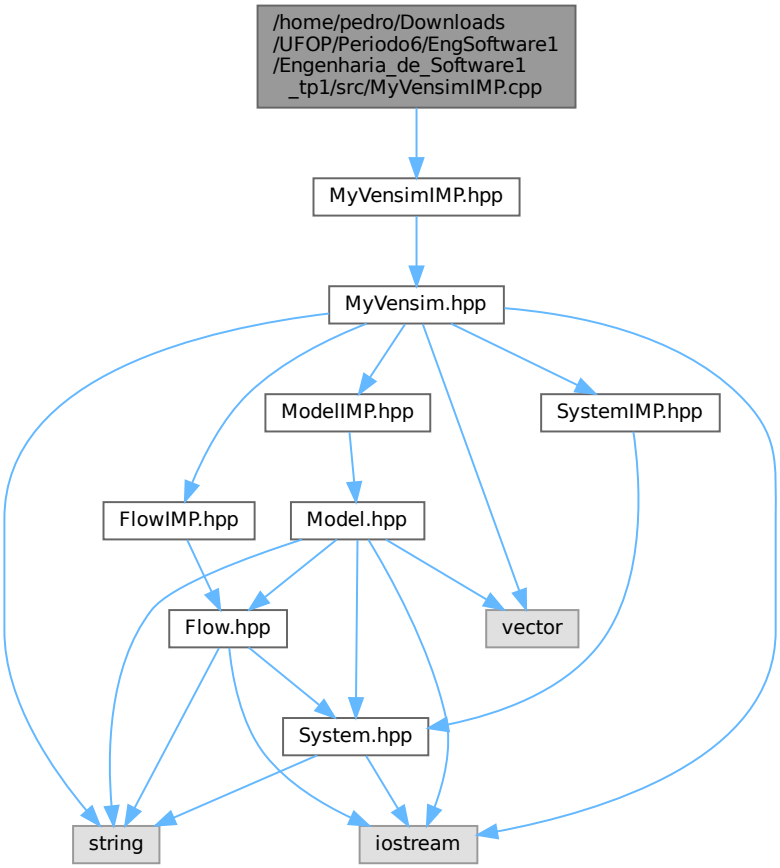
Go to the documentation of this file.

```
00001 /*****************************************************
00002  * @file MyVensim.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the factory interface
00005 *****************************************************/
00006
00007 #ifndef MYVENSIM_HPP
00008 #define MYVENSIM_HPP
00009
00010 #include "ModelIMP.hpp"
00011 #include "FlowIMP.hpp"
00012 #include "SystemIMP.hpp"
00013
00014 #include <vector>
00015 #include <iostream>
00016 #include <string>
00017
00018 /*********************************************************************************************
00019  *@brief  he MyVensim Interface is the Interface that defines the methods to be implemented
00020 *********************************************************************************************/
00021
00022 class MyVensim{
00023     public:
00024         //Iteradores
00028         typedef std::vector<System*>::iterator systemIterator;
00032         typedef std::vector<Flow*>::iterator flowIterator;
00036         typedef std::vector<Model*>::iterator modelIterator;
00037
00038         //Destrutor
00042         virtual ~MyVensim() {}
00043
00050         virtual System* createSystem(const std::string& name = "NO_NAME", const double& value = 0.0) =
      0;
00051
00059         template <typename FlowType>
00060         Flow* createFlow(const std::string name = "NONAME", System* source = nullptr, System* destiny
      = nullptr){
00061             Flow* f = new FlowType(name, source, destiny);
00062             add(f);
00063             return f;
00064         }
00065
00073         virtual Model* createModel(const std::string& name = "NO_NAME", const int& startTime = 0,
      const int& endTime = 1) = 0;
00074
00075          //add
00080         virtual void add(System* system) = 0;
00085         virtual void add(Flow* flow) = 0;
00090         virtual void add(Model* model) = 0;
00091         //remove
00097         virtual bool rmv(const System* system) = 0;
00103         virtual bool rmv(const Flow* flow) = 0;
00109         virtual bool rmv(const Model* model) = 0;
00110
00111         //Geters e seters
00112         //Name
00117         virtual std::string getName() const = 0;
00122         virtual void setName(const std::string& name) = 0;
00123         //Vector
00128         virtual std::vector<System*> getSystems() const = 0;
00133         virtual std::vector<Flow*> getFlows() const = 0;
00138         virtual std::vector<Model*> getModels() const = 0;
00143         virtual void setSystems(const std::vector<System*> systems) = 0;
00148         virtual void setFlows(const std::vector<Flow*> flows) = 0;
00153         virtual void setModels(const std::vector<Model*> models) = 0;
00154
00159         virtual bool run() = 0;
00160
00161 };
00162
00163 #endif
```

5.13 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/src/MyVensimIMP.cpp File Reference

```
#include "MyVensimIMP.hpp"
```

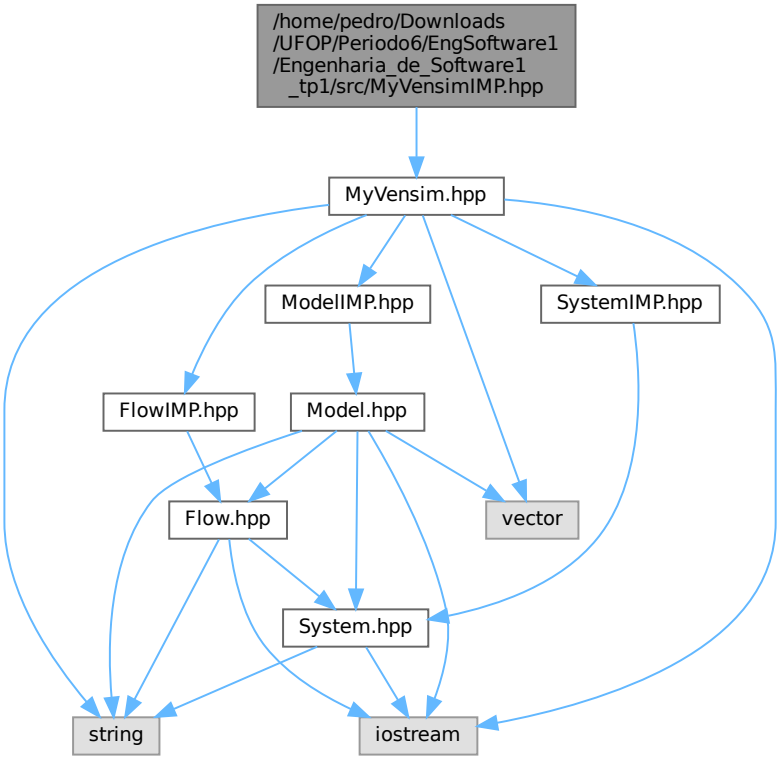
Include dependency graph for MyVensimIMP.cpp:



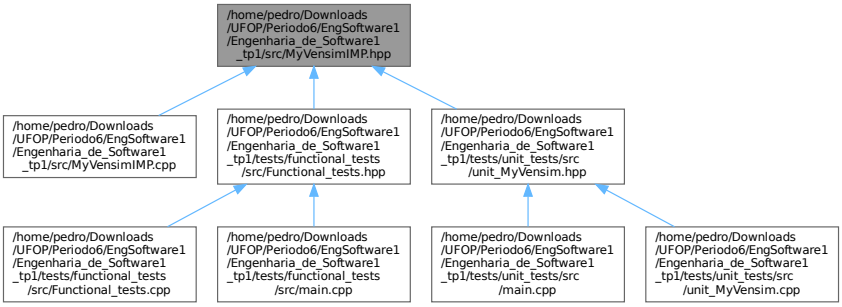
5.14 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/src/MyVensimIMP.hpp File Reference

```
#include "MyVensim.hpp"
```

Include dependency graph for MyVensimIMP.hpp:



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



Classes

- class MyVensimIMP

5.15 MyVensimIMP.hpp

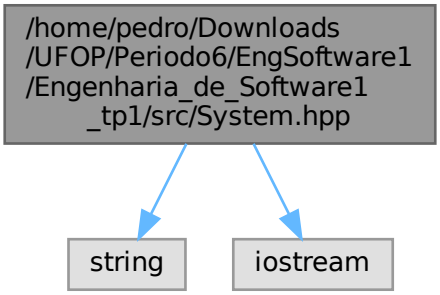
Go to the documentation of this file.

```
00001 /*******************************************************
00002  * @file MyVensimIMP.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the factory implementation
00005 *******************************************************/
00006
00007 #ifndef MYVENSIMIMP_HPP
00008 #define MYVENSIMIMP_HPP
00009
00010 #include "MyVensim.hpp"
00011
00012 /*****************************************************************************
00013  *@brief The MyVensin implementation defines the attributes and implements the methods
00014 *****************************************************************************/
00015
00016 class MyVensimIMP : public MyVensim{
00017     private:
00023         MyVensimIMP& operator=(const MyVensimIMP& other); // Operador de atribuição
00028         MyVensimIMP(const MyVensimIMP& other); //Copia outro MyVensim
00029
00030     protected:
00031         std::string name;
00032         std::vector<Model*> models;
00033         std::vector<System*> systems;
00034         std::vector<Flow*> flows;
00036     public:
00037
00038         //Constructor
00043         MyVensimIMP(const std::string& name = "NO_NAME");
00044
00045         //Destructor
00049         virtual ~MyVensimIMP() override;
00050
00057         System* createSystem(const std::string& name = "NO_NAME", const double& value = 0.0) override;
00058
00066         Model* createModel(const std::string& name = "NO_NAME", const int& startTime = 0, const int&
      endTime = 1) override;
00067
00068          //add
00073         void add(System* system) override;
00078         void add(Flow* flow) override;
00083         void add(Model* model) override;
00084         //remove
00090         bool rmv(const System* system) override;
00096         bool rmv(const Flow* flow) override;
00102         bool rmv(const Model* model) override;
00103
00104         //Geters e seters
00105         //Name
00110         std::string getName() const;
00115         void setName(const std::string& name);
00116         //Vector
00121         std::vector<System*> getSystems() const;
00126         std::vector<Flow*> getFlows() const;
00131         std::vector<Model*> getModels() const;
00136         void setSystems(const std::vector<System*> systems);
00141         void setFlows(const std::vector<Flow*> flows);
00146         void setModels(const std::vector<Model*> models);
00147
00152         bool run() override;
00153
00154         //Sobrecarga de operadores
00160         bool operator==(const MyVensimIMP& other) const; // Operador de igualdade
00166         bool operator!=(const MyVensimIMP& other) const; // Operador de igualdade
00167 };
00168
00169 #endif
```

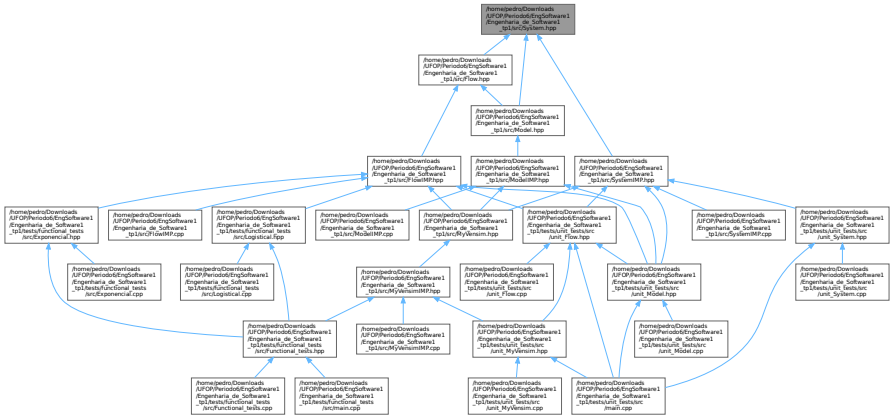
5.16 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/src/System.hpp File Reference

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

Include dependency graph for System.hpp:



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



Classes

- class System

5.17 System.hpp

Go to the documentation of this file.

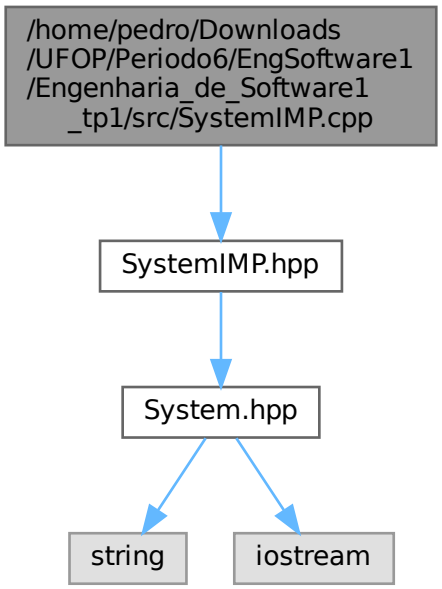
```
00001 /*************************************************
00002  * @file System.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the System interface
00005 *************************************************/
00006
00007 #ifndef SYSTEM_HPP
00008 #define SYSTEM_HPP
00009
00010 #include <string>
00011 #include <iostream>
00012
00013 /*****************************************************************************************
```

```
00014  *@brief The System Interface is the Interface that defines the methods to be implemented
00015 ********************************************************************************************/
00016
00017 class System{
00018     public:
00019         //Destructors
00023         virtual ~System() {};
00024
00025         //Geters e seters
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
```

5.18 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/src/SystemIMP.cpp File Reference

```
#include "SystemIMP.hpp"
```

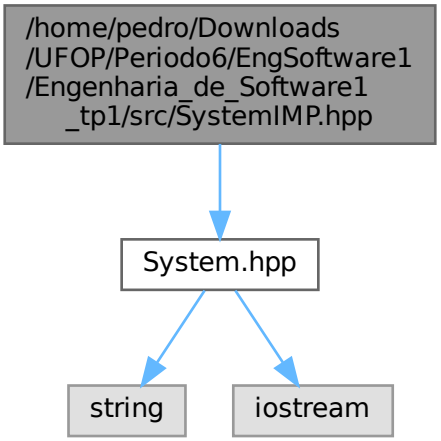
Include dependency graph for SystemIMP.cpp:



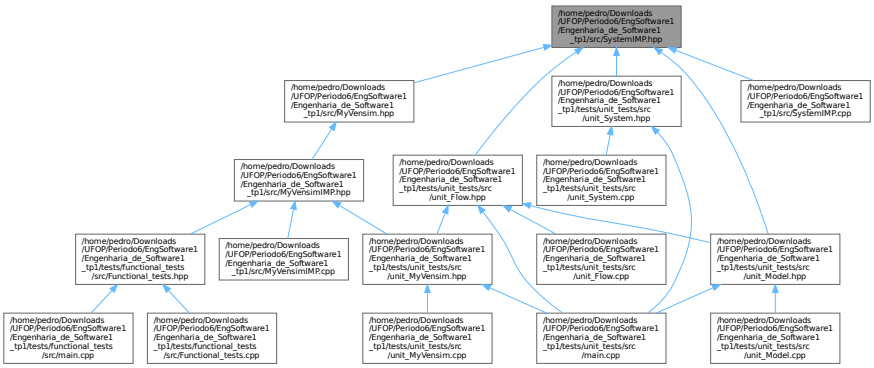
5.19 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/src/SystemIMP.hpp File Reference

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

Include dependency graph for SystemIMP.hpp:



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



Classes

- class SystemIMP

5.20 SystemIMP.hpp

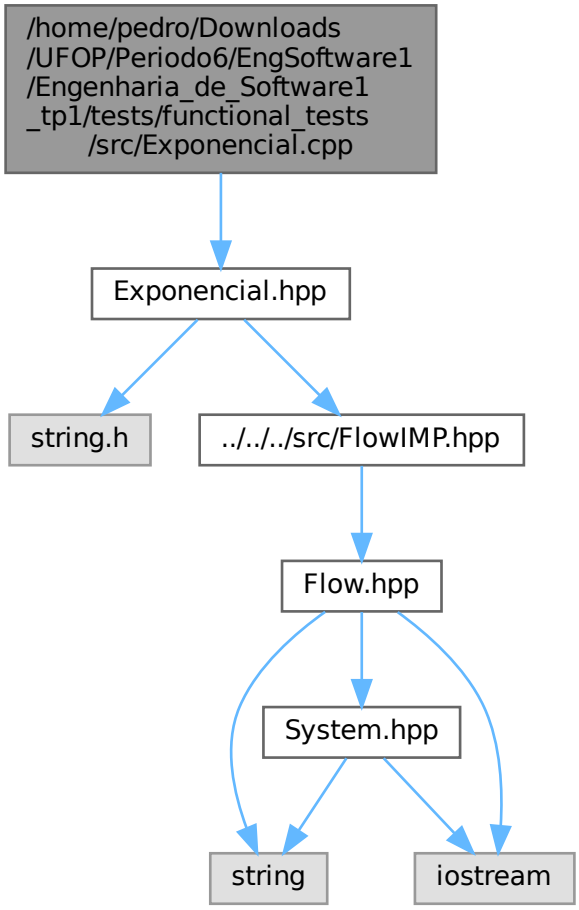
Go to the documentation of this file.

```
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         //Contructors
00042         SystemIMP(const std::string& name = "NO_NAME", const double& value = 0.0);
00043
00047         //Destructors
00048         virtual ~SystemIMP() override;
00049
00050         //Geters e seters
00051         //Nome
00056         std::string getName() const override;
00061         void setName(const std::string& name) override;
00062         //Value
00067         double getValue() const override;
00072         void setValue(const double& value) override;
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
```

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

```
#include "Exponencial.hpp"
```

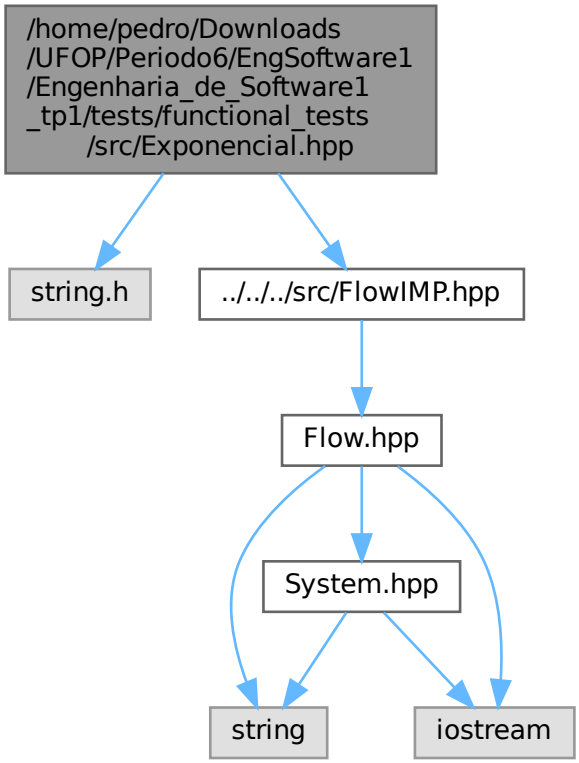
Include dependency graph for Exponencial.cpp:



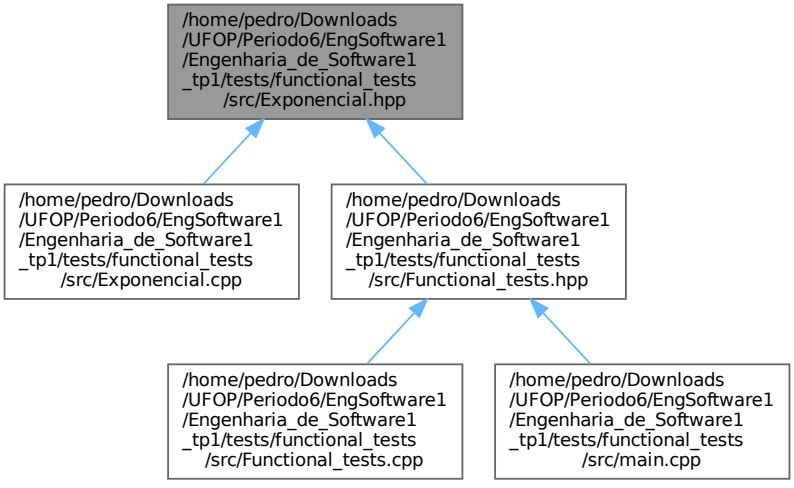
5.22 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/functional_tests/src/↩ Exponencial.hpp File Reference

```
#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

5.23 Exponencial.hpp

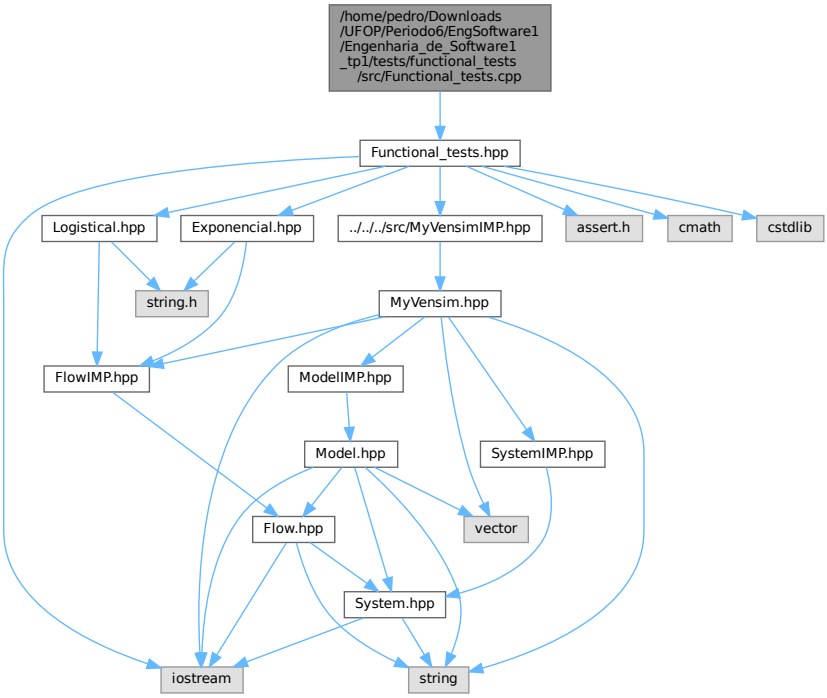
Go to the documentation of this file.

