MyVensin

Generated by Doxygen 1.9.3

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

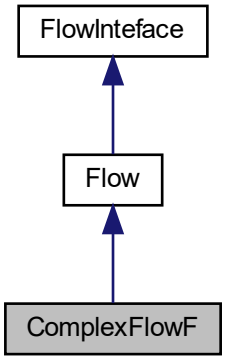
Class Documentation

4.1 ComplexFlowF Class Reference

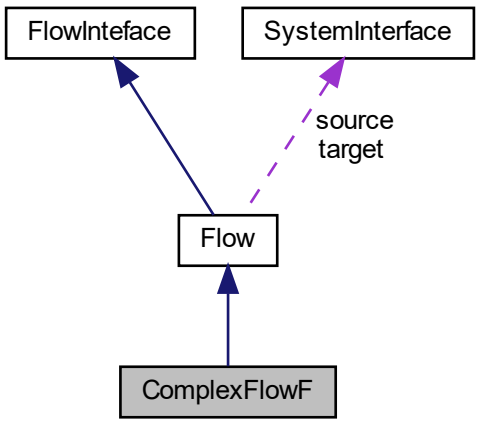
Class ComplexFlowF.

`#include <funcional_tests.h>`

Inheritance diagram for ComplexFlowF:



Collaboration diagram for ComplexFlowF:



Public Member Functions

- ComplexFlowF (string name, SystemInterface *source, SystemInterface *target)
- double execute ()

Additional Inherited Members

4.1.1 Detailed Description

Class ComplexFlowF.

This Class represents a flow with limitless growth, and is used in the Complex Model test.

Definition at line 77 of file funcional_tests.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 ComplexFlowF()

```
ComplexFlowF::ComplexFlowF (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the ComplexFlowF Class.

Parameters

name	the name of the ComplexFlowF Class.
source	a pointer to the source system of the ComplexFlowF Class.
target	a pointer to the target system of the ComplexFlowF Class.

Definition at line 85 of file funcional_tests.h.

4.1.3 Member Function Documentation

4.1.3.1 execute()

```
double ComplexFlowF::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the complex model.

Implements Flow.

Definition at line 91 of file funcional_tests.h.

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

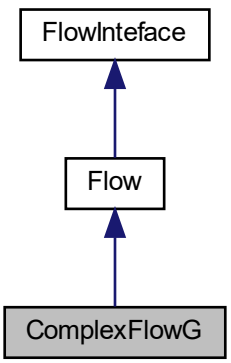
- test/funcional/funcional_tests.h

4.2 ComplexFlowG Class Reference

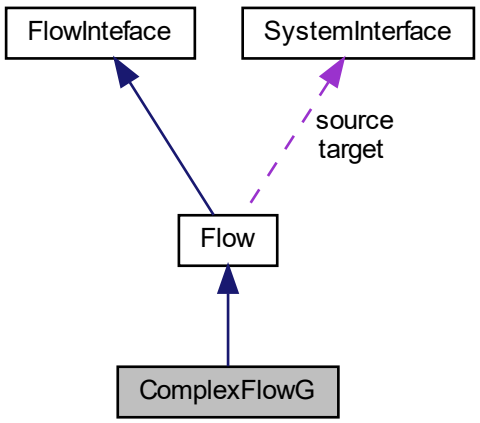
Class ComplexFlowG.

```
#include <funcional_tests.h>
```

Inheritance diagram for ComplexFlowG:



Collaboration diagram for ComplexFlowG:



Public Member Functions

- ComplexFlowG (string name, SystemInterface *source, SystemInterface *target)
- double execute ()

Additional Inherited Members

4.2.1 Detailed Description

Class ComplexFlowG.

This Class represents a flow with limitless growth, and is used in the Complex Model test.