```
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
      one system to another
00015
      *****************************************************************************************************************/
00016
00017 class Exponencial : public FlowIMP{
00018     private:
00023         Exponencial(const Exponencial& other);
00024
00025     public:
00026         //Contructor
00033         Exponencial(const std::string& name = "NO_NAME", System* source = nullptr, System* target =
      nullptr);
00034
00035         //Destructor
00039         virtual ~Exponencial() override;
00040
00041         //Metodos
00046         virtual double execute() override;
00047 };
00048
00049 #endif
```

5.24 /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.

5.24.1 Function Documentation

5.24.1.1 Complex_test_run()

```
void Complex_test_run ( )
```

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

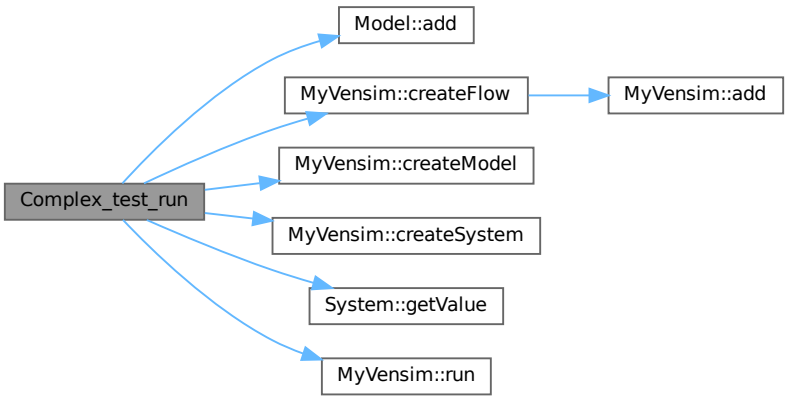
```
00062                       {
00063     std::cout « "  Complex functional test" « std::endl;
00064
```

```
00065     MyVensim* mV = new MyVensimIMP("mV");
00066     Model* model = mV->createModel("Model", 0, 100);
00067     System* q1 = mV->createSystem("q1", 100.0);
00068     System* q2 = mV->createSystem("q2", 0.0);
00069     System* q3 = mV->createSystem("q3", 100.0);
00070     System* q4 = mV->createSystem("q4", 0.0);
00071     System* q5 = mV->createSystem("q5", 0.0);
00072     Flow* f = mV->createFlow<Exponencial>("f", q1, q2);
00073     Flow* t = mV->createFlow<Exponencial>("t", q2, q3);
00074     Flow* u = mV->createFlow<Exponencial>("u", q3, q4);
00075     Flow* v = mV->createFlow<Exponencial>("v", q4, q1);
00076     Flow* g = mV->createFlow<Exponencial>("g", q1, q3);
00077     Flow* r = mV->createFlow<Exponencial>("r", q2, q5);
00078
00079     model->add(q1);
00080     model->add(q2);
00081     model->add(q3);
00082     model->add(q4);
00083     model->add(q5);
00084     model->add(f);
00085     model->add(t);
00086     model->add(u);
00087     model->add(v);
00088     model->add(g);
00089     model->add(r);
00090
00091     mV->run();
00092
00093     assert(fabs((round((q1->getValue() * 10000)) - 10000 * 31.8513)) < 0.0001);
00094     assert(fabs((round((q2->getValue() * 10000)) - 10000 * 18.4003)) < 0.0001);
00095     assert(fabs((round((q3->getValue() * 10000)) - 10000 * 77.1143)) < 0.0001);
00096     assert(fabs((round((q4->getValue() * 10000)) - 10000 * 56.1728)) < 0.0001);
00097     assert(fabs((round((q5->getValue() * 10000)) - 10000 * 16.4612)) < 0.0001);
00098
00099     delete model;
00100     delete q1;
00101     delete q2;
00102     delete q3;
00103     delete q4;
00104     delete q5;
00105     delete f;
00106     delete t;
00107     delete u;
00108     delete v;
00109     delete g;
00110     delete r;
00111     delete mV;
00112
00113     std::cout « "  Complex functional test passed" « std::endl;
00114 }
```

References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

5.24.1.2 exponencial_test_run()

```
void exponencial_test_run ( )
```

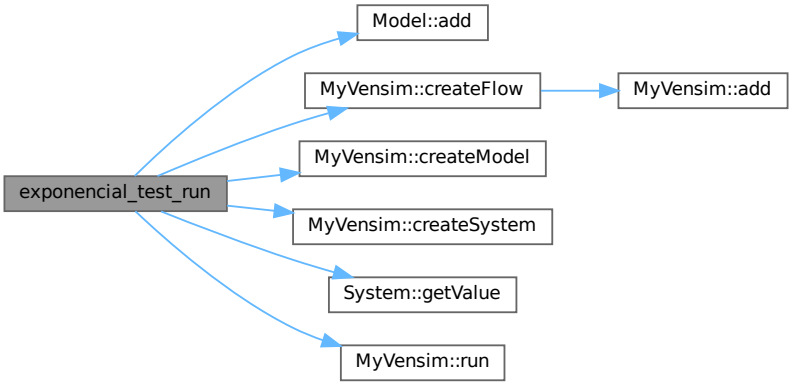
This function performs the exponential functional test.

```
00003                           {
00004     std::cout « "  Exponencial functional test" « std::endl;
00005
00006     MyVensim* mV = new MyVensimIMP("mV");
00007     System* pop1 = mV->createSystem("pop1", 100.0);
00008     System* pop2 = mV->createSystem("pop2", 0.0);
00009     Flow* exp = mV->createFlow<Exponencial>("exp", pop1, pop2);
00010     Model* exponencial = mV->createModel("Exponencial", 0, 100);
00011
00012     //Add os systems e flows ao modelo
00013     exponencial->add(pop1);
00014     exponencial->add(pop2);
00015     exponencial->add(exp);
00016
00017     //Roda o modelo
00018     mV->run();
00019
00020     assert(fabs((round(pop1->getValue() * 10000) - 10000 * 36.6032)) < 0.0001);
00021     assert(fabs((round(pop2->getValue() * 10000) - 10000 * 63.3968)) < 0.0001);
00022
00023     delete pop1;
00024     delete pop2;
00025     delete exp;
00026     delete exponencial;
00027     delete mV;
00028
00029
00030     std::cout « "  Exponencial functional test passed\n" « std::endl;
00031 }
```

References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

5.24.1.3 logistical_test_run()

```
void logistical_test_run ( )
```

This function performs the logistic test.

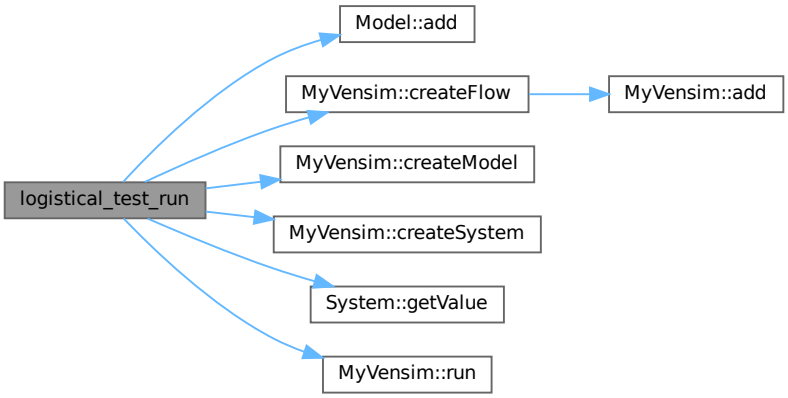
```
00033                           {
00034     std::cout « "  Logistical functional test" « std::endl;
00035
00036     MyVensim* mV = new MyVensimIMP("mV");
00037     System* p1 = mV->createSystem("p1", 100.0);
00038     System* p2 = mV->createSystem("p2", 10.0);
00039     Flow* log = mV->createFlow<Logistical>("log", p1, p2);
00040     Model* logistical = mV->createModel("Logistical", 0, 100);
00041
00042     //Add os systems e flows ao modelo
00043     logistical->add(p1);
00044     logistical->add(p2);
00045     logistical->add(log);
00046
00047     //Roda o modelo
00048     mV->run();
00049
00050     assert(fabs(round(p1->getValue() * 10000) - 10000 * 88.2167) < 0.0001);
00051     assert(fabs(round(p2->getValue() * 10000) - 10000 * 21.7833) < 0.0001);
00052
00053     delete logistical;
00054     delete log;
00055     delete p1;
00056     delete p2;
00057     delete mV;
00058
```

```
00059     std::cout « "  Logistical functional test passed\n" « std::endl;
00060 }
```

References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by main().

Here is the call graph for this function:



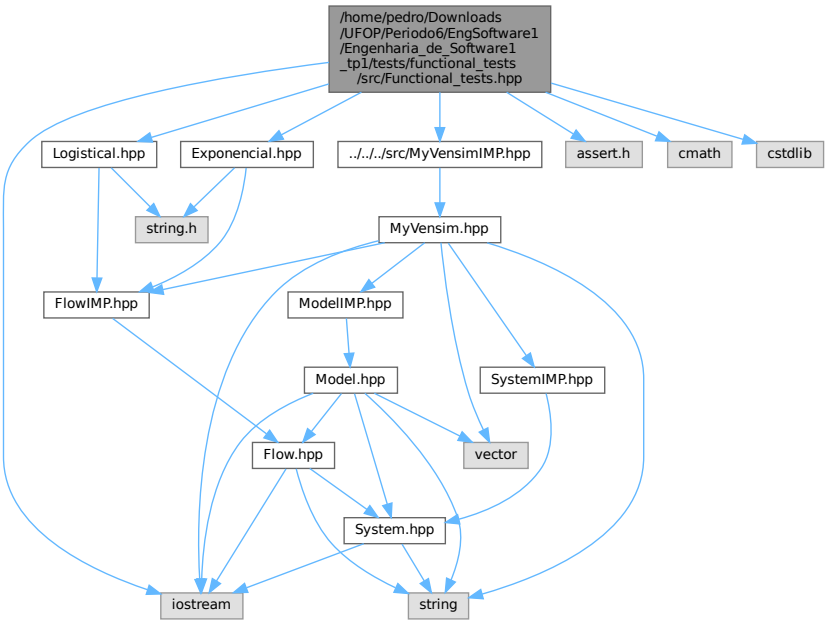
Here is the caller graph for this function:

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

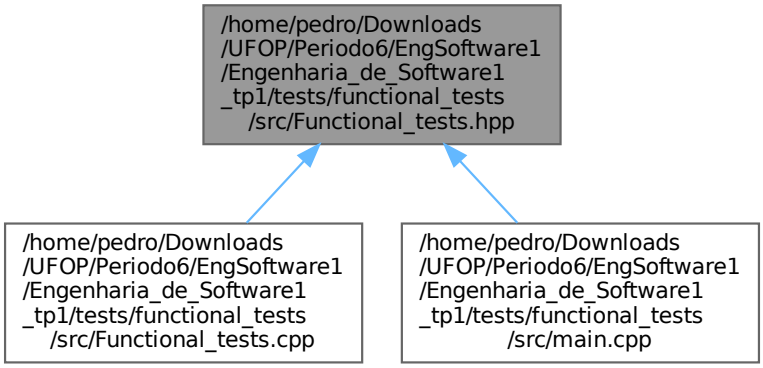
```
#include "../../../src/MyVensimIMP.hpp"
#include "Exponencial.hpp"
#include "Logistical.hpp"
#include <assert.h>
#include <cmath>
#include <iostream>
```

```
#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.

5.25.1 Function Documentation

5.25.1.1 Complex_test_run()

```
void Complex_test_run ( )
```

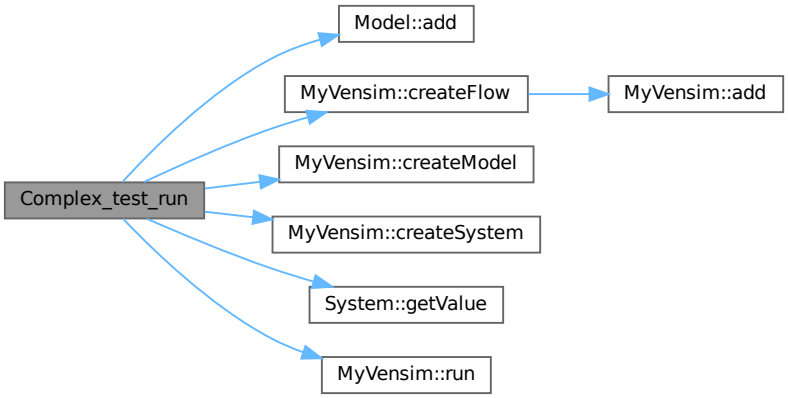
This function runs the "complex" test, which has multiple systems and flows.

```
00062                       {
00063     std::cout « "  Complex functional test" « std::endl;
00064
00065     MyVensim* mV = new MyVensimIMP("mV");
00066     Model* model = mV->createModel("Model", 0, 100);
00067     System* q1 = mV->createSystem("q1", 100.0);
00068     System* q2 = mV->createSystem("q2", 0.0);
00069     System* q3 = mV->createSystem("q3", 100.0);
00070     System* q4 = mV->createSystem("q4", 0.0);
00071     System* q5 = mV->createSystem("q5", 0.0);
00072     Flow* f = mV->createFlow<Exponencial>("f", q1, q2);
00073     Flow* t = mV->createFlow<Exponencial>("t", q2, q3);
00074     Flow* u = mV->createFlow<Exponencial>("u", q3, q4);
00075     Flow* v = mV->createFlow<Exponencial>("v", q4, q1);
00076     Flow* g = mV->createFlow<Exponencial>("g", q1, q3);
00077     Flow* r = mV->createFlow<Exponencial>("r", q2, q5);
00078
00079     model->add(q1);
00080     model->add(q2);
00081     model->add(q3);
00082     model->add(q4);
00083     model->add(q5);
00084     model->add(f);
00085     model->add(t);
00086     model->add(u);
00087     model->add(v);
00088     model->add(g);
00089     model->add(r);
00090
00091     mV->run();
00092
00093     assert(fabs((round((q1->getValue() * 10000)) - 10000 * 31.8513)) < 0.0001);
00094     assert(fabs((round((q2->getValue() * 10000)) - 10000 * 18.4003)) < 0.0001);
00095     assert(fabs((round((q3->getValue() * 10000)) - 10000 * 77.1143)) < 0.0001);
00096     assert(fabs((round((q4->getValue() * 10000)) - 10000 * 56.1728)) < 0.0001);
00097     assert(fabs((round((q5->getValue() * 10000)) - 10000 * 16.4612)) < 0.0001);
00098
00099     delete model;
00100     delete q1;
00101     delete q2;
00102     delete q3;
00103     delete q4;
00104     delete q5;
00105     delete f;
00106     delete t;
00107     delete u;
00108     delete v;
00109     delete g;
00110     delete r;
00111     delete mV;
00112
00113     std::cout « "  Complex functional test passed" « std::endl;
00114 }
```

References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

5.25.1.2 exponencial_test_run()

```
void exponencial_test_run ( )
```

This function performs the exponential functional test.

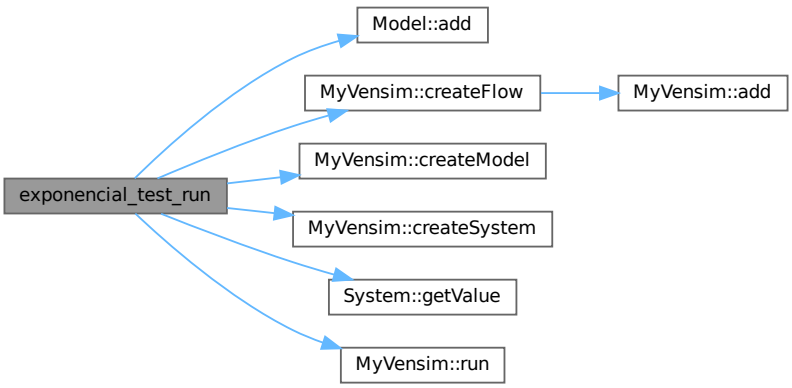
```
00003                           {
00004     std::cout « "  Exponencial functional test" « std::endl;
00005 
00006     MyVensim* mV = new MyVensimIMP("mV");
00007     System* pop1 = mV->createSystem("pop1", 100.0);
00008     System* pop2 = mV->createSystem("pop2", 0.0);
00009     Flow* exp = mV->createFlow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

```
00028
00029
00030     std::cout « "  Exponencial functional test passed\n" « std::endl;
00031 }
```

References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

5.25.1.3 logistical_test_run()

```
void logistical_test_run ( )
```

This function performs the logistic test.

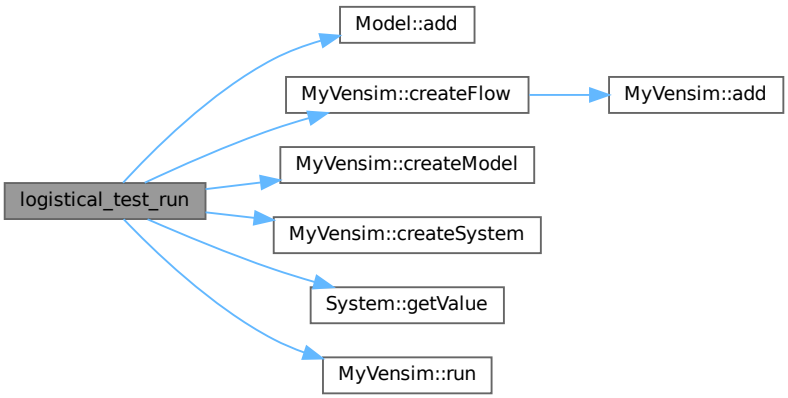
```
00033                           {
00034     std::cout « "  Logistical functional test" « std::endl;
00035
00036     MyVensim* mV = new MyVensimIMP("mV");
00037     System* p1 = mV->createSystem("p1", 100.0);
00038     System* p2 = mV->createSystem("p2", 10.0);
00039     Flow* log = mV->createFlow<Logistical>("log", p1, p2);
00040     Model* logistical = mV->createModel("Logistical", 0, 100);
00041
00042     //Add os systems e flows ao modelo
00043     logistical->add(p1);
00044     logistical->add(p2);
```

```
00045     logistical->add(log);
00046
00047     //Roda o modelo
00048     mV->run();
00049
00050     assert(fabs(round(p1->getValue() * 10000) - 10000 * 88.2167) < 0.0001);
00051     assert(fabs(round(p2->getValue() * 10000) - 10000 * 21.7833) < 0.0001);
00052
00053     delete logistical;
00054     delete log;
00055     delete p1;
00056     delete p2;
00057     delete mV;
00058
00059     std::cout « "  Logistical functional test passed\n" « std::endl;
00060 }
```

References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

5.26 Functional_tests.hpp

Go to the documentation of this file.

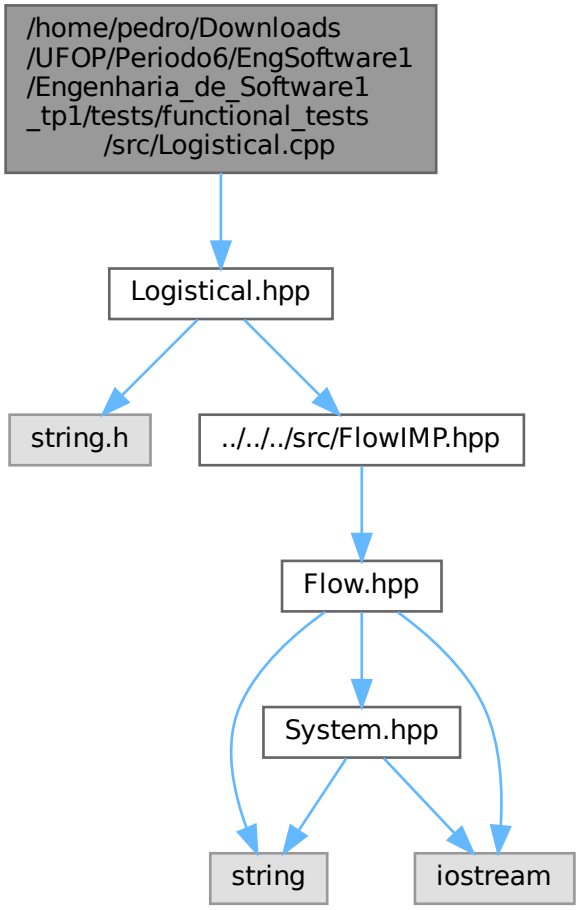
```
00001 /*************************************************************
00002  * @file Exponencial.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
```

```
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/MyVensimIMP.hpp"
00011 #include "Exponencial.hpp"
00012 #include "Logistical.hpp"
00013 #include <assert.h>
00014 #include <cmath>
00015 #include <iostream>
00016 #include <cstdlib>
00017
00018 /************************************
00019  * @brief execution of functional tests
00020 ************************************/
00021
00025 void exponencial_test_run();
00026
00030 void logistical_test_run();
00031
00035 void Complex_test_run();
00036
00037 #endif
```

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

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

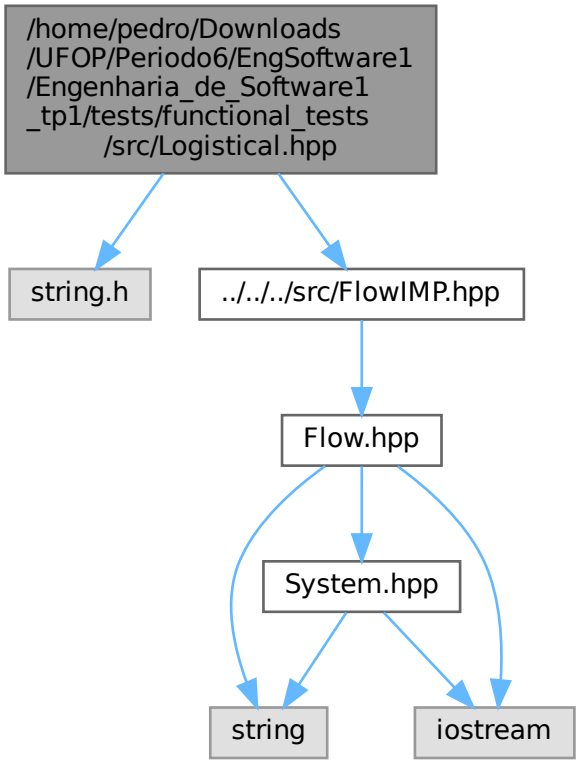
Include dependency graph for Logistical.cpp:



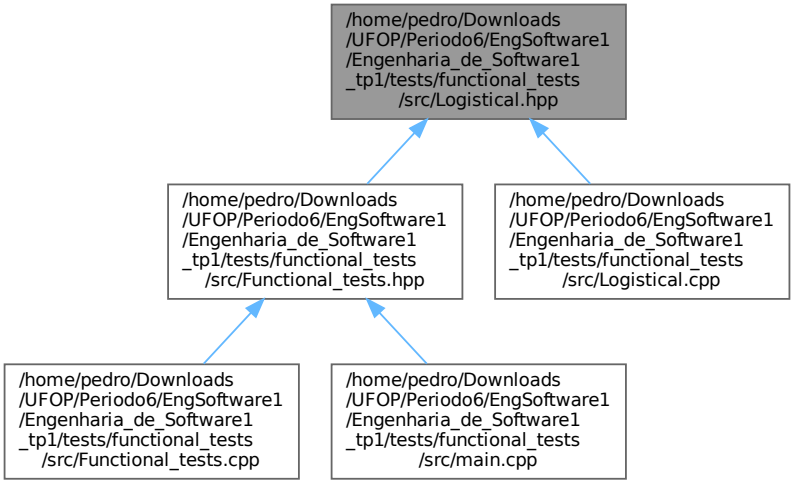
5.28 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/functional_tests/src/↩ Logistical.hpp File Reference

```
#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

5.29 Logistical.hpp

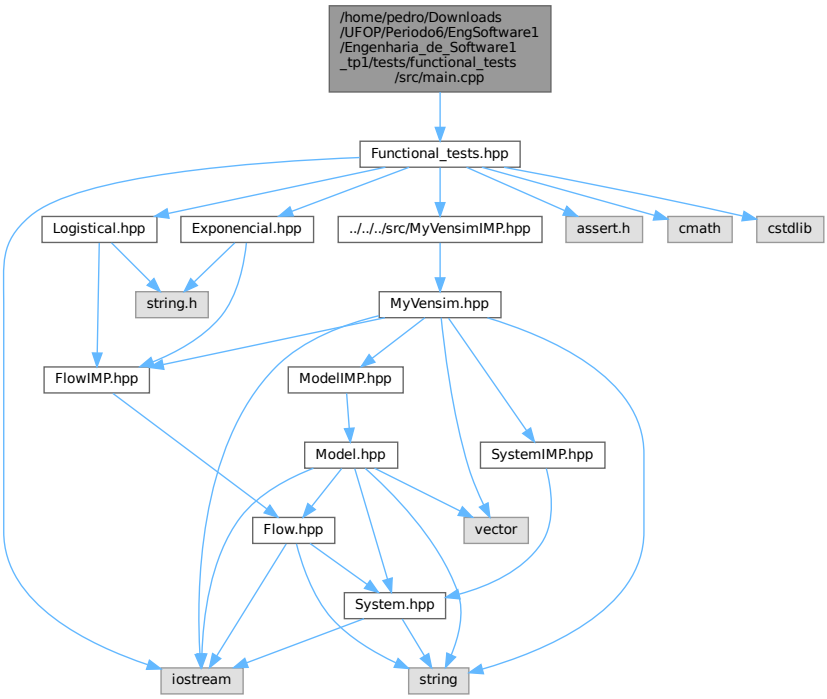
Go to the documentation of this file.

```
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
      one system to another
00015
      *****************************************************************************************************************/
00016
00017 class Logistical : public FlowIMP{
00018     private:
00023         Logistical(const Logistical& other);
00024
00025     public:
00026         //Contructor
00033         Logistical(const std::string& name = "NO_NAME", System* source = nullptr, System* target =
      nullptr);
00034
00035         //Destructor
00039         virtual ~Logistical() override;
00040
00041         //Metodos
00046         virtual double execute() override;
00047 };
00048
00049 #endif
```

5.30 /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 ()

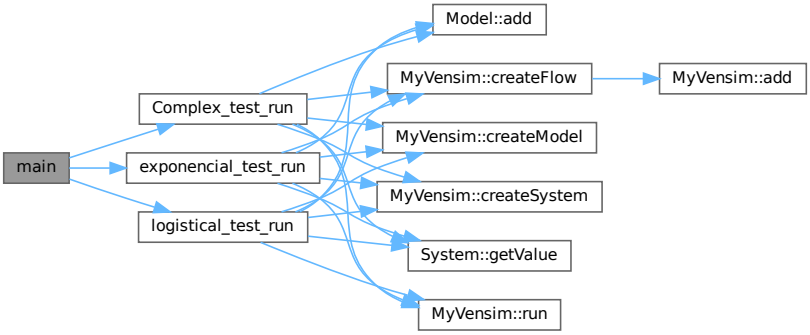
5.30.1 Function Documentation

5.30.1.1 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().

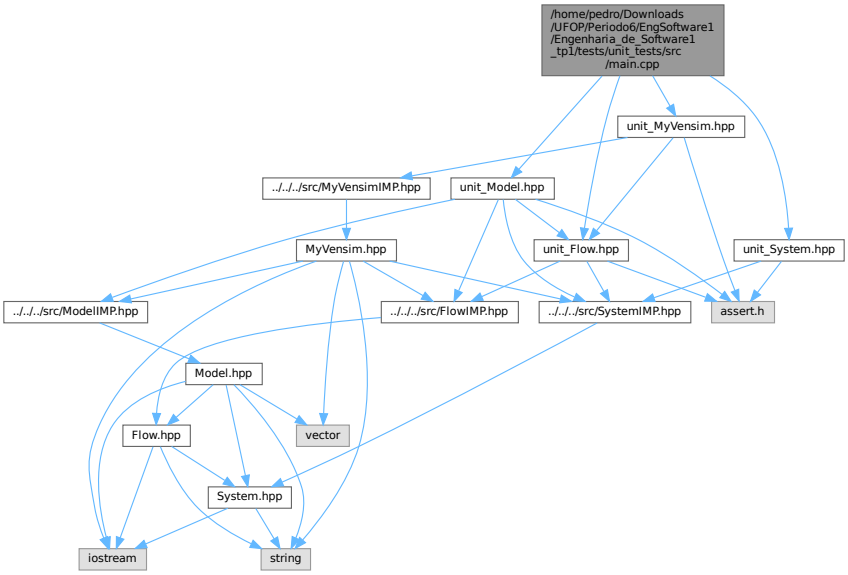
Here is the call graph for this function:



5.31 /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 "unit_MyVensim.hpp"
```

Include dependency graph for main.cpp:



Functions

- int main ()

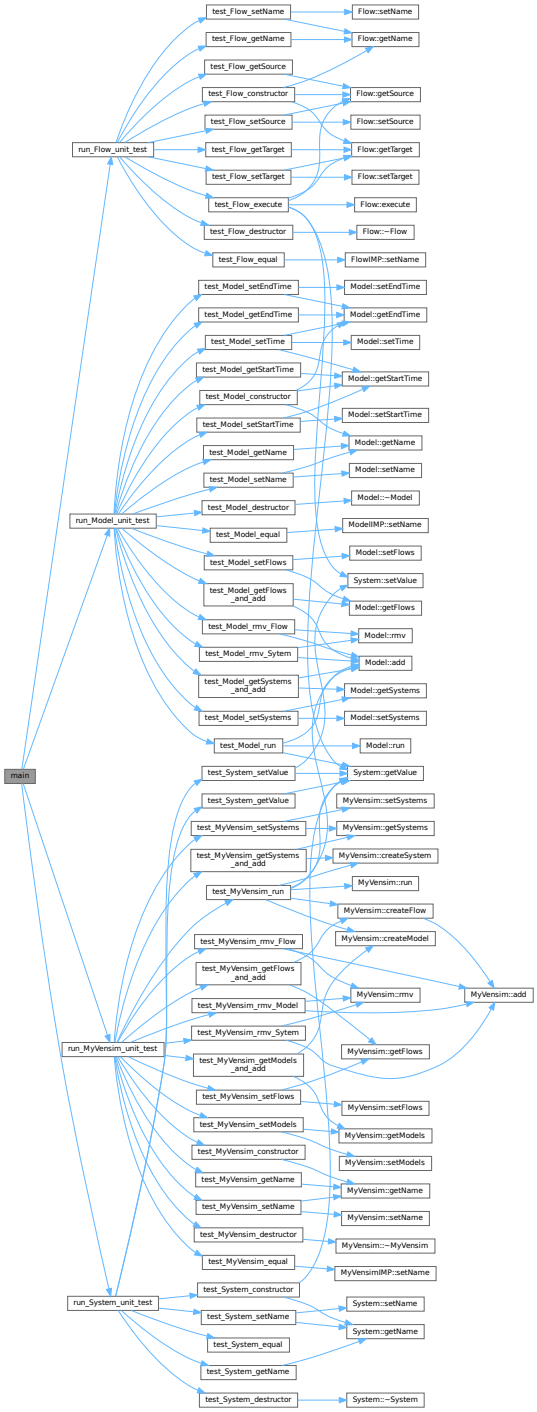
5.31.1 Function Documentation

5.31.1.1 main()

```
int main ( )
00006           {
00007     std::cout « "\nStart unit tests\n"
00008               « "*******************************************************\n";
00009     run_System_unit_test();
00010     run_Flow_unit_test();
00011     run_Model_unit_test();
00012     run_MyVensim_unit_test();
00013     std::cout « "*******************************************************\n"
00014               « "End unit tests\n\n";
00015     return 0;
00016 }
```

References run_Flow_unit_test(), run_Model_unit_test(), run_MyVensim_unit_test(), and run_System_unit_test().

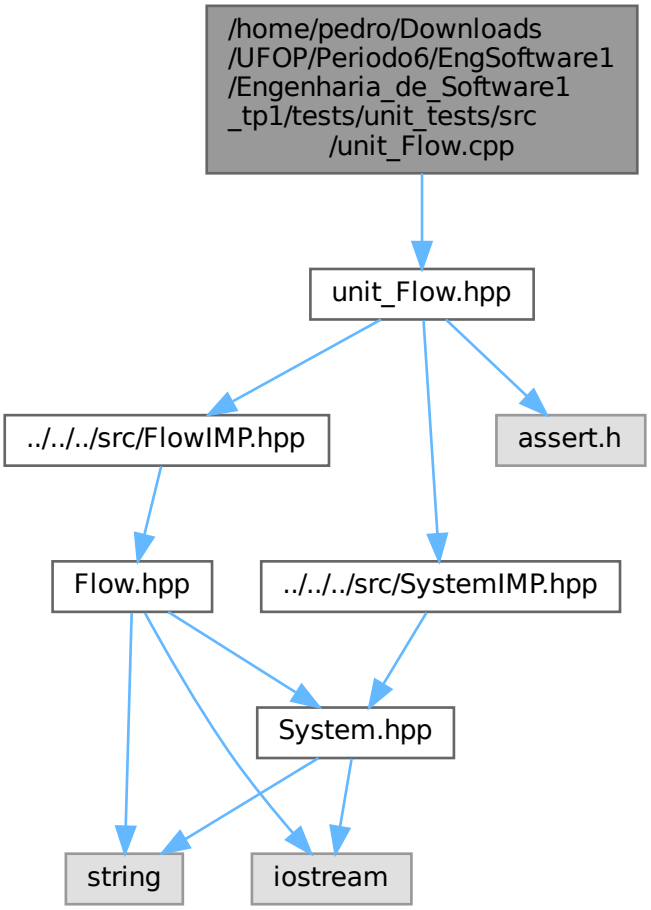
Here is the call graph for this function:



5.32 /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 comparation.
- void run_Flow_unit_test ()

  This function run the unit tests of the flow.

5.32.1 Function Documentation

5.32.1.1 run_Flow_unit_test()

```
void run_Flow_unit_test ( )
```

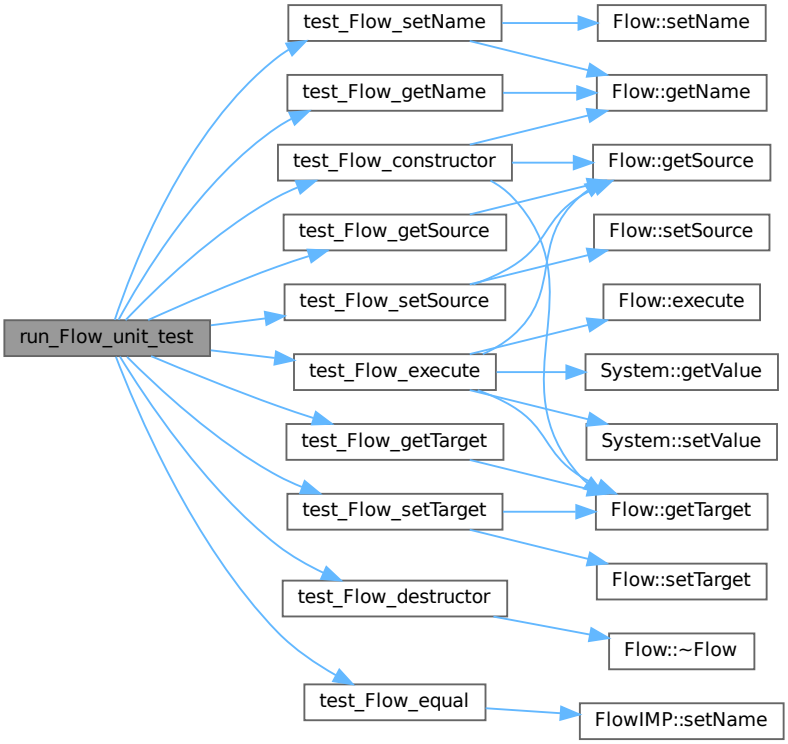
This function run the unit tests of the flow.

```
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:

5.32.1.2 test_Flow_constructor()

```
void test_Flow_constructor ( )
```

This function run the unit test of the flow constructor.

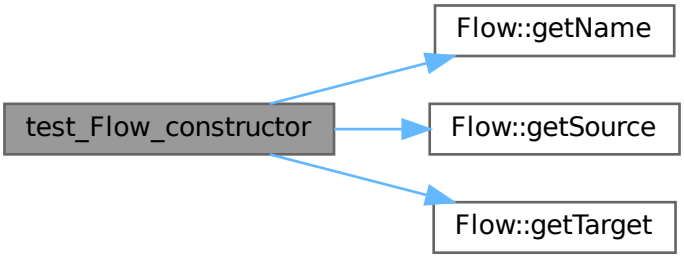
```
00023                                {
00024     std::cout « "        * Constructor tests\n";
00025     Flow* f1 = new Flow_unit_test();
00026     assert(f1->getName() == "NO_NAME");
00027     assert(f1->getSource() == NULL);
00028     assert(f1->getTarget() == NULL);
00029 
00030     Flow* f2 =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

5.32.1.3 test_Flow_destructor()

```
void test_Flow_destructor ( )
```

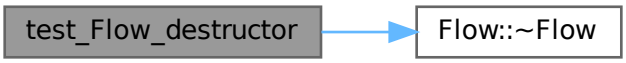
This function run the unit test of the flow destructor.

```
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:

5.32.1.4 test_Flow_equal()

```
void test_Flow_equal ( )
```

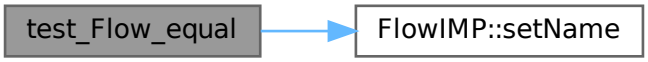
This function run the unit test of the flow equal comparation.

```
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 }
```

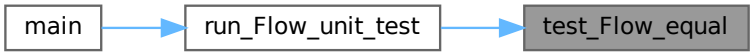
References FlowIMP::setName().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.5 test_Flow_execute()

```
void test_Flow_execute ( )
```

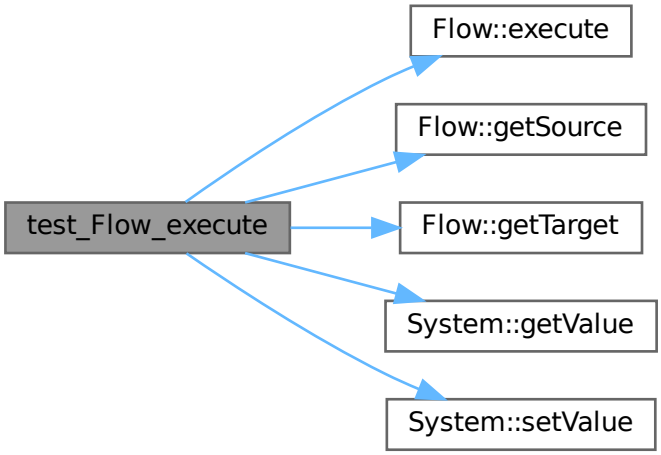
This function run the unit test of the flow execute.

```
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 }
```

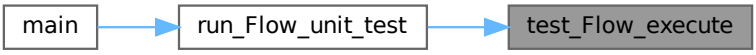
References Flow::execute(), Flow::getSource(), Flow::getTarget(), System::getValue(), and System::setValue().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.6 test_Flow_getName()

```
void test_Flow_getName ( )
```

This function run the unit test of the flow getName.

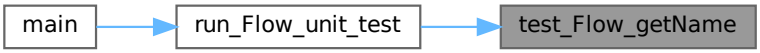
```
00063                           {
00064     std::cout « "         * getName tests\n";
00065     Flow* f1 = new Flow_unit_test();
00066     assert(f1->getName() == "NO_NAME");
00067
00068     Flow* f2 = new Flow_unit_test("f2");
00069     assert(f2->getName() == "f2");
00070
00071     delete f1;
00072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



5.32.1.7 test_Flow_getSource()

```
void test_Flow_getSource ( )
```

This function run the unit test of the flow getSource.

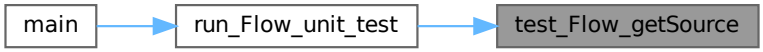
```
00075                                {
00076     std::cout « "        * getSource tests\n";
00077     Flow* f1 = new Flow_unit_test();
00078     assert(f1->getSource() == NULL);
00079
00080     System* s1 = new SystemIMP();
00081     Flow* f2 = new Flow_unit_test("f2", s1);
00082     assert(f2->getSource()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



5.32.1.8 test_Flow_getTarget()

```
void test_Flow_getTarget ( )
```

This function run the unit test of the flow getTarge.

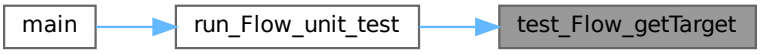
```
00089                                {
00090     std::cout « "        * getTarget tests\n";
00091     Flow* f1 = new Flow_unit_test();
00092     assert(f1->getTarget() == NULL);
00093
00094     System* s1 = new SystemIMP();
00095     System* s2 = new SystemIMP();
00096     Flow* f2 = new Flow_unit_test("f2", 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:



5.32.1.9 test_Flow_setName()

```
void test_Flow_setName ( )
```

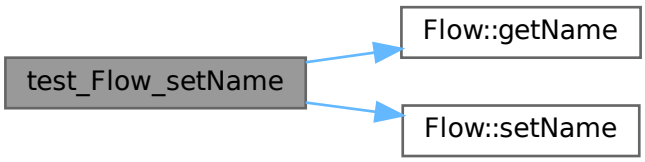
This function run the unit test of the flow setName.

```
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 }
```

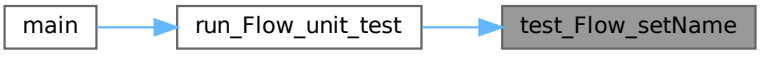
References Flow::getName(), and Flow::setName().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.10 test_Flow_setSource()

```
void test_Flow_setSource ( )
```

This function run the unit test of the flow setSource.

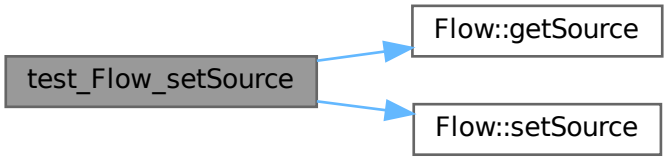
```
00119                             {
00120     std::cout « "         * setSource tests\n";
00121     System* s1 = new SystemIMP();
00122     Flow* f1 = new Flow_unit_test();
00123     f1->setSource(s1);
00124     assert(f1->getSource() != NULL);
00125
```

```
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 }
```

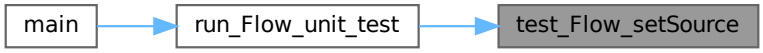
References Flow::getSource(), and Flow::setSource().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.11 test_Flow_setTarget()

```
void test_Flow_setTarget ( )
```

This function run the unit test of the flow setTarge.

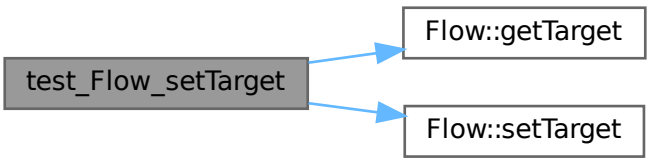
```
00137                                {
00138     std::cout « "        * setTarget tests\n";
00139     System* s1 = new SystemIMP();
00140     Flow* f1 = new Flow_unit_test();
00141     f1->setTarget(s1);
00142     assert(f1->getTarget() != NULL);
00143
00144     System* s2 = new SystemIMP();
00145     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 }
```

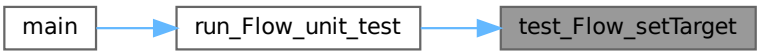
References Flow::getTarget(), and Flow::setTarget().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



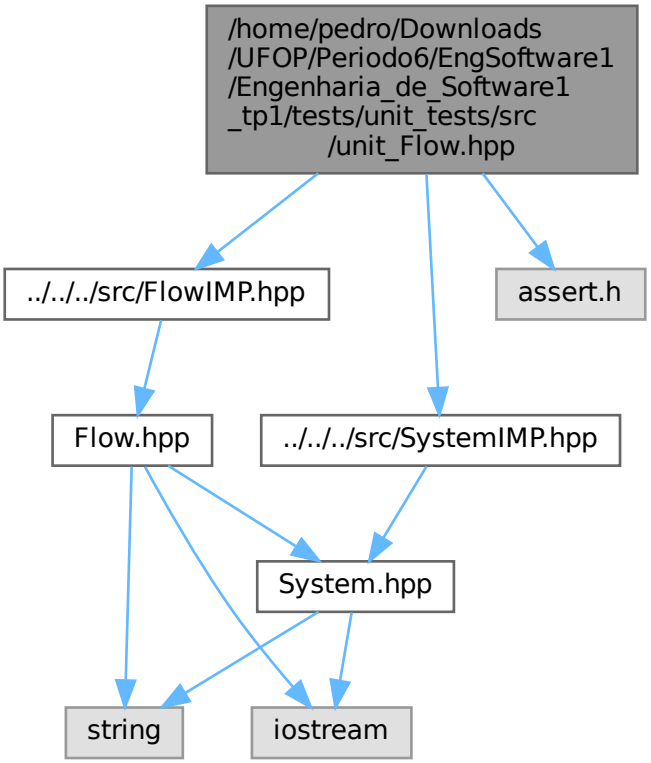
Here is the caller graph for this function:



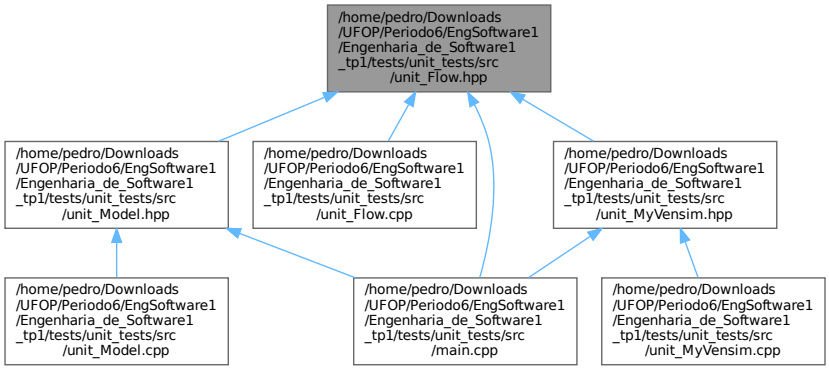
5.33 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_Flow.hpp File Reference

```
#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 comparation.
- void run_Flow_unit_test ()

  This function run the unit tests of the flow.