Definition at line 183 of file funcional_tests.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ComplexFlowG()

```
ComplexFlowG::ComplexFlowG (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the ComplexFlowG Class.

Parameters

name	the name of the ComplexFlowG Class.
source	a pointer to the source system of the ComplexFlowG Class.
target	a pointer to the target system of the ComplexFlowG Class.

Definition at line 191 of file funcional_tests.h.

4.2.3 Member Function Documentation

4.2.3.1 execute()

```
double ComplexFlowG::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the complex model.

Implements Flow.

Definition at line 197 of file funcional_tests.h.

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

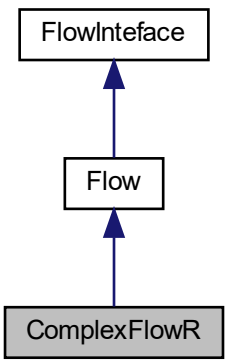
- test/funcional/funcional_tests.h

4.3 ComplexFlowR Class Reference

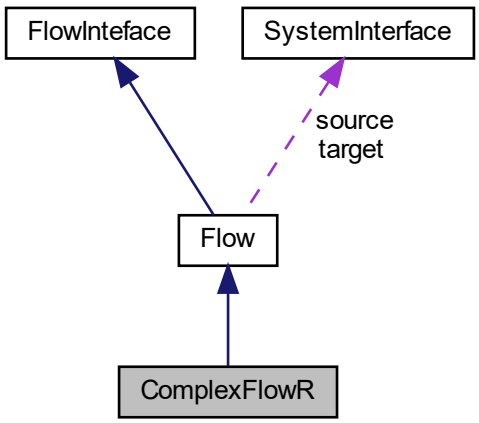
Class ComplexFlowR.

```
#include <funcional_tests.h>
```

Inheritance diagram for ComplexFlowR:



Collaboration diagram for ComplexFlowR:



Public Member Functions

- ComplexFlowR (string name, SystemInterface *source, SystemInterface *target)
- double execute ()

Additional Inherited Members

4.3.1 Detailed Description

Class ComplexFlowR.

This Class represents a flow with limitless growth, and is used in the Complex Model test.

Definition at line 210 of file funcional_tests.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ComplexFlowR()

```
ComplexFlowR::ComplexFlowR (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the ComplexFlowR Class.

Parameters

name	the name of the ComplexFlowR Class.
source	a pointer to the source system of the ComplexFlowR Class.
target	a pointer to the target system of the ComplexFlowR Class.

Definition at line 218 of file funcional_tests.h.

4.3.3 Member Function Documentation

4.3.3.1 execute()

```
double ComplexFlowR::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the complex model.

Implements Flow.

Definition at line 224 of file funcional_tests.h.

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

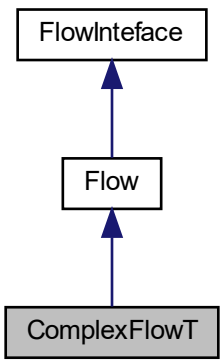
- test/funcional/funcional_tests.h

4.4 ComplexFlowT Class Reference

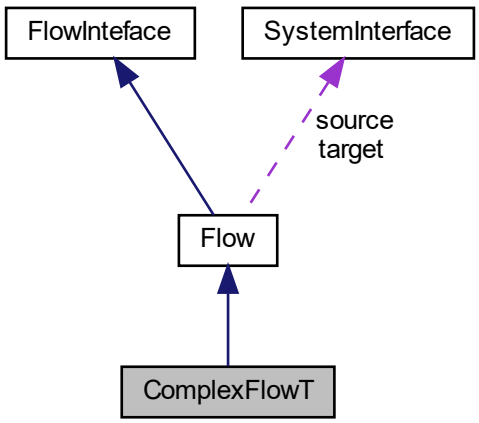
Class ComplexFlowT.

```
#include <funcional_tests.h>
```

Inheritance diagram for ComplexFlowT:



Collaboration diagram for ComplexFlowT:



Public Member Functions

- ComplexFlowT (string name, SystemInterface *source, SystemInterface *target)
- double execute ()

Additional Inherited Members

4.4.1 Detailed Description

Class ComplexFlowT.

This Class represents a flow with limitless growth, and is used in the Complex Model test.