5.33.1 Function Documentation

5.33.1.1 run_Flow_unit_test()

```
void run_Flow_unit_test ( )
```

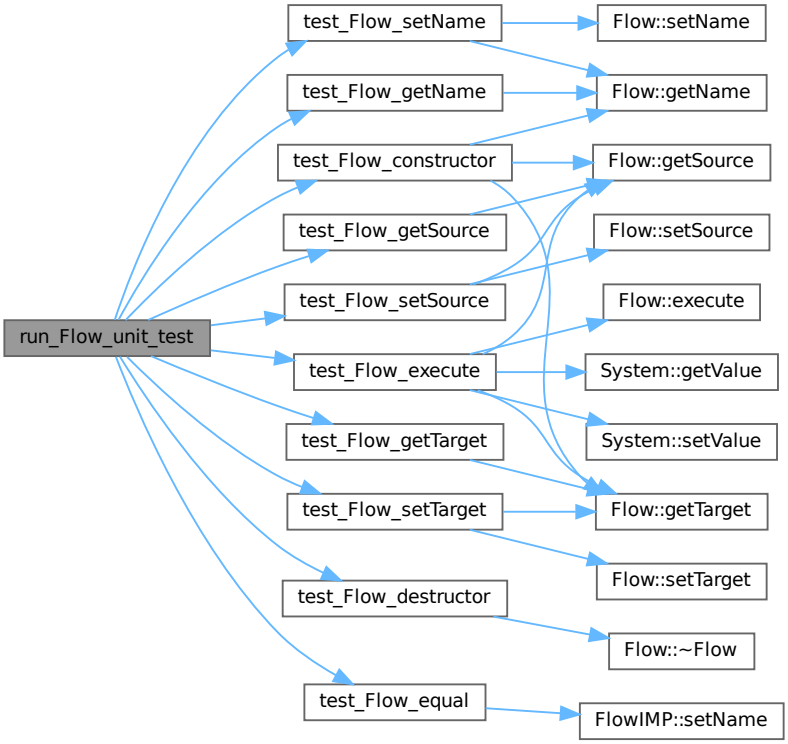
This function run the unit tests of the flow.

```
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:

5.33.1.2 test_Flow_constructor()

```
void test_Flow_constructor ( )
```

This function run the unit test of the flow constructor.

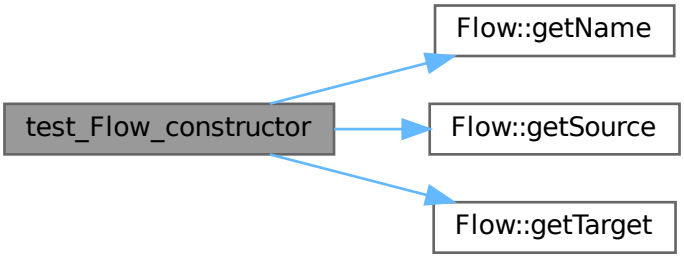
```
00023                                {
00024     std::cout « "        * Constructor tests\n";
00025     Flow* f1 = new Flow_unit_test();
00026     assert(f1->getName() == "NO_NAME");
00027     assert(f1->getSource() == NULL);
00028     assert(f1->getTarget() == NULL);
00029
00030     Flow* f2 = new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

5.33.1.3 test_Flow_destructor()

```
void test_Flow_destructor ( )
```

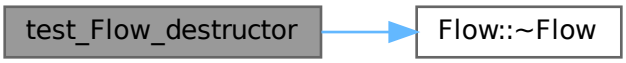
This function run the unit test of the flow destructor.

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

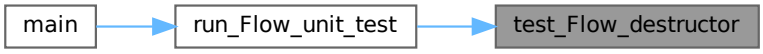
References Flow::~Flow().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.4 test_Flow_equal()

```
void test_Flow_equal ( )
```

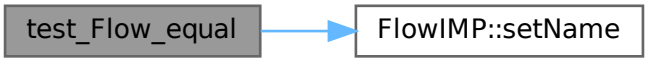
This function run the unit test of the flow equal comparation.

```
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 }
```

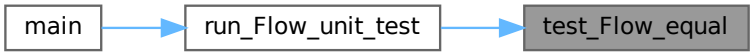
References FlowIMP::setName().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.5 test_Flow_execute()

```
void test_Flow_execute ( )
```

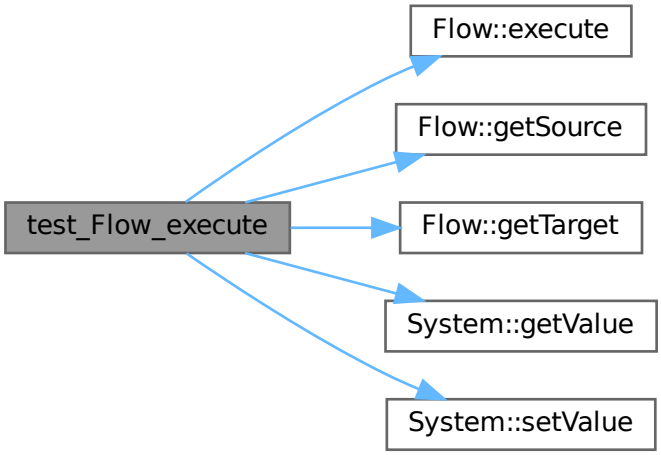
This function run the unit test of the flow execute.

```
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 }
```

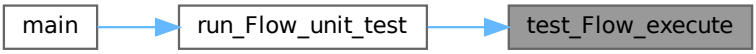
References Flow::execute(), Flow::getSource(), Flow::getTarget(), System::getValue(), and System::setValue().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.6 test_Flow_getName()

```
void test_Flow_getName ( )
```

This function run the unit test of the flow getName.

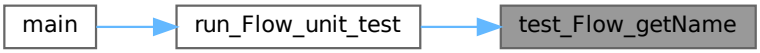
```
00063                           {
00064     std::cout « "         * getName tests\n";
00065     Flow* f1 = new Flow_unit_test();
00066     assert(f1->getName() == "NO_NAME");
00067
00068     Flow* f2 = new Flow_unit_test("f2");
00069     assert(f2->getName() == "f2");
00070
00071     delete f1;
00072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



5.33.1.7 test_Flow_getSource()

```
void test_Flow_getSource ( )
```

This function run the unit test of the flow getSource.

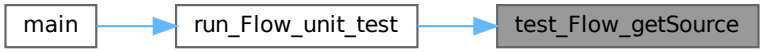
```
00075                                {
00076     std::cout « "        * getSource tests\n";
00077     Flow* f1 = new Flow_unit_test();
00078     assert(f1->getSource() == NULL);
00079
00080     System* s1 = new SystemIMP();
00081     Flow* f2 = new Flow_unit_test("f2", s1);
00082     assert(f2->getSource()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



5.33.1.8 test_Flow_getTarget()

```
void test_Flow_getTarget ( )
```

This function run the unit test of the flow getTarge.

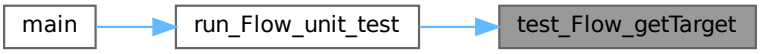
```
00089                                {
00090     std::cout « "        * getTarget tests\n";
00091     Flow* f1 = new Flow_unit_test();
00092     assert(f1->getTarget() == NULL);
00093
00094     System* s1 = new SystemIMP();
00095     System* s2 = new SystemIMP();
00096     Flow* f2 = new Flow_unit_test("f2", 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:



5.33.1.9 test_Flow_setName()

```
void test_Flow_setName ( )
```

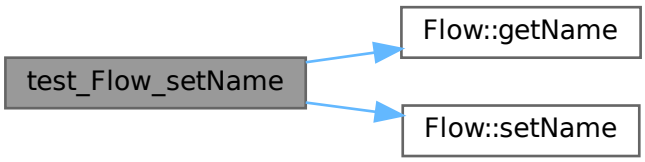
This function run the unit test of the flow setName.

```
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 }
```

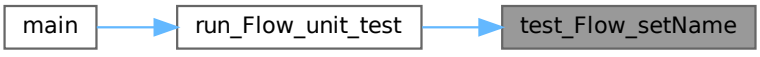
References Flow::getName(), and Flow::setName().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.10 test_Flow_setSource()

```
void test_Flow_setSource ( )
```

This function run the unit test of the flow setSource.

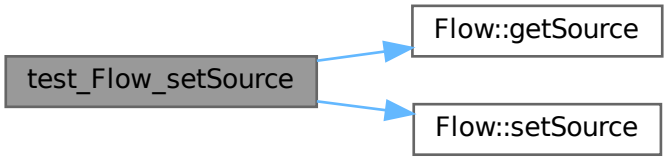
```
00119                             {
00120     std::cout « "         * setSource tests\n";
00121     System* s1 = new SystemIMP();
00122     Flow* f1 = new Flow_unit_test();
00123     f1->setSource(s1);
00124     assert(f1->getSource() != NULL);
00125
```

```
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 }
```

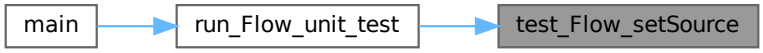
References Flow::getSource(), and Flow::setSource().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.11 test_Flow_setTarget()

```
void test_Flow_setTarget ( )
```

This function run the unit test of the flow setTarge.

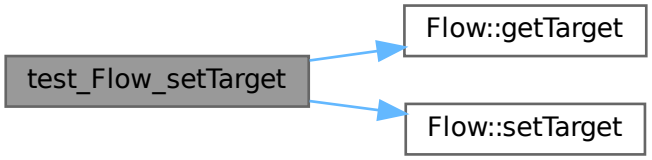
```
00137                                {
00138     std::cout « "        * setTarget tests\n";
00139     System* s1 = new SystemIMP();
00140     Flow* f1 = new Flow_unit_test();
00141     f1->setTarget(s1);
00142     assert(f1->getTarget() != NULL);
00143
00144     System* s2 = new SystemIMP();
00145     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 }
```

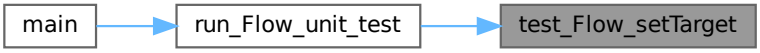
References Flow::getTarget(), and Flow::setTarget().

Referenced by run_Flow_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.34 unit_Flow.hpp

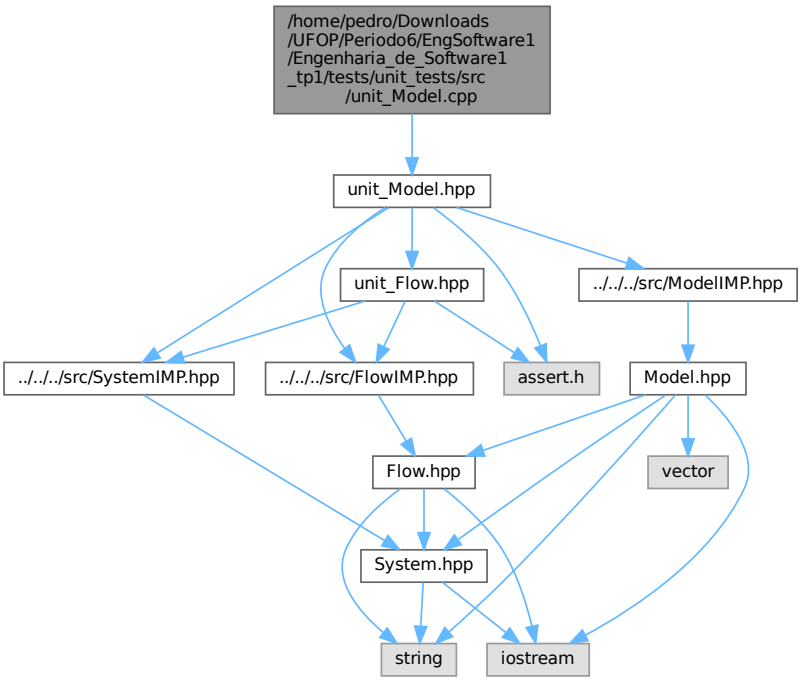
Go to the documentation of this file.

```
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
      one system to another
00017
      *************************************************************************************************************************
00018 class Flow_unit_test : public FlowIMP{
00019     private:
00024         Flow_unit_test(const Flow_unit_test& other);
00025
00026     public:
00027         //Contructor
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
```

5.35 /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.

5.35.1 Function Documentation

5.35.1.1 run_Model_unit_test()

```
void run_Model_unit_test ( )
```

This function run the unit tests of the model.

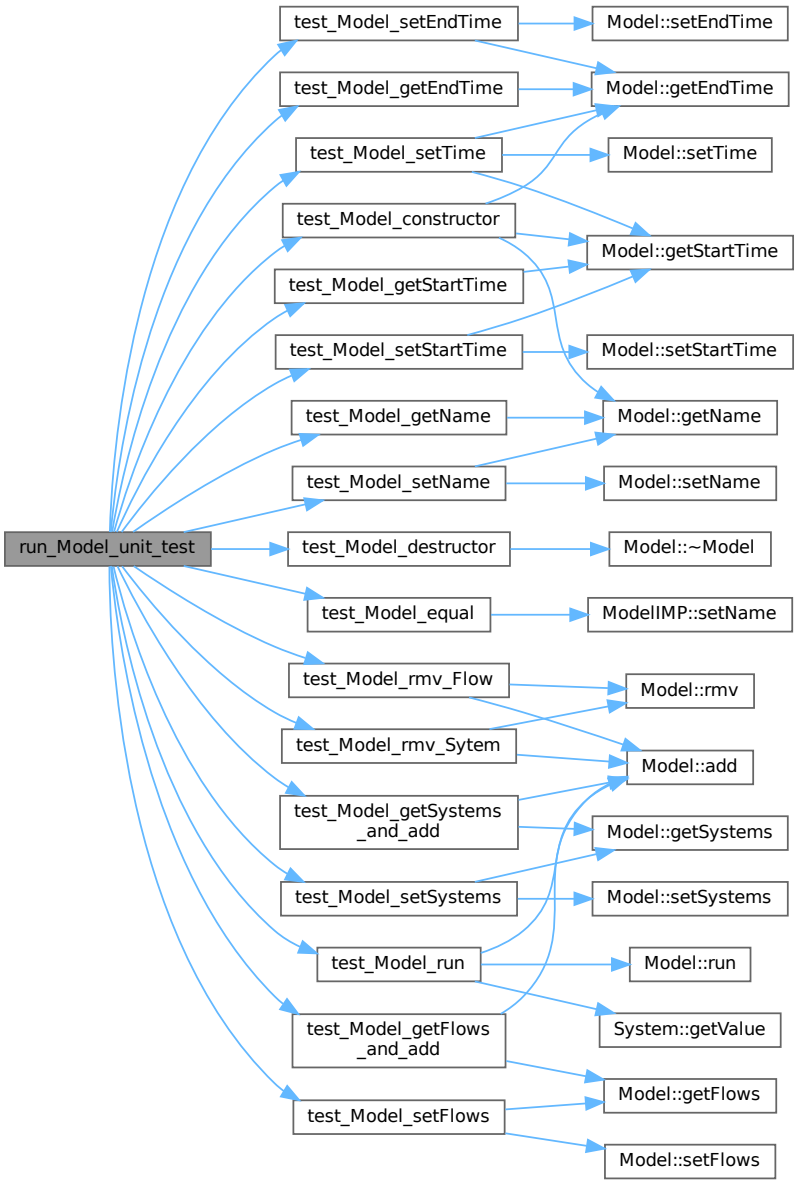
```
00245                                {
00246     std::cout « "    Start Model unit tests\n";
00247     test_Model_constructor();
00248     test_Model_destructor();
00249     test_Model_getName();
00250     test_Model_getSystems_and_add();
00251     test_Model_getFlows_and_add();
00252     test_Model_getStartTime();
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\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:

5.35.1.2 test_Model_constructor()

```
void test_Model_constructor ( )
```

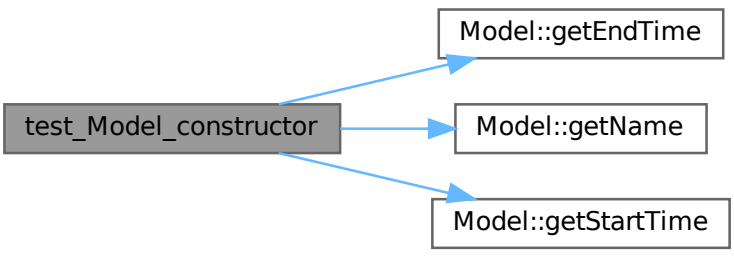
This function run the unit test of the model constructor.

```
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:

5.35.1.3 test_Model_destructor()

```
void test_Model_destructor ( )
```

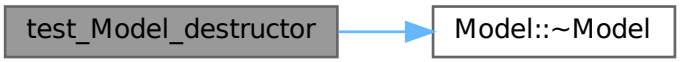
This function run the unit test of the model destructor.

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

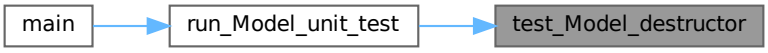
References Model::~Model().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.4 test_Model_equal()

```
void test_Model_equal ( )
```

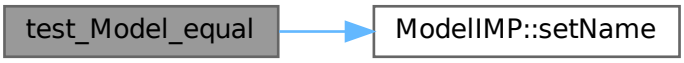
This function run the unit test of the model equal.

```
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 }
```

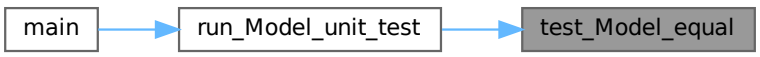
References ModelIMP::setName().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.5 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:

5.35.1.6 test_Model_getFlows_and_add()

```
void test_Model_getFlows_and_add ( )
```

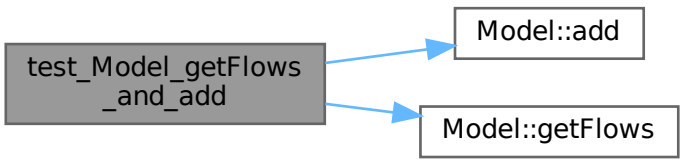
This function run the unit test of the model getFlows and add Flow.

```
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 }
```

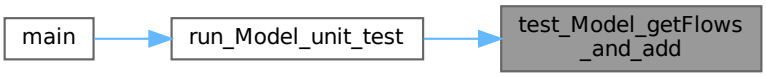
References Model::add(), and Model::getFlows().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.7 test_Model_getName()

```
void test_Model_getName ( )
```

This function run the unit test of the model getName.

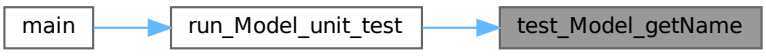
```
00037                           {
00038     std::cout « "         * getName tests\n";
00039     Model* m1 = new ModelIMP();
00040     assert(m1->getName() == "NO_NAME");
00041
00042     Model* m2 = new ModelIMP("m2");
00043     assert(m2->getName() == "m2");
00044
00045     delete m1;
00046     delete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



5.35.1.8 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:

5.35.1.9 test_Model_getSystems_and_add()

```
void test_Model_getSystems_and_add ( )
```

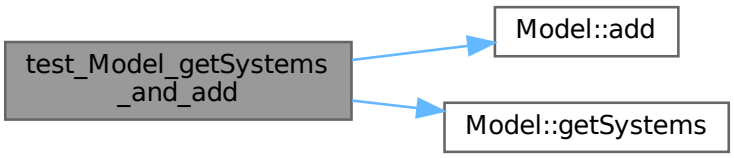
This function run the unit test of the model getSystems and add System.

```
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 }
```

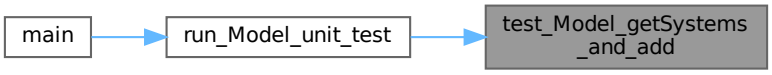
References Model::add(), and Model::getSystems().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.10 test_Model_rmv_Flow()

```
void test_Model_rmv_Flow ( )
```

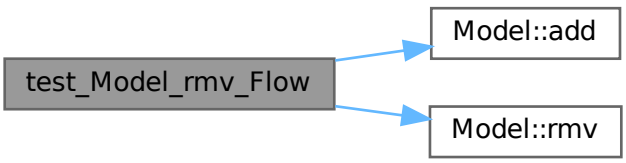
This function run the unit test of the model rmv Flow.

```
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 }
```

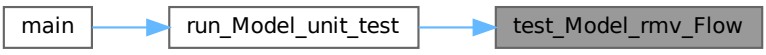
References Model::add(), and Model::rmv().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.11 test_Model_rmv_Sytem()

```
void test_Model_rmv_Sytem ( )
```

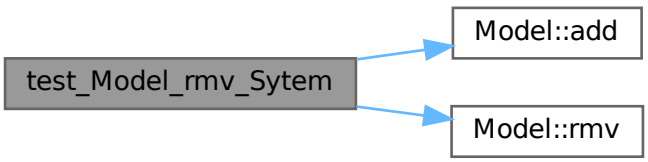
This function run the unit test of the model rmv System.

```
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:

5.35.1.12 test_Model_run()

```
void test_Model_run ( )
```

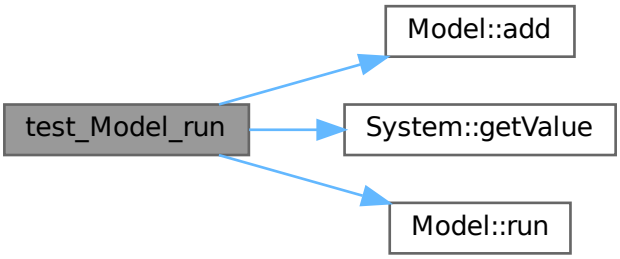
This function run the unit test of the model run.

```
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 }
```

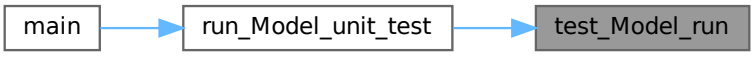
References Model::add(), System::getValue(), and Model::run().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.13 test_Model_setEndTime()

```
void test_Model_setEndTime ( )
```

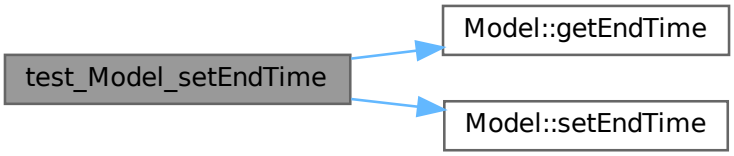
This function run the unit test of the model setEndTime.

```
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:

5.35.1.14 test_Model_setFlows()

```
void test_Model_setFlows ( )
```

This function run the unit test of the model setFlows.

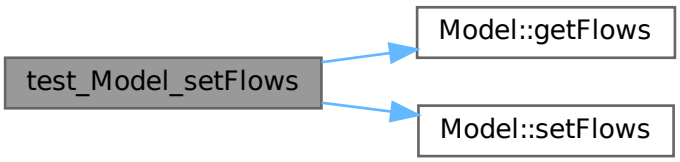
```
00126                                {
00127     std::cout « "        * setFlows tests\n";
00128     Flow* f1 = new Flow_unit_test("f1");
00129     Model* m1 = new ModelIMP("m1");
00130     std::vector<Flow*> flows;
00131     flows.push_back(f1);
00132     m1->setFlows(flows);
00133     assert(m1->getFlows()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

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

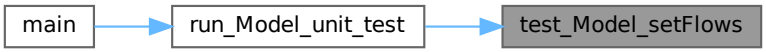
References Model::getFlows(), and Model::setFlows().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.15 test_Model_setName()

```
void test_Model_setName ( )
```

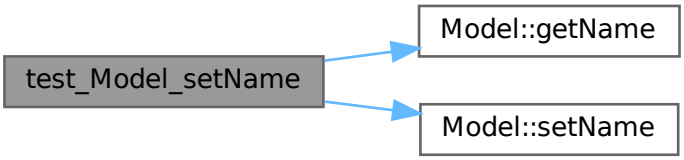
This function run the unit test of the model setName.

```
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 }
```

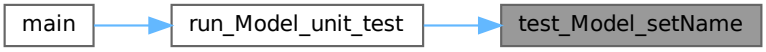
References Model::getName(), and Model::setName().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.16 test_Model_setStartTime()

```
void test_Model_setStartTime ( )
```

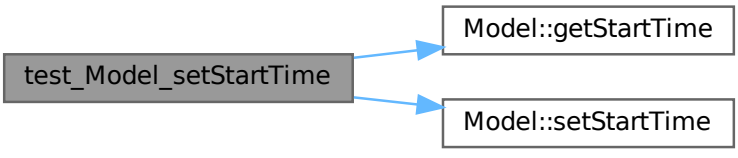
This function run the unit test of the model setStartTime.

```
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:

5.35.1.17 test_Model_setSystems()

```
void test_Model_setSystems ( )
```

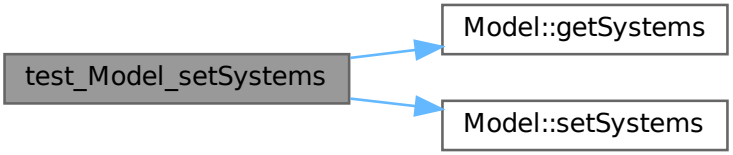
This function run the unit test of the model setSystems.

```
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:

5.35.1.18 test_Model_setTime()

```
void test_Model_setTime ( )
```

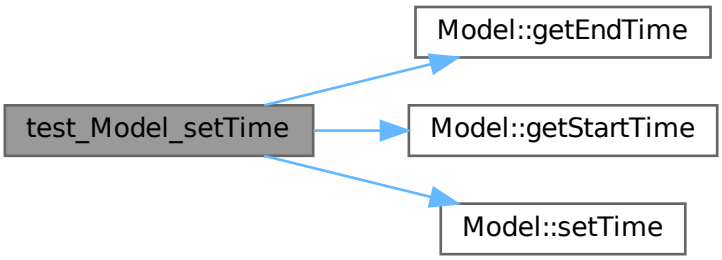
This function run the unit test of the model setTime.

```
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 }
```

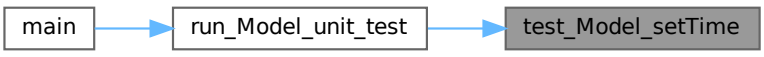
References Model::getEndTime(), Model::getStartTime(), and Model::setTime().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:

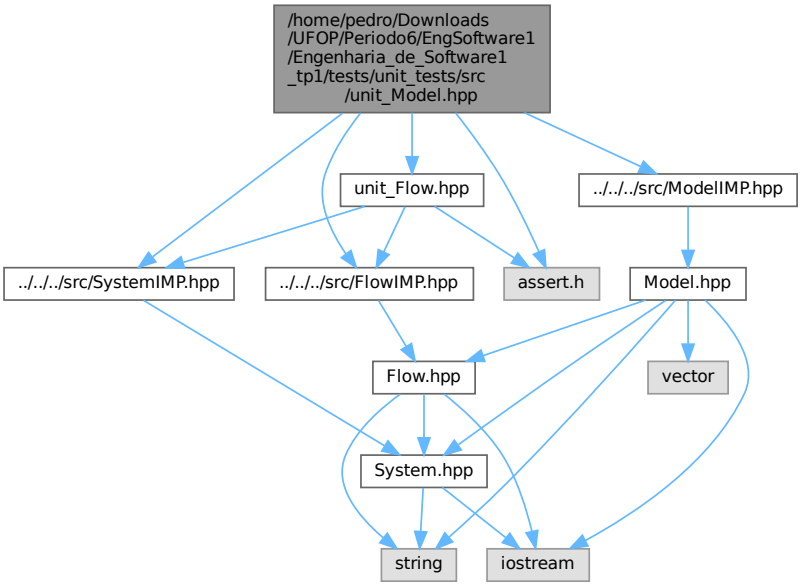


5.36 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_Model.hpp File Reference

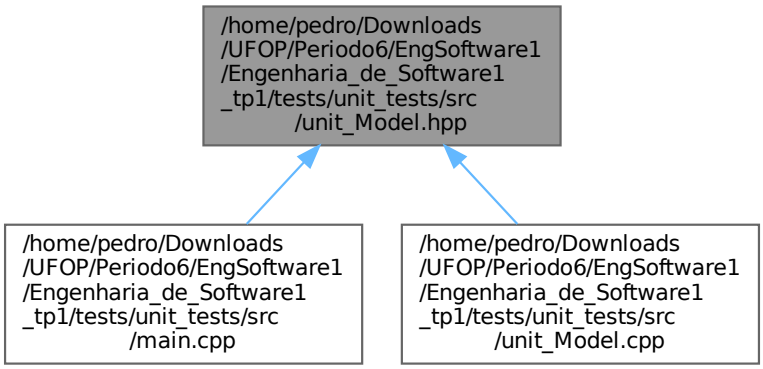
```
#include "../../../src/FlowIMP.hpp"
#include "../../../src/SystemIMP.hpp"
```

```
#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.

5.36.1 Function Documentation

5.36.1.1 run_Model_unit_test()

```
void run_Model_unit_test ( )
```

This function run the unit tests of the model.

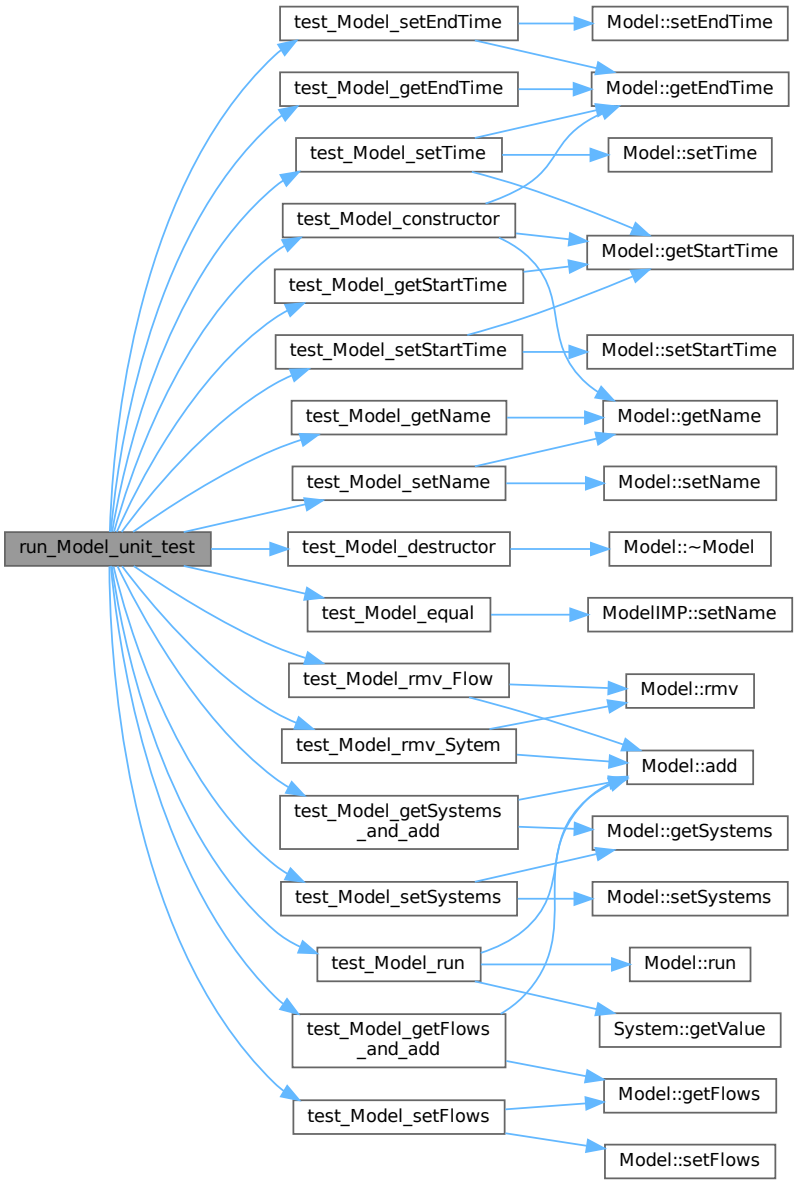
```
00245                                {
00246     std::cout « "    Start Model unit tests\n";
00247     test_Model_constructor();
00248     test_Model_destructor();
00249     test_Model_getName();
00250     test_Model_getSystems_and_add();
00251     test_Model_getFlows_and_add();
00252     test_Model_getStartTime();
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\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:

5.36.1.2 test_Model_constructor()

```
void test_Model_constructor ( )
```

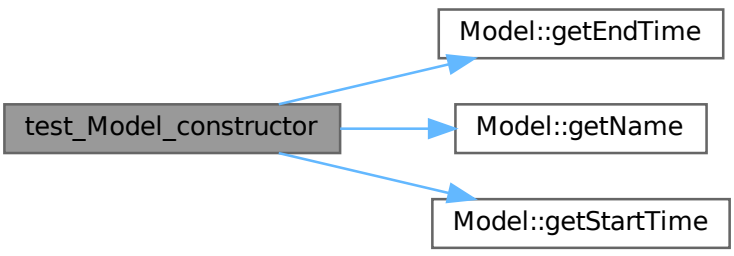
This function run the unit test of the model constructor.

```
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:

5.36.1.3 test_Model_destructor()

```
void test_Model_destructor ( )
```

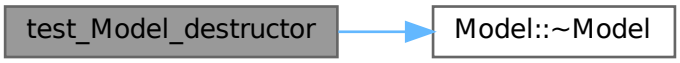
This function run the unit test of the model destructor.

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

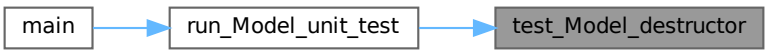
References Model::~Model().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.4 test_Model_equal()

```
void test_Model_equal ( )
```

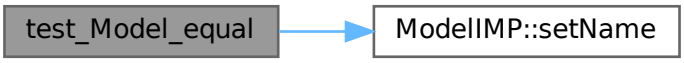
This function run the unit test of the model equal.

```
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 }
```

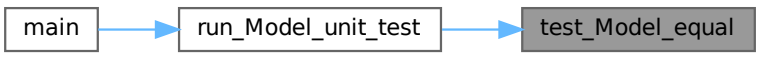
References ModelIMP::setName().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.5 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:

5.36.1.6 test_Model_getFlows_and_add()

```
void test_Model_getFlows_and_add ( )
```

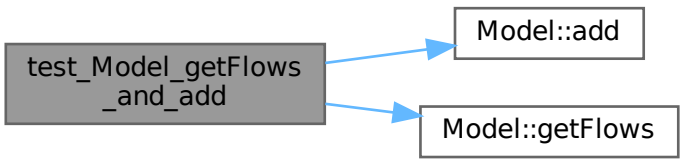
This function run the unit test of the model getFlows and add Flow.

```
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 }
```

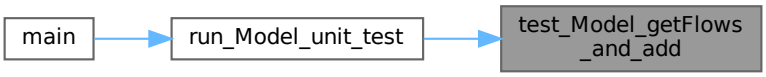
References Model::add(), and Model::getFlows().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.7 test_Model_getName()

```
void test_Model_getName ( )
```

This function run the unit test of the model getName.

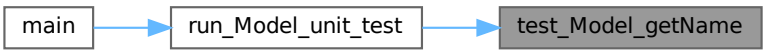
```
00037                           {
00038     std::cout « "         * getName tests\n";
00039     Model* m1 = new ModelIMP();
00040     assert(m1->getName() == "NO_NAME");
00041
00042     Model* m2 = new ModelIMP("m2");
00043     assert(m2->getName() == "m2");
00044
00045     delete m1;
00046     delete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



5.36.1.8 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:

5.36.1.9 test_Model_getSystems_and_add()

```
void test_Model_getSystems_and_add ( )
```

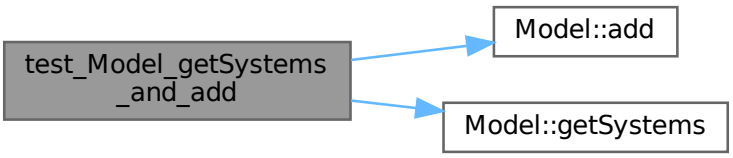
This function run the unit test of the model getSystems and add System.

```
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 }
```

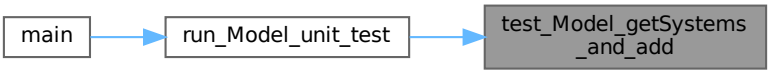
References Model::add(), and Model::getSystems().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.10 test_Model_rmv_Flow()

```
void test_Model_rmv_Flow ( )
```

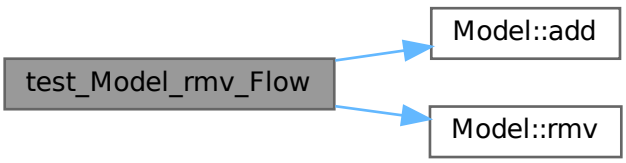
This function run the unit test of the model rmv Flow.

```
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 }
```

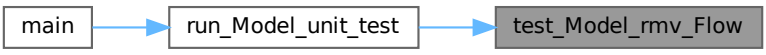
References Model::add(), and Model::rmv().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.11 test_Model_rmv_Sytem()

```
void test_Model_rmv_Sytem ( )
```

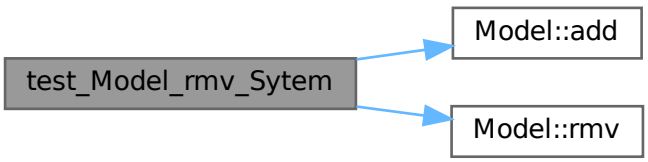
This function run the unit test of the model rmv System.

```
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:

5.36.1.12 test_Model_run()

```
void test_Model_run ( )
```

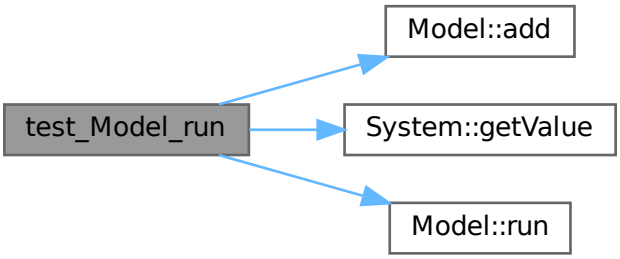
This function run the unit test of the model run.

```
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 }
```

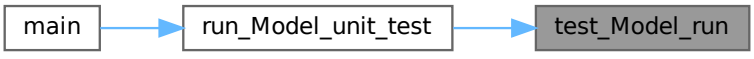
References Model::add(), System::getValue(), and Model::run().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.13 test_Model_setEndTime()

```
void test_Model_setEndTime ( )
```

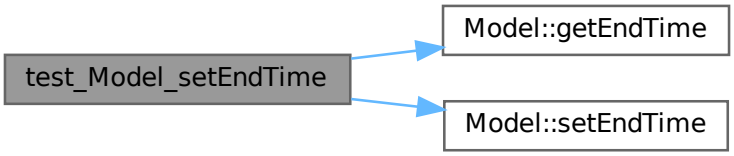
This function run the unit test of the model setEndTime.

```
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:

5.36.1.14 test_Model_setFlows()

```
void test_Model_setFlows ( )
```

This function run the unit test of the model setFlows.

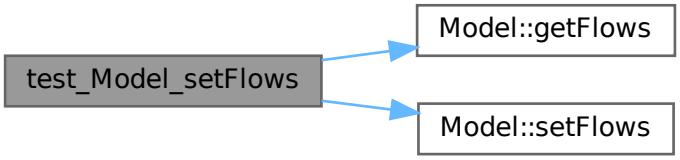
```
00126                                 {
00127     std::cout « "        * setFlows tests\n";
00128     Flow* f1 = new Flow_unit_test("f1");
00129     Model* m1 = new ModelIMP("m1");
00130     std::vector<Flow*> flows;
00131     flows.push_back(f1);
00132     m1->setFlows(flows);
00133     assert(m1->getFlows()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

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

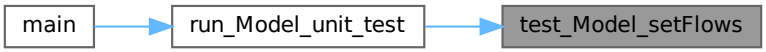
References Model::getFlows(), and Model::setFlows().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.15 test_Model_setName()

```
void test_Model_setName ( )
```

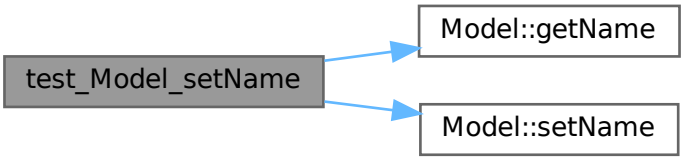
This function run the unit test of the model setName.

```
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 }
```

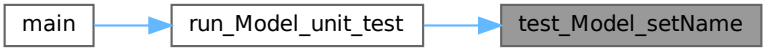
References Model::getName(), and Model::setName().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.16 test_Model_setStartTime()

```
void test_Model_setStartTime ( )
```

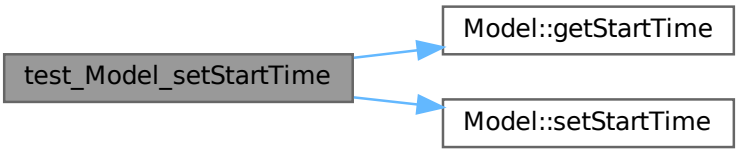
This function run the unit test of the model setStartTime.

```
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:

5.36.1.17 test_Model_setSystems()

```
void test_Model_setSystems ( )
```

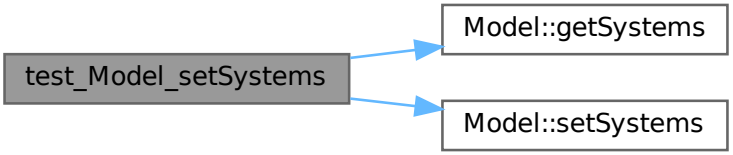
This function run the unit test of the model setSystems.

```
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:

5.36.1.18 test_Model_setTime()

```
void test_Model_setTime ( )
```

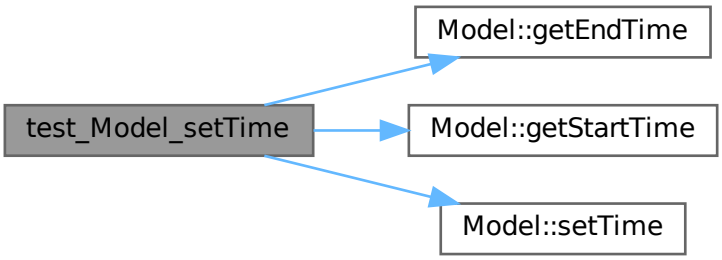
This function run the unit test of the model setTime.

```
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 }
```

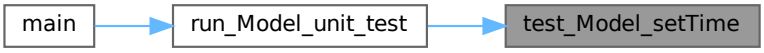
References Model::getEndTime(), Model::getStartTime(), and Model::setTime().