Definition at line 103 of file funcional_tests.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ComplexFlowT()

```
ComplexFlowT::ComplexFlowT (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the ComplexFlowT Class.

Parameters

name	the name of the ComplexFlowT Class.
source	a pointer to the source system of the ComplexFlowT Class.
target	a pointer to the target system of the ComplexFlowT Class.

Definition at line 111 of file funcional_tests.h.

4.4.3 Member Function Documentation

4.4.3.1 execute()

```
double ComplexFlowT::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the complex model.

Implements Flow.

Definition at line 117 of file funcional_tests.h.

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

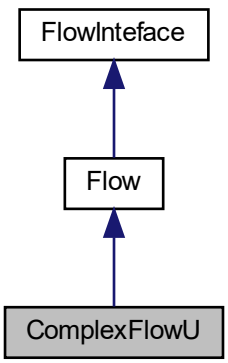
- test/funcional/funcional_tests.h

4.5 ComplexFlowU Class Reference

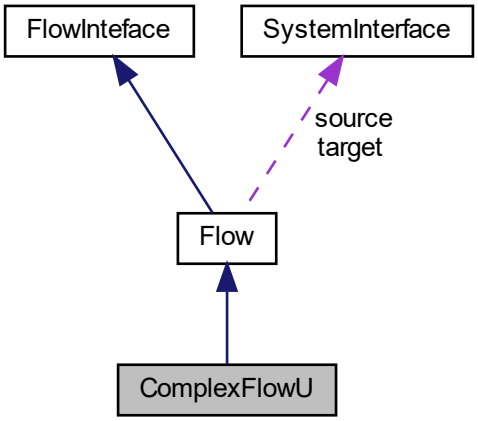
Class ComplexFlowU.

```
#include <funcional_tests.h>
```

Inheritance diagram for ComplexFlowU:



Collaboration diagram for ComplexFlowU:



Public Member Functions

- ComplexFlowU (string name, SystemInterface *source, SystemInterface *target)
- double execute ()

Additional Inherited Members

4.5.1 Detailed Description

Class ComplexFlowU.

This Class represents a flow with limitless growth, and is used in the Complex Model test.

Definition at line 129 of file funcional_tests.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ComplexFlowU()

```
ComplexFlowU::ComplexFlowU (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the ComplexFlowU Class.

Parameters

name	the name of the ComplexFlowU Class.
source	a pointer to the source system of the ComplexFlowU Class.
target	a pointer to the target system of the ComplexFlowU Class.

Definition at line 137 of file funcional_tests.h.

4.5.3 Member Function Documentation

4.5.3.1 execute()

```
double ComplexFlowU::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the complex model.

Implements Flow.

Definition at line 143 of file funcional_tests.h.

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

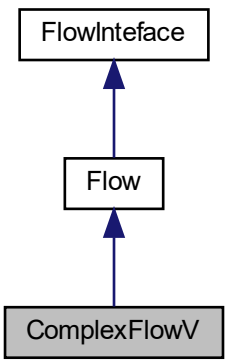
- test/funcional/funcional_tests.h

4.6 ComplexFlowV Class Reference

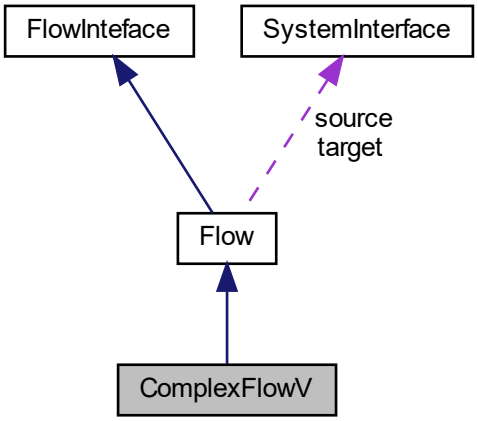
Class ComplexFlowV.

```
#include <funcional_tests.h>
```

Inheritance diagram for ComplexFlowV:



Collaboration diagram for ComplexFlowV:



Public Member Functions

- ComplexFlowV (string name, SystemInterface *source, SystemInterface *target)
- double execute ()

Additional Inherited Members

4.6.1 Detailed Description

Class ComplexFlowV.