Referenced by run_Model_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.37 unit_Model.hpp

Go to the documentation of this file.

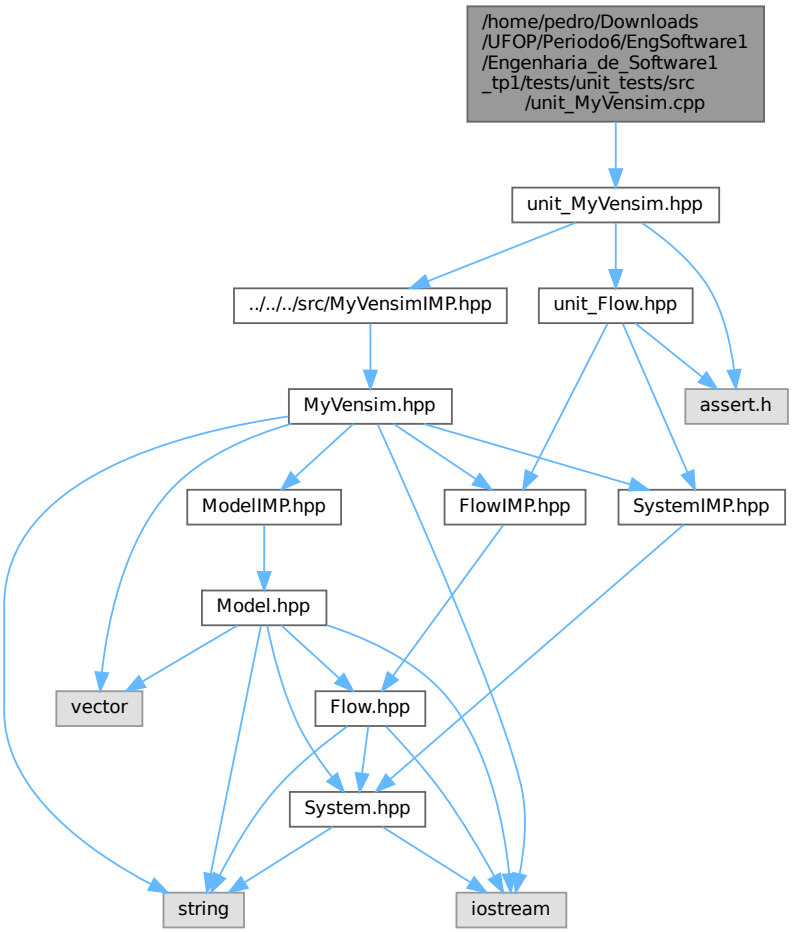
```
00001 /*************************************************************
00002  * @file unit_Model.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
```

```
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
```

5.38 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_MyVensim.cpp File Reference

```
#include "unit_MyVensim.hpp"
```

Include dependency graph for unit_MyVensim.cpp:



Functions

- void test_MyVensim_constructor ()

  This function run the unit test of the MyVensim constructor.
- void test_MyVensim_destructor ()

  This function run the unit test of the MyVensim destructor.
- void test_MyVensim_getName ()

  This function run the unit test of the MyVensim getName.
- void test_MyVensim_getSystems_and_add ()

  This function run the unit test of the MyVensim getSystems and add System.
- void test_MyVensim_getFlows_and_add ()

This function run the unit test of the MyVensim getFlows and add Flow.

- void test_MyVensim_getModels_and_add ()

  This function run the unit test of the MyVensim getModels and add Model.
- void test_MyVensim_setName ()

  This function run the unit test of the MyVensim setName.
- void test_MyVensim_setSystems ()

  This function run the unit test of the MyVensim setSystems.
- void test_MyVensim_setFlows ()

  This function run the unit test of the MyVensim setFlows.
- void test_MyVensim_setModels ()

  This function run the unit test of the MyVensim setModels.
- void test_MyVensim_equal ()

  This function run the unit test of the MyVensim equal.
- void test_MyVensim_rmv_Sytem ()

  This function run the unit test of the MyVensim rmv System.
- void test_MyVensim_rmv_Flow ()

  This function run the unit test of the MyVensim rmv Flow.
- void test_MyVensim_rmv_Model ()

  This function run the unit test of the MyVensim rmv Model.
- void test_MyVensim_run ()

  This function run the unit test of the MyVensim run.
- void run_MyVensim_unit_test ()

  This function run the unit tests of the MyVensim.

5.38.1 Function Documentation

5.38.1.1 run_MyVensim_unit_test()

```
void run_MyVensim_unit_test ( )
```

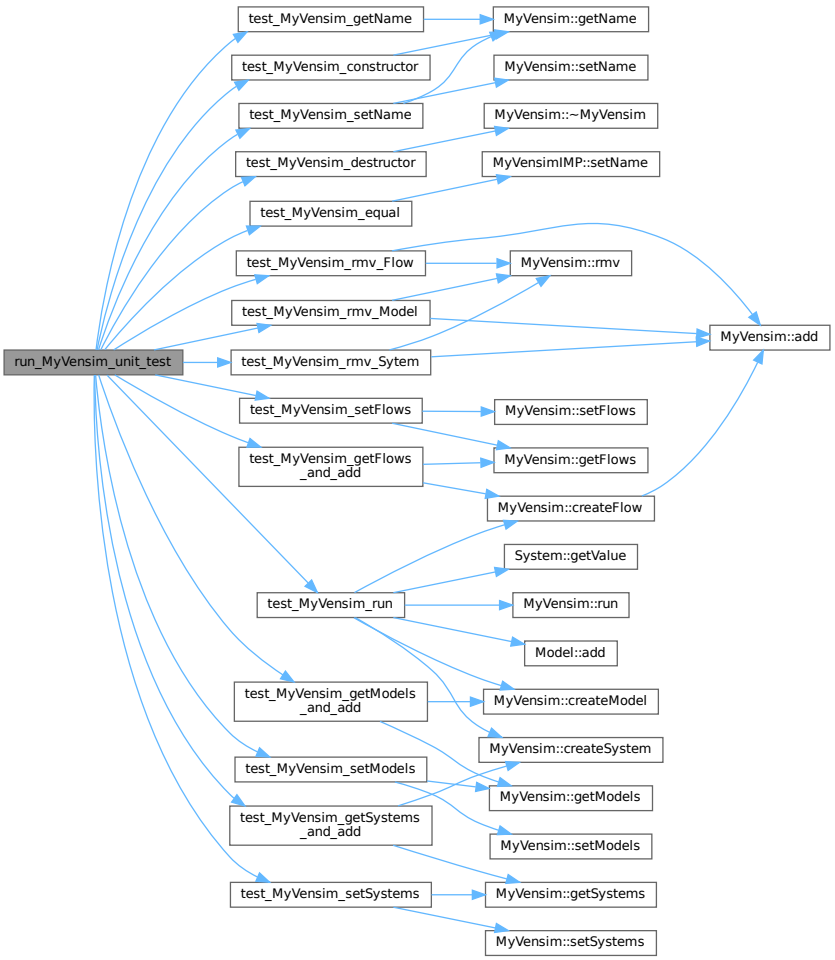
This function run the unit tests of the MyVensim.

```
00222                                {
00223     std::cout « "    Start MyVensim unit tests\n";
00224     test_MyVensim_constructor();
00225     test_MyVensim_destructor();
00226     test_MyVensim_getName();
00227     test_MyVensim_getSystems_and_add();
00228     test_MyVensim_getFlows_and_add();
00229     test_MyVensim_getModels_and_add();
00230     test_MyVensim_setName();
00231     test_MyVensim_setSystems();
00232     test_MyVensim_setFlows();
00233     test_MyVensim_setModels();
00234     test_MyVensim_rmv_Sytem();
00235     test_MyVensim_rmv_Flow();
00236     test_MyVensim_rmv_Model();
00237     test_MyVensim_run();
00238     test_MyVensim_equal();
00239     std::cout « "    End MyVensim unit tests\n";
00240 }
```

References test_MyVensim_constructor(), test_MyVensim_destructor(), test_MyVensim_equal(), test_MyVensim_getFlows_and_add(), test_MyVensim_getModels_and_add(), test_MyVensim_getName(), test_MyVensim_getSystems_and_add(), test_MyVensim_rmv_Flow(), test_MyVensim_rmv_Model(), test_MyVensim_rmv_Sytem(), test_MyVensim_run(), test_MyVensim_setFlows(), test_MyVensim_setModels(), test_MyVensim_setName(), and test_MyVensim_setSystems().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

5.38.1.2 test_MyVensim_constructor()

```
void test_MyVensim_constructor ( )
```

This function run the unit test of the MyVensim constructor.

```
00003                                   {
```

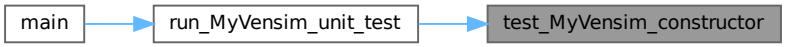
```
00004     std::cout « "        * Constructor tests\n";
00005
00006     MyVensim* m1 = new MyVensimIMP();
00007     assert(m1->getName() == "NO_NAME");
00008
00009     MyVensim* m2 = new MyVensimIMP("m2");
00010     assert(m2->getName() == "m2");
00011
00012     delete m1;
00013     delete m2;
00014 }
```

References MyVensim::getName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:

Here is the caller graph for this function:



5.38.1.3 test_MyVensim_destructor()

```
void test_MyVensim_destructor ( )
```

This function run the unit test of the MyVensim destructor.

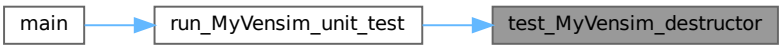
```
00016                                {
00017     std::cout « "        * Destructor tests\n";
00018     MyVensim* m1 = new MyVensimIMP();
00019     m1->~MyVensim();
00020 }
```

References MyVensim::~MyVensim().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:

Here is the caller graph for this function:



5.38.1.4 test_MyVensim_equal()

```
void test_MyVensim_equal ( )
```

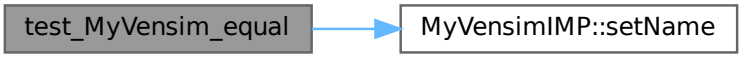
This function run the unit test of the MyVensim equal.

```
00144                                {
00145     std::cout « "         * Equal tests\n";
00146
00147     MyVensimIMP* m1 = new MyVensimIMP();
00148     MyVensimIMP* m2 = new MyVensimIMP();
00149     assert(*m1 == *m2);
00150
00151     m1->setName("m1");
00152     assert(m1 != m2);
00153
00154     delete m1;
00155     delete m2;
00156 }
```

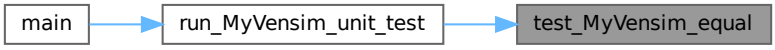
References MyVensimIMP::setName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.5 test_MyVensim_getFlows_and_add()

```
void test_MyVensim_getFlows_and_add ( )
```

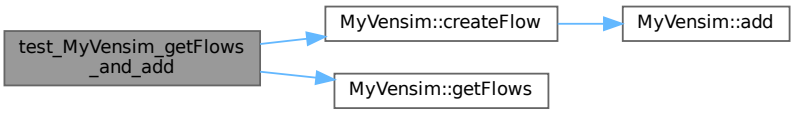
This function run the unit test of the MyVensim getFlows and add Flow.

```
00049                                       {
00050     std::cout « "        * getFlows and add Flows tests\n";
00051
00052     MyVensim* m1 = new MyVensimIMP("m1");
00053     Flow* f1 = m1->createFlow<Flow_unit_test>("f1");
00054
00055     std::vector<Flow*> flows;
00056     flows.push_back(f1);
00057
00058     assert(m1->getFlows() == flows);
00059
00060     delete f1;
00061     delete m1;
00062 }
```

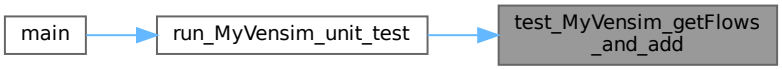
References MyVensim::createFlow(), and MyVensim::getFlows().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.6 test_MyVensim_getModels_and_add()

```
void test_MyVensim_getModels_and_add ( )
```

This function run the unit test of the MyVensim getModels and add Model.

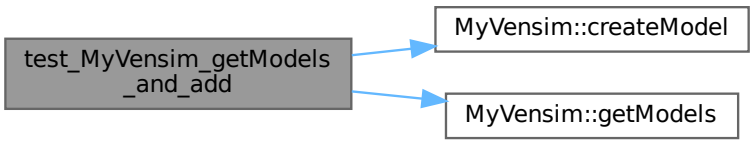
```
00064                                       {
00065     std::cout « "        * getModels and add Models tests\n";
00066
00067     MyVensim* m1 = new MyVensimIMP("m1");
00068     Model* f1 = m1->createModel("f1");
00069
00070     std::vector<Model*> Models;
00071     Models.push_back(f1);
00072
00073     assert(m1->getModels() == Models);
00074
```

```
00075     delete f1;
00076     delete m1;
00077 }
```

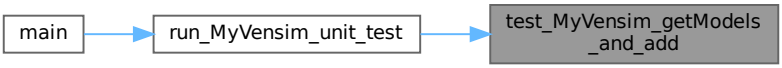
References MyVensim::createModel(), and MyVensim::getModels().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.7 test_MyVensim_getName()

```
void test_MyVensim_getName ( )
```

This function run the unit test of the MyVensim getName.

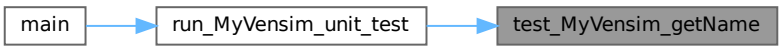
```
00022                       {
00023     std::cout « "        * getName tests\n";
00024     MyVensim* m1 = new MyVensimIMP();
00025     assert(m1->getName() == "NO_NAME");
00026
00027     MyVensim* m2 = new MyVensimIMP("m2");
00028     assert(m2->getName() == "m2");
00029
00030     delete m1;
00031     delete m2;
00032 }
```

References MyVensim::getName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:

Here is the caller graph for this function:



5.38.1.8 test_MyVensim_getSystems_and_add()

```
void test_MyVensim_getSystems_and_add ( )
```

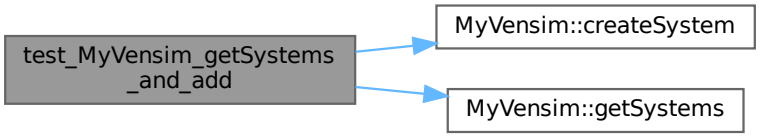
This function run the unit test of the MyVensim getSystems and add System.

```
00034                                       {
00035     std::cout << "        * getSystems and add Systems tests\n";
00036
00037     MyVensim* m1 = new MyVensimIMP("m1");
00038     System* s1 = m1->createSystem("s1");
00039
00040     std::vector<System*> systems;
00041     systems.push_back(s1);
00042
00043     assert(m1->getSystems() == systems);
00044
00045     delete s1;
00046     delete m1;
00047 }
```

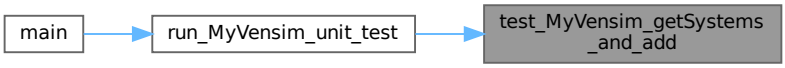
References MyVensim::createSystem(), and MyVensim::getSystems().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.9 test_MyVensim_rmv_Flow()

```
void test_MyVensim_rmv_Flow ( )
```

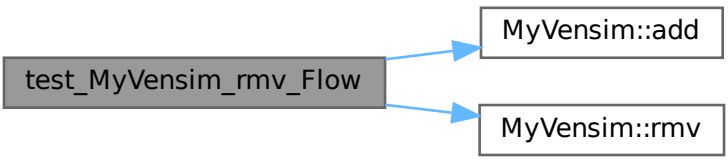
This function run the unit test of the MyVensim rmv Flow.

```
00171                                   {
00172     std::cout « "         * Remove Flow tests\n";
00173
00174     MyVensim* MyVensim1 = new MyVensimIMP();
00175     FlowIMP* flow1 = new Flow_unit_test("flow1");
00176
00177     MyVensim1->add(flow1);
00178     assert(MyVensim1->rmv(flow1));
00179
00180     delete MyVensim1;
00181     delete flow1;
00182 }
```

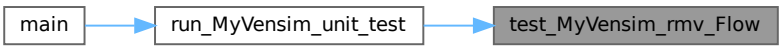
References MyVensim::add(), and MyVensim::rmv().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.10 test_MyVensim_rmv_Model()

```
void test_MyVensim_rmv_Model ( )
```

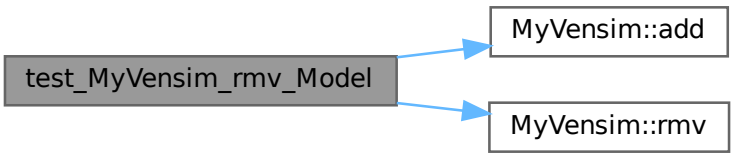
This function run the unit test of the MyVensim rmv Model.

```
00184                                {
00185     std::cout « "        * Remove Model tests\n";
00186
00187     MyVensim* MyVensim1 = new MyVensimIMP();
00188     ModelIMP* Model1 = new ModelIMP("Model1");
00189
00190     MyVensim1->add(Model1);
00191     assert(MyVensim1->rmv(Model1));
00192
00193     delete MyVensim1;
00194     delete Model1;
00195 }
```

References MyVensim::add(), and MyVensim::rmv().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:

5.38.1.11 test_MyVensim_rmv_Sytem()

```
void test_MyVensim_rmv_Sytem ( )
```

This function run the unit test of the MyVensim rmv System.

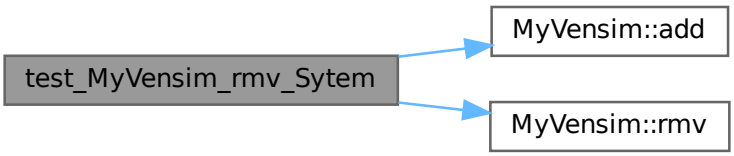
```
00158                                {
00159     std::cout « "        * Remove System tests\n";
00160
00161     MyVensim* MyVensim1 = new MyVensimIMP();
00162     System* system1 = new SystemIMP("system1");
00163
00164     MyVensim1->add(system1);
00165     assert(MyVensim1->rmv(system1));
00166
00167     delete MyVensim1;
```

```
00168     delete system1;
00169 }
```

References MyVensim::add(), and MyVensim::rmv().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:

5.38.1.12 test_MyVensim_run()

```
void test_MyVensim_run ( )
```

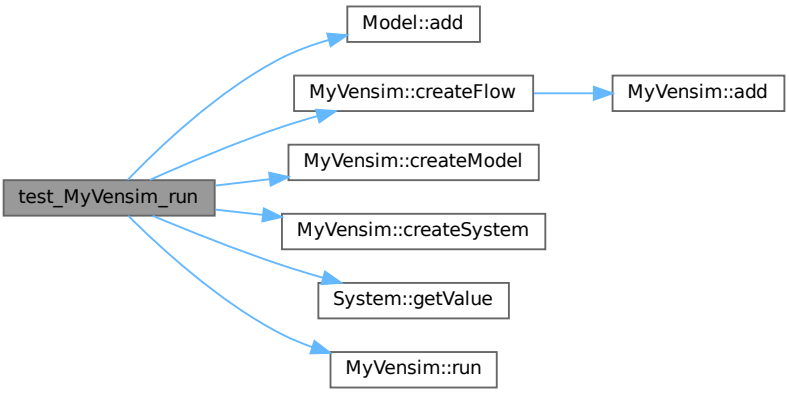
This function run the unit test of the MyVensim run.

```
00197                       {
00198     std::cout « "        * Run tests\n";
00199
00200     MyVensim* mv = new MyVensimIMP("mv");
00201     System* s1 = mv->createSystem("s1", 100);
00202     System* s2 = mv->createSystem("s2", 0.0);
00203     Flow* f1 = mv->createFlow<Flow_unit_test>("f1", s1, s2);
00204     Model* m1 = mv->createModel("m1", 0, 1);
00205
00206     m1->add(s1);
00207     m1->add(s2);
00208     m1->add(f1);
00209
00210     mv->run();
00211
00212     assert(s1->getValue() == 0);
00213     assert(s2->getValue() == 100);
00214
00215     delete mv;
00216     delete s1;
00217     delete s2;
00218     delete f1;
00219     delete m1;
00220 }
```

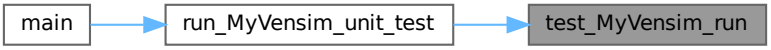
References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.13 test_MyVensim_setFlows()

```
void test_MyVensim_setFlows ( )
```

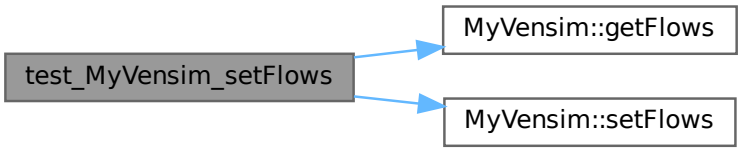
This function run the unit test of the MyVensim setFlows.

```
00113                                {
00114     std::cout « "         * setFlows tests\n";
00115     MyVensim* m1 = new MyVensimIMP("m1");
00116     Flow* f1 = new Flow_unit_test("f1");
00117
00118     std::vector<Flow*> flows;
00119     flows.push_back(f1);
00120     m1->setFlows(flows);
00121
00122     assert(m1->getFlows() == flows);
00123
00124     delete f1;
00125     delete m1;
00126 }
```

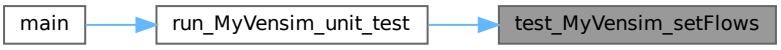
References MyVensim::getFlows(), and MyVensim::setFlows().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.14 test_MyVensim_setModels()

```
void test_MyVensim_setModels ( )
```

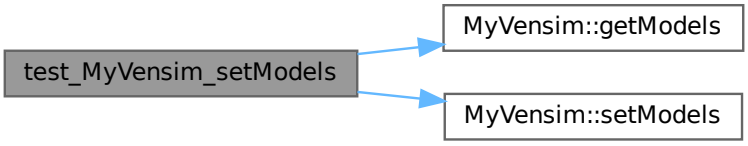
This function run the unit test of the MyVensim setModels.

```
00128                                   {
00129     std::cout « "         * setModels tests\n";
00130
00131     Model* f1 = new ModelIMP("f1");
00132     MyVensim* m1 = new MyVensimIMP("m1");
00133
00134     std::vector<Model*> Models;
00135     Models.push_back(f1);
00136     m1->setModels(Models);
00137
00138     assert(m1->getModels() == Models);
00139
00140     delete f1;
00141     delete m1;
00142 }
```

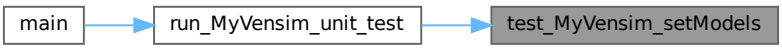
References MyVensim::getModels(), and MyVensim::setModels().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.15 test_MyVensim_setName()

```
void test_MyVensim_setName ( )
```

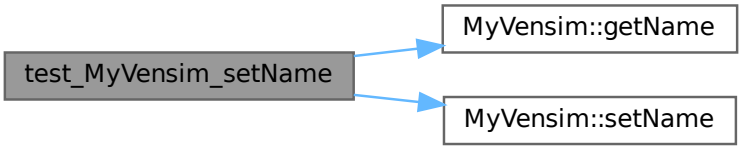
This function run the unit test of the MyVensim setName.

```
00079                                {
00080     std::cout « "         * setName tests\n";
00081
00082     MyVensim* m1 = new MyVensimIMP();
00083
00084     m1->setName("m1");
00085     assert(m1->getName() != "NO_NAME");
00086
00087     MyVensim* m2 = new MyVensimIMP("m");
00088
00089     m2->setName("m2");
00090
00091     assert(m2->getName() == "m2");
00092
00093     delete m1;
00094     delete m2;
00095 }
```

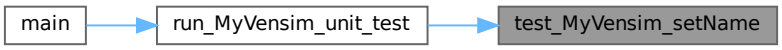
References MyVensim::getName(), and MyVensim::setName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.1.16 test_MyVensim_setSystems()

```
void test_MyVensim_setSystems ( )
```

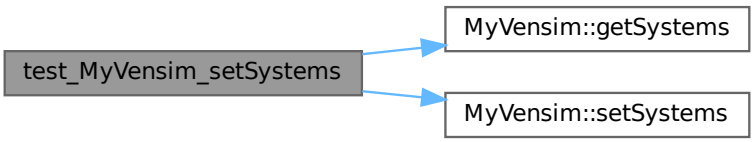
This function run the unit test of the MyVensim setSystems.

```
00097                                {
00098     std::cout « "        * setSystems tests\n";
00099
00100     System* s1 = new SystemIMP("s1");
00101     MyVensim* m1 = new MyVensimIMP("m1");
00102
00103     std::vector<System*> systems;
00104     systems.push_back(s1);
00105     m1->setSystems(systems);
00106
00107     assert(m1->getSystems() == systems);
00108
00109     delete s1;
00110     delete m1;
00111 }
```

References MyVensim::getSystems(), and MyVensim::setSystems().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



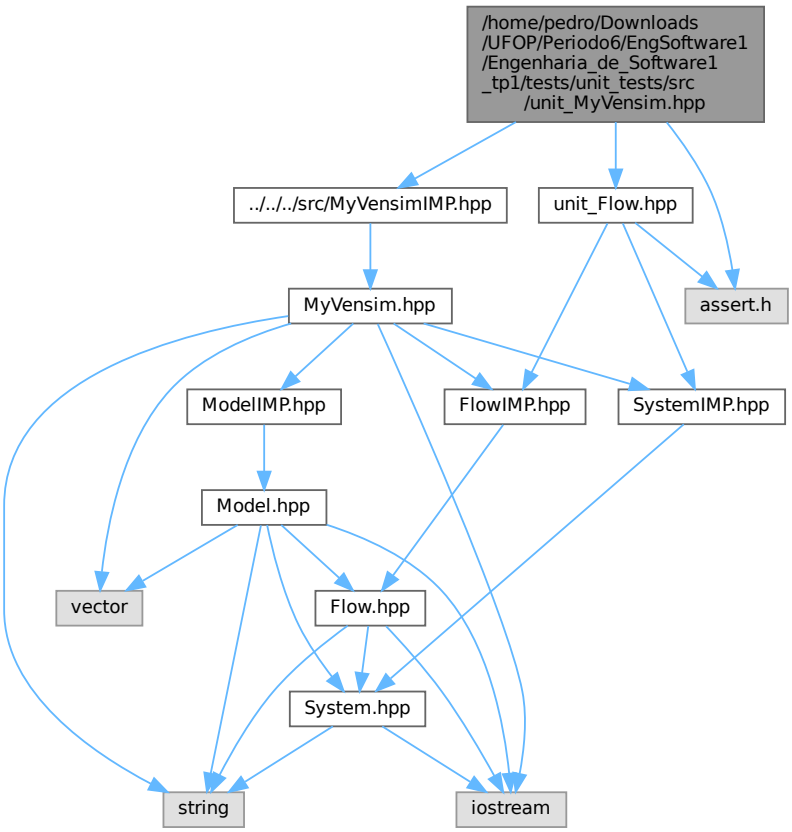
Here is the caller graph for this function:

5.39 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_My↩ Vensim.hpp File Reference

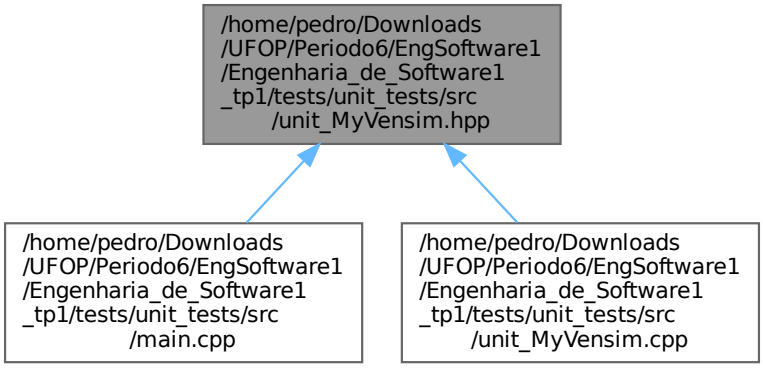
```
#include "../../../src/MyVensimIMP.hpp"
#include "unit_Flow.hpp"
```

```
#include <assert.h>
```

Include dependency graph for unit_MyVensim.hpp:



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



Functions

- void test_MyVensim_constructor ()

  This function run the unit test of the MyVensim constructor.
- void test_MyVensim_destructor ()

  This function run the unit test of the MyVensim destructor.
- void test_MyVensim_getName ()

  This function run the unit test of the MyVensim getName.
- void test_MyVensim_getSystems_and_add ()

  This function run the unit test of the MyVensim getSystems and add System.
- void test_MyVensim_getFlows_and_add ()

  This function run the unit test of the MyVensim getFlows and add Flow.
- void test_MyVensim_getModels_and_add ()

  This function run the unit test of the MyVensim getModels and add Model.
- void test_MyVensim_setName ()

  This function run the unit test of the MyVensim setName.
- void test_MyVensim_setSystems ()

  This function run the unit test of the MyVensim setSystems.
- void test_MyVensim_setFlows ()

  This function run the unit test of the MyVensim setFlows.
- void test_MyVensim_setModels ()

  This function run the unit test of the MyVensim setModels.
- void test_MyVensim_rmv_Sytem ()

  This function run the unit test of the MyVensim rmv System.
- void test_MyVensim_rmv_Flow ()

  This function run the unit test of the MyVensim rmv Flow.
- void test_MyVensim_rmv_Model ()

  This function run the unit test of the MyVensim rmv Model.
- void test_MyVensim_equal ()

  This function run the unit test of the MyVensim equal.
- void test_MyVensim_run ()

  This function run the unit test of the MyVensim run.
- void run_MyVensim_unit_test ()

  This function run the unit tests of the MyVensim.

5.39.1 Function Documentation

5.39.1.1 run_MyVensim_unit_test()

```
void run_MyVensim_unit_test ( )
```

This function run the unit tests of the MyVensim.

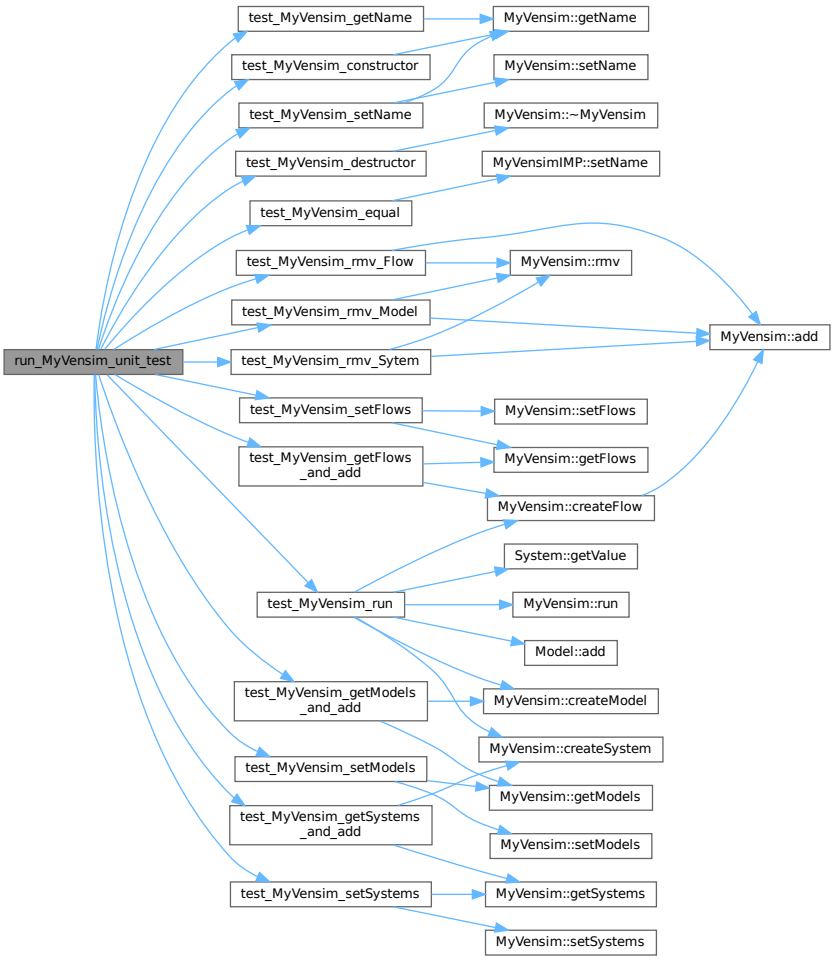
```
00222                                {
00223     std::cout « "    Start MyVensim unit tests\n";
00224     test_MyVensim_constructor();
00225     test_MyVensim_destructor();
00226     test_MyVensim_getName();
00227     test_MyVensim_getSystems_and_add();
00228     test_MyVensim_getFlows_and_add();
00229     test_MyVensim_getModels_and_add();
00230     test_MyVensim_setName();
00231     test_MyVensim_setSystems();
00232     test_MyVensim_setFlows();
00233     test_MyVensim_setModels();
00234     test_MyVensim_rmv_Sytem();
00235     test_MyVensim_rmv_Flow();
```

```
00236     test_MyVensim_rmv_Model();
00237     test_MyVensim_run();
00238     test_MyVensim_equal();
00239     std::cout « "    End MyVensim unit tests\n";
00240 }
```

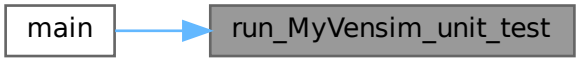
References test_MyVensim_constructor(), test_MyVensim_destructor(), test_MyVensim_equal(), test_MyVensim_getFlows_and_add(), test_MyVensim_getModels_and_add(), test_MyVensim_getName(), test_MyVensim_getSystems_and_add(), test_MyVensim_rmv_Flow(), test_MyVensim_rmv_Model(), test_MyVensim_rmv_Sytem(), test_MyVensim_run(), test_MyVensim_setFlows(), test_MyVensim_setModels(), test_MyVensim_setName(), and test_MyVensim_setSystems().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.2 test_MyVensim_constructor()

```
void test_MyVensim_constructor ( )
```

This function run the unit test of the MyVensim constructor.

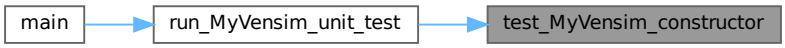
```
00003                                   {
00004     std::cout « "        * Constructor tests\n";
00005
00006     MyVensim* m1 = new MyVensimIMP();
00007     assert(m1->getName() == "NO_NAME");
00008
00009     MyVensim* m2 = new MyVensimIMP("m2");
00010     assert(m2->getName() == "m2");
00011
00012     delete m1;
00013     delete m2;
00014 }
```

References MyVensim::getName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:

Here is the caller graph for this function:



5.39.1.3 test_MyVensim_destructor()

```
void test_MyVensim_destructor ( )
```

This function run the unit test of the MyVensim destructor.

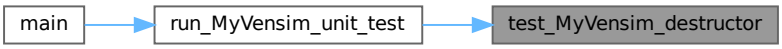
```
00016                                   {
00017     std::cout « "        * Destructor tests\n";
00018     MyVensim* m1 = new MyVensimIMP();
00019     m1->~MyVensim();
00020 }
```

References MyVensim::~MyVensim().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:

Here is the caller graph for this function:



5.39.1.4 test_MyVensim_equal()

```
void test_MyVensim_equal ( )
```

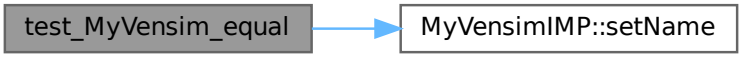
This function run the unit test of the MyVensim equal.

```
00144                                {
00145     std::cout « "        * Equal tests\n";
00146
00147     MyVensimIMP* m1 = new MyVensimIMP();
00148     MyVensimIMP* m2 = new MyVensimIMP();
00149     assert(*m1 == *m2);
00150
00151     m1->setName("m1");
00152     assert(m1 != m2);
00153
00154     delete m1;
00155     delete m2;
00156 }
```

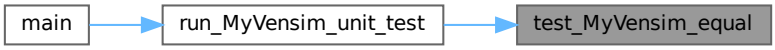
References MyVensimIMP::setName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.5 test_MyVensim_getFlows_and_add()

```
void test_MyVensim_getFlows_and_add ( )
```

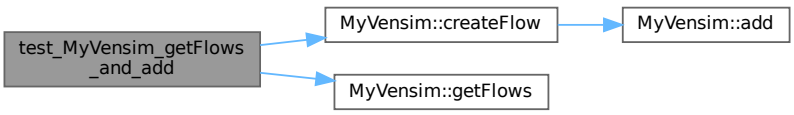
This function run the unit test of the MyVensim getFlows and add Flow.

```
00049                                       {
00050     std::cout « "        * getFlows and add Flows tests\n";
00051
00052     MyVensim* m1 = new MyVensimIMP("m1");
00053     Flow* f1 = m1->createFlow<Flow_unit_test>("f1");
00054
00055     std::vector<Flow*> flows;
00056     flows.push_back(f1);
00057
00058     assert(m1->getFlows() == flows);
00059
00060     delete f1;
00061     delete m1;
00062 }
```

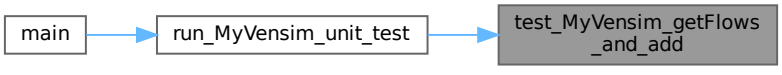
References MyVensim::createFlow(), and MyVensim::getFlows().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.6 test_MyVensim_getModels_and_add()

```
void test_MyVensim_getModels_and_add ( )
```

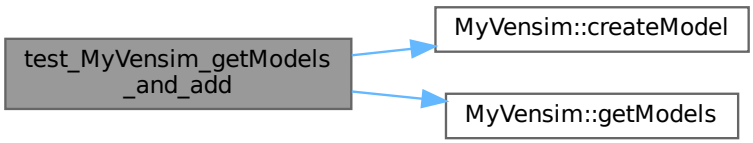
This function run the unit test of the MyVensim getModels and add Model.

```
00064                                       {
00065     std::cout « "        * getModels and add Models tests\n";
00066
00067     MyVensim* m1 = new MyVensimIMP("m1");
00068     Model* f1 = m1->createModel("f1");
00069
00070     std::vector<Model*> Models;
00071     Models.push_back(f1);
00072
00073     assert(m1->getModels() == Models);
00074
00075     delete f1;
00076     delete m1;
00077 }
```

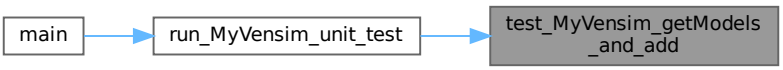
References MyVensim::createModel(), and MyVensim::getModels().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.7 test_MyVensim_getName()

```
void test_MyVensim_getName ( )
```

This function run the unit test of the MyVensim getName.

```
00022                                   {
00023     std::cout « "        * getName tests\n";
00024     MyVensim* m1 = new MyVensimIMP();
00025     assert(m1->getName() == "NO_NAME");
00026
00027     MyVensim* m2 = new MyVensimIMP("m2");
00028     assert(m2->getName() == "m2");
00029
```

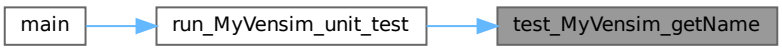
```
00030     delete m1;
00031     delete m2;
00032 }
```

References MyVensim::getName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:

Here is the caller graph for this function:



5.39.1.8 test_MyVensim_getSystems_and_add()

```
void test_MyVensim_getSystems_and_add ( )
```

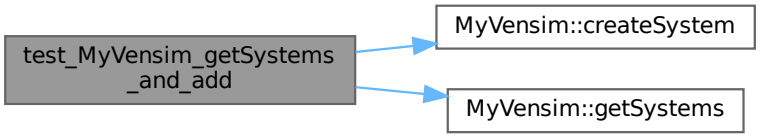
This function run the unit test of the MyVensim getSystems and add System.

```
00034                                       {
00035     std::cout « "        * getSystems and add Systems tests\n";
00036
00037     MyVensim* m1 = new MyVensimIMP("m1");
00038     System* s1 = m1->createSystem("s1");
00039
00040     std::vector<System*> systems;
00041     systems.push_back(s1);
00042
00043     assert(m1->getSystems() == systems);
00044
00045     delete s1;
00046     delete m1;
00047 }
```

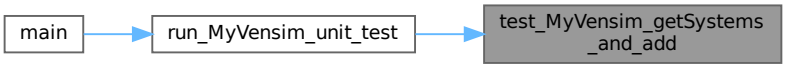
References MyVensim::createSystem(), and MyVensim::getSystems().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.9 test_MyVensim_rmv_Flow()

```
void test_MyVensim_rmv_Flow ( )
```

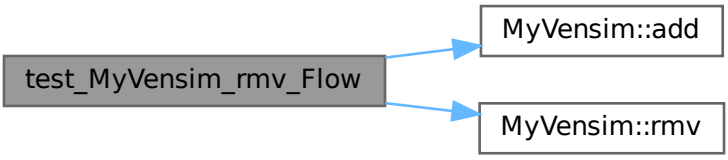
This function run the unit test of the MyVensim rmv Flow.

```
00171                                {
00172     std::cout « "        * Remove Flow tests\n";
00173
00174     MyVensim* MyVensim1 = new MyVensimIMP();
00175     FlowIMP* flow1 = new Flow_unit_test("flow1");
00176
00177     MyVensim1->add(flow1);
00178     assert(MyVensim1->rmv(flow1));
00179
00180     delete MyVensim1;
00181     delete flow1;
00182 }
```

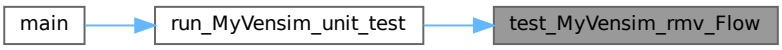
References MyVensim::add(), and MyVensim::rmv().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.10 test_MyVensim_rmv_Model()

```
void test_MyVensim_rmv_Model ( )
```

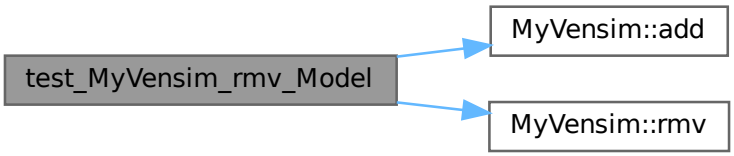
This function run the unit test of the MyVensim rmv Model.

```
00184                                {
00185     std::cout « "        * Remove Model tests\n";
00186
00187     MyVensim* MyVensim1 = new MyVensimIMP();
00188     ModelIMP* Model1 = new ModelIMP("Model1");
00189
00190     MyVensim1->add(Model1);
00191     assert(MyVensim1->rmv(Model1));
00192
00193     delete MyVensim1;
00194     delete Model1;
00195 }
```

References MyVensim::add(), and MyVensim::rmv().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:

5.39.1.11 test_MyVensim_rmv_Sytem()

```
void test_MyVensim_rmv_Sytem ( )
```

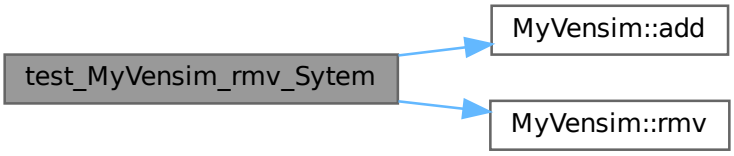
This function run the unit test of the MyVensim rmv System.

```
00158                                {
00159     std::cout « "        * Remove System tests\n";
00160
00161     MyVensim* MyVensim1 = new MyVensimIMP();
00162     System* system1 = new SystemIMP("system1");
00163
00164     MyVensim1->add(system1);
00165     assert(MyVensim1->rmv(system1));
00166
00167     delete MyVensim1;
00168     delete system1;
00169 }
```

References MyVensim::add(), and MyVensim::rmv().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:

5.39.1.12 test_MyVensim_run()

```
void test_MyVensim_run ( )
```

This function run the unit test of the MyVensim run.

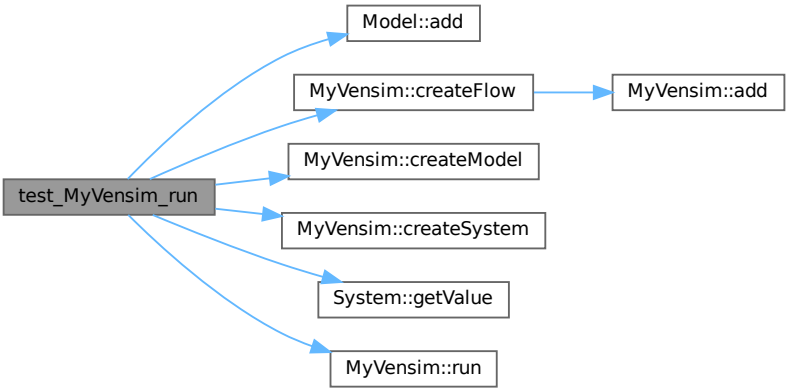
```
00197                               {
00198     std::cout « "        * Run tests\n";
00199
00200     MyVensim* mv = new MyVensimIMP("mv");
00201     System* s1 = mv->createSystem("s1", 100);
00202     System* s2 = mv->createSystem("s2", 0.0);
00203     Flow* f1 = mv->createFlow<Flow_unit_test>("f1", s1, s2);
00204     Model* m1 = mv->createModel("m1", 0, 1);
00205
00206     m1->add(s1);
```

```
00207     m1->add(s2);
00208     m1->add(f1);
00209
00210     mv->run();
00211
00212     assert(s1->getValue() == 0);
00213     assert(s2->getValue() == 100);
00214
00215     delete mv;
00216     delete s1;
00217     delete s2;
00218     delete f1;
00219     delete m1;
00220 }
```

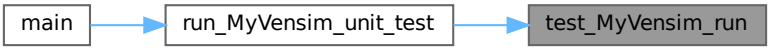
References Model::add(), MyVensim::createFlow(), MyVensim::createModel(), MyVensim::createSystem(), System::getValue(), and MyVensim::run().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.13 test_MyVensim_setFlows()

```
void test_MyVensim_setFlows ( )
```

This function run the unit test of the MyVensim setFlows.

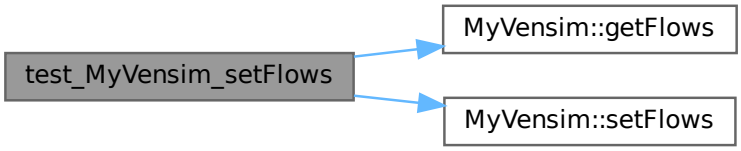
```
00113                                 {
00114     std::cout « "        * setFlows tests\n";
00115     MyVensim* m1 = new MyVensimIMP("m1");
00116     Flow* f1 = new Flow_unit_test("f1");
00117
```

```
00118     std::vector<Flow*> flows;
00119     flows.push_back(f1);
00120     m1->setFlows(flows);
00121
00122     assert(m1->getFlows() == flows);
00123
00124     delete f1;
00125     delete m1;
00126 }
```

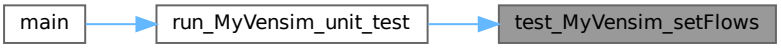
References MyVensim::getFlows(), and MyVensim::setFlows().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.14 test_MyVensim_setModels()

```
void test_MyVensim_setModels ( )
```

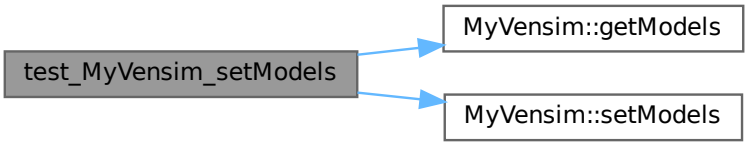
This function run the unit test of the MyVensim setModels.

```
00128                                {
00129     std::cout « "        * setModels tests\n";
00130
00131     Model* f1 = new ModelIMP("f1");
00132     MyVensim* m1 = new MyVensimIMP("m1");
00133
00134     std::vector<Model*> Models;
00135     Models.push_back(f1);
00136     m1->setModels(Models);
00137
00138     assert(m1->getModels() == Models);
00139
00140     delete f1;
00141     delete m1;
00142 }
```

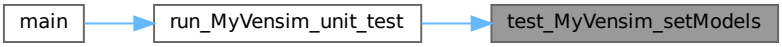
References MyVensim::getModels(), and MyVensim::setModels().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.15 test_MyVensim_setName()

```
void test_MyVensim_setName ( )
```

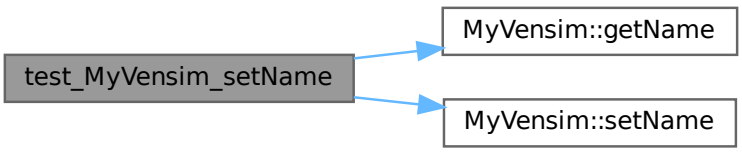
This function run the unit test of the MyVensim setName.

```
00079                                {
00080     std::cout « "        * setName tests\n";
00081
00082     MyVensim* m1 = new MyVensimIMP();
00083
00084     m1->setName("m1");
00085     assert(m1->getName() != "NO_NAME");
00086
00087     MyVensim* m2 = new MyVensimIMP("m");
00088
00089     m2->setName("m2");
00090
00091     assert(m2->getName() == "m2");
00092
00093     delete m1;
00094     delete m2;
00095 }
```

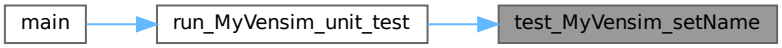
References MyVensim::getName(), and MyVensim::setName().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.1.16 test_MyVensim_setSystems()

```
void test_MyVensim_setSystems ( )
```

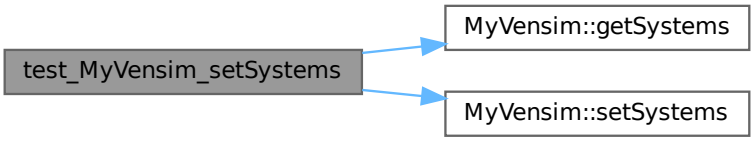
This function run the unit test of the MyVensim setSystems.

```
00097                                {
00098     std::cout « "        * setSystems tests\n";
00099
00100     System* s1 = new SystemIMP("s1");
00101     MyVensim* m1 = new MyVensimIMP("m1");
00102
00103     std::vector<System*> systems;
00104     systems.push_back(s1);
00105     m1->setSystems(systems);
00106
00107     assert(m1->getSystems() == systems);
00108
00109     delete s1;
00110     delete m1;
00111 }
```

References MyVensim::getSystems(), and MyVensim::setSystems().

Referenced by run_MyVensim_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:

5.40 unit_MyVensim.hpp

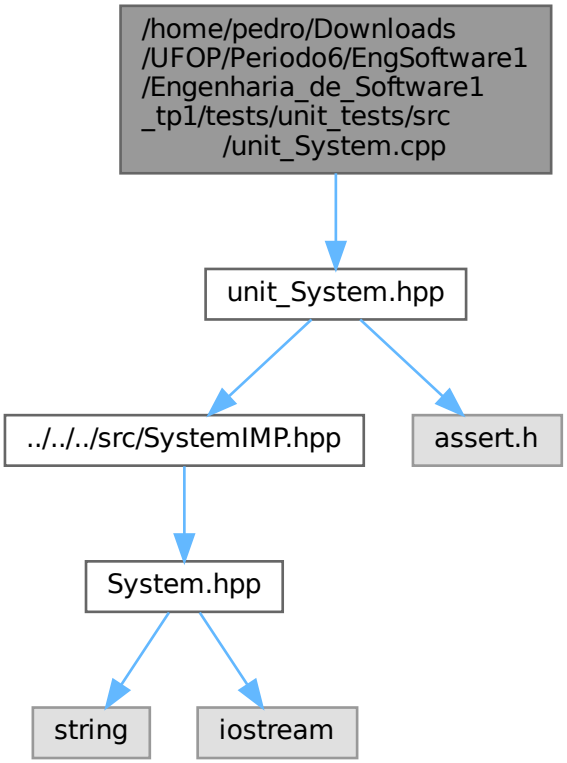
Go to the documentation of this file.

```
00001 /************************************************************
00002  * @file unit_MyVensim.hpp
00003  * @author Pedro Augusto Sousa Gonçalves
00004  * @brief This file represents the MyVensim units tests
00005 ************************************************************/
00006
00007 #ifndef UNIT_MYVENSIM_HPP
00008 #define UNIT_MYVENSIM_HPP
00009
00010 #include "../../../src/MyVensimIMP.hpp"
00011 #include "unit_Flow.hpp"
00012
00013 #include <assert.h>
00014
00018 void test_MyVensim_constructor();
00022 void test_MyVensim_destructor();
00026 void test_MyVensim_getName();
00030 void test_MyVensim_getSystems_and_add();
00034 void test_MyVensim_getFlows_and_add();
00038 void test_MyVensim_getModels_and_add();
00042 void test_MyVensim_setName();
00046 void test_MyVensim_setSystems();
00050 void test_MyVensim_setFlows();
00054 void test_MyVensim_setModels();
00058 void test_MyVensim_rmv_Sytem();
00062 void test_MyVensim_rmv_Flow();
00066 void test_MyVensim_rmv_Model();
00070 void test_MyVensim_equal();
00074 void test_MyVensim_run();
00078 void run_MyVensim_unit_test();
00079
00080
00081 #endif
```

5.41 /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 comparation.
- void run_System_unit_test ()

  This function run the unit tests of the system.

5.41.1 Function Documentation

5.41.1.1 run_System_unit_test()

```
void run_System_unit_test ( )
```

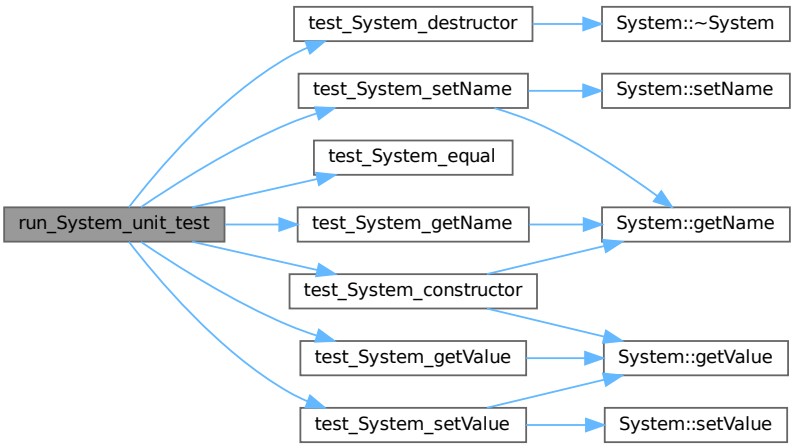
This function run the unit tests of the system.

```
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:

5.41.1.2 test_System_constructor()

```
void test_System_constructor ( )
```

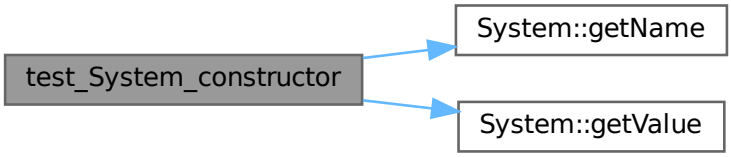
This function run the unit test of the system constructor.

```
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 }
```

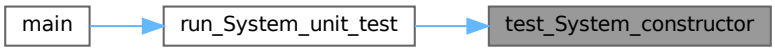
References System::getName(), and System::getValue().

Referenced by run_System_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.41.1.3 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();
```

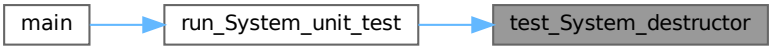
```
00025     s1->~System();
00026 }
```

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:



5.41.1.4 test_System_equal()

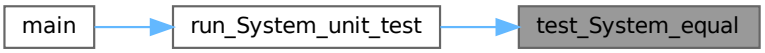
```
void test_System_equal ( )
```

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

```
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:



5.41.1.5 test_System_getName()

```
void test_System_getName ( )
```

This function run the unit test of the system getName.

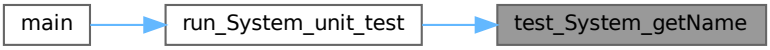
```
00028                                {
00029     std::cout « "        * getName tests\n";
00030     System* s1 = new SystemIMP();
00031     assert(s1->getName() == "NO_NAME");
00032
00033     System* s2 = new SystemIMP("s2");
00034     assert(s2->getName() == "s2");
00035
00036     delete s1;
00037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



5.41.1.6 test_System_getValue()

```
void test_System_getValue ( )
```

This function run the unit test of the system getValue.

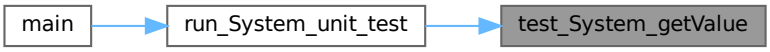
```
00040                                 {
00041     std::cout « "        * getValue tests\n";
00042     System* s1 = new SystemIMP();
00043     assert(s1->getValue() == 0);
00044
00045     System* s2 = new SystemIMP("s2", 22);
00046     assert(s2->getValue() == 22);
00047
00048     delete s1;
00049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



5.41.1.7 test_System_setName()

```
void test_System_setName ( )
```

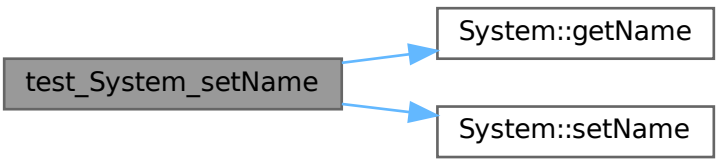
This function run the unit test of the system setName.

```
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 }
```

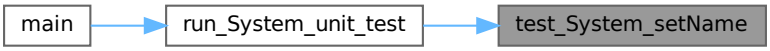
References System::getName(), and System::setName().

Referenced by run_System_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.41.1.8 test_System_setValue()

```
void test_System_setValue ( )
```

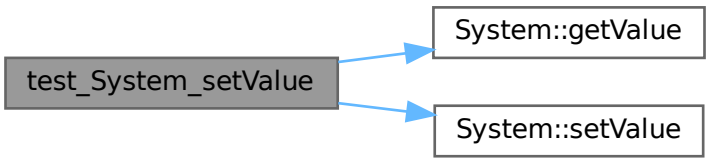
This function run the unit test of the system setValeu.

```
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 }
```

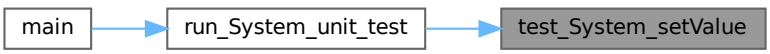
References System::getValue(), and System::setValue().

Referenced by run_System_unit_test().

Here is the call graph for this function:



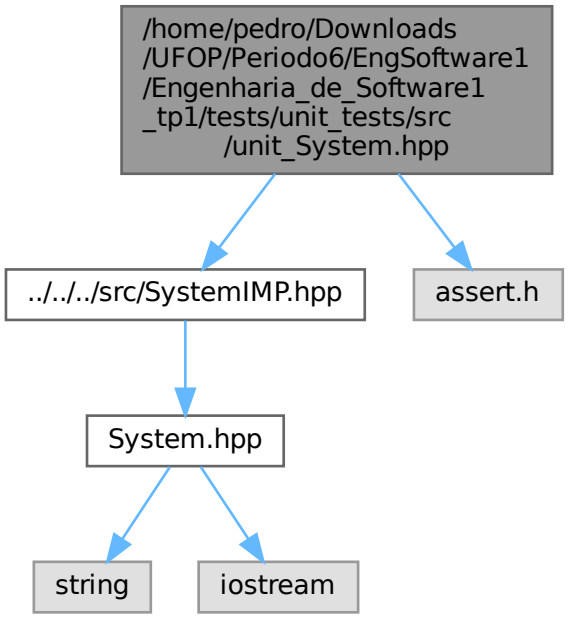
Here is the caller graph for this function:



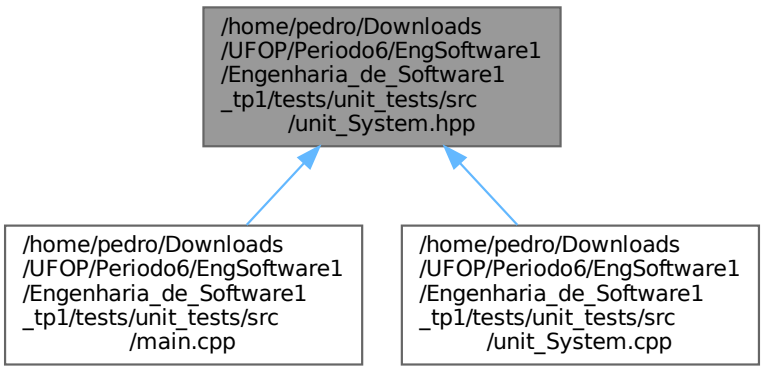
5.42 /home/pedro/Downloads/UFOP/Periodo6/EngSoftware1/↩ Engenharia_de_Software1_tp1/tests/unit_tests/src/unit_↩ System.hpp File Reference

```
#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 comparation.
- void run_System_unit_test ()

  This function run the unit tests of the system.

5.42.1 Function Documentation

5.42.1.1 run_System_unit_test()

```
void run_System_unit_test ( )
```

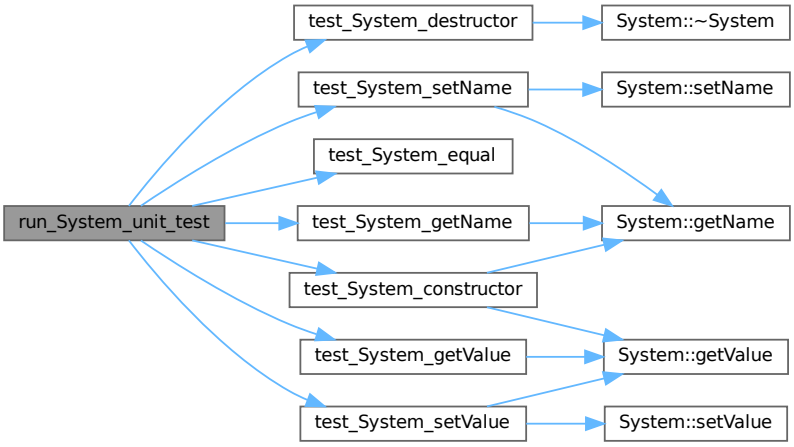
This function run the unit tests of the system.

```
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:

5.42.1.2 test_System_constructor()

```
void test_System_constructor ( )
```

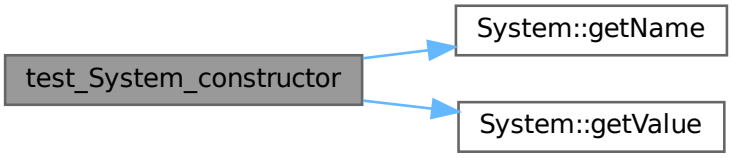
This function run the unit test of the system constructor.

```
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 }
```

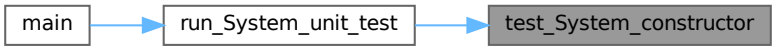
References System::getName(), and System::getValue().

Referenced by run_System_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.42.1.3 test_System_destructor()

```
void test_System_destructor ( )
```

This function run the unit test of the system destructor.

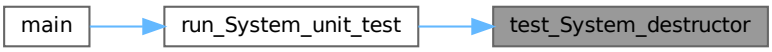
```
00022                                {
00023     std::cout « "        * Destructor tests\n";
00024     System* s1 = new SystemIMP();
00025     s1->~System();
00026 }
```

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:



5.42.1.4 test_System_equal()

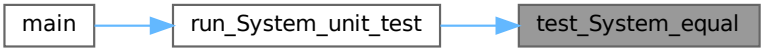
```
void test_System_equal ( )
```

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

```
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:



5.42.1.5 test_System_getName()

```
void test_System_getName ( )
```

This function run the unit test of the system getName.

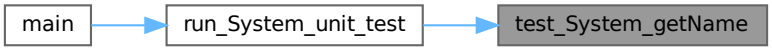
```
00028                            {
00029     std::cout « "         * getName tests\n";
00030     System* s1 = new SystemIMP();
00031     assert(s1->getName() == "NO_NAME");
00032
00033     System* s2 = new SystemIMP("s2");
00034     assert(s2->getName() == "s2");
00035
00036     delete s1;
00037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



5.42.1.6 test_System_getValue()

```
void test_System_getValue ( )
```

This function run the unit test of the system getValue.

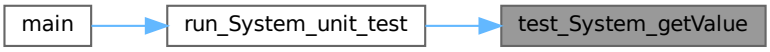
```
00040                                {
00041     std::cout « "        * getValue tests\n";
00042     System* s1 = new SystemIMP();
00043     assert(s1->getValue() == 0);
00044
00045     System* s2 = new SystemIMP("s2", 22);
00046     assert(s2->getValue() == 22);
00047
00048     delete s1;
00049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



5.42.1.7 test_System_setName()

```
void test_System_setName ( )
```

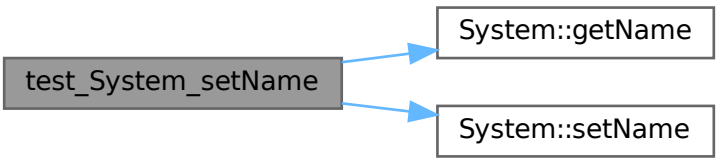
This function run the unit test of the system setName.

```
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 }
```

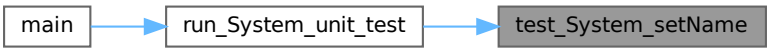
References System::getName(), and System::setName().

Referenced by run_System_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.42.1.8 test_System_setValue()

```
void test_System_setValue ( )
```

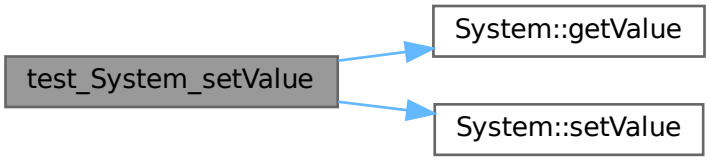
This function run the unit test of the system setValeu.

```
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 }
```

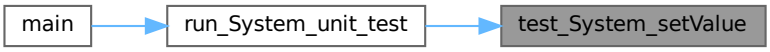
References System::getValue(), and System::setValue().

Referenced by run_System_unit_test().

Here is the call graph for this function:



Here is the caller graph for this function:



5.43 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
```