This Class represents a flow with limitless growth, and is used in the Complex Model test.

Definition at line 156 of file funcional_tests.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ComplexFlowV()

```
ComplexFlowV::ComplexFlowV (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the ComplexFlowV Class.

Parameters

name	the name of the ComplexFlowV Class.
source	a pointer to the source system of the ComplexFlowV Class.
target	a pointer to the target system of the ComplexFlowV Class.

Definition at line 164 of file funcional_tests.h.

4.6.3 Member Function Documentation

4.6.3.1 execute()

```
double ComplexFlowV::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the complex model.

Implements Flow.

Definition at line 170 of file funcional_tests.h.

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

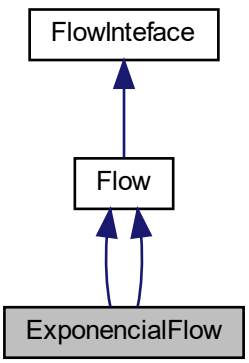
- test/funcional/funcional_tests.h

4.7 ExponencialFlow Class Reference

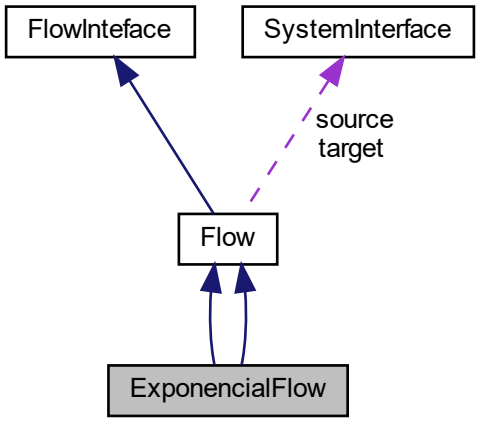
Functional tests.

```
#include <funcional_tests.h>
```

Inheritance diagram for ExponencialFlow:



Collaboration diagram for ExponencialFlow:



Public Member Functions

- ExponencialFlow (string name, SystemInterface *source, SystemInterface *target)
- double execute ()
- ExponencialFlow (string name="", System *source=NULL, System *target=NULL)
- double execute ()

Additional Inherited Members

4.7.1 Detailed Description

Functional tests.

Class ExponencialFlow.

Implementation of the functional tests for the Flow, System and Model classes. Class ExponencialFlow This Class represents a flow with limitless growth, and is used in the Exponencial Model test.

This Class represents a flow with limitless growth, and is used in the Exponencial Model test.

Definition at line 22 of file funcional_tests.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 ExponencialFlow() [1/2]

```
ExponencialFlow::ExponencialFlow (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the ExponencialFlow Class.

Parameters

name	the name of the ExponencialFlow Class.
source	a pointer to the source system of the ExponencialFlow Class.
target	a pointer to the target system of the ExponencialFlow Class.

Definition at line 30 of file funcional_tests.h.

4.7.2.2 ExponencialFlow() [2/2]

```
ExponencialFlow::ExponencialFlow (
            string name = "",
            System * source = NULL,
            System * target = NULL )  [inline]
```

This is a more elaborated constructor for the ExponencialFlow Class.

Parameters

name	the name of the ExponencialFlow Class.
source	a pointer to the source system of the ExponencialFlow Class.
target	a pointer to the target system of the ExponencialFlow Class.

Definition at line 48 of file unit_flow.h.

4.7.3 Member Function Documentation

4.7.3.1 execute() [1/2]

```
double ExponencialFlow::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the exponencial model.

Implements Flow.

Definition at line 36 of file funcional_tests.h.

4.7.3.2 execute() [2/2]

```
double ExponencialFlow::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the exponencial model.

Implements Flow.

Definition at line 53 of file unit_flow.h.

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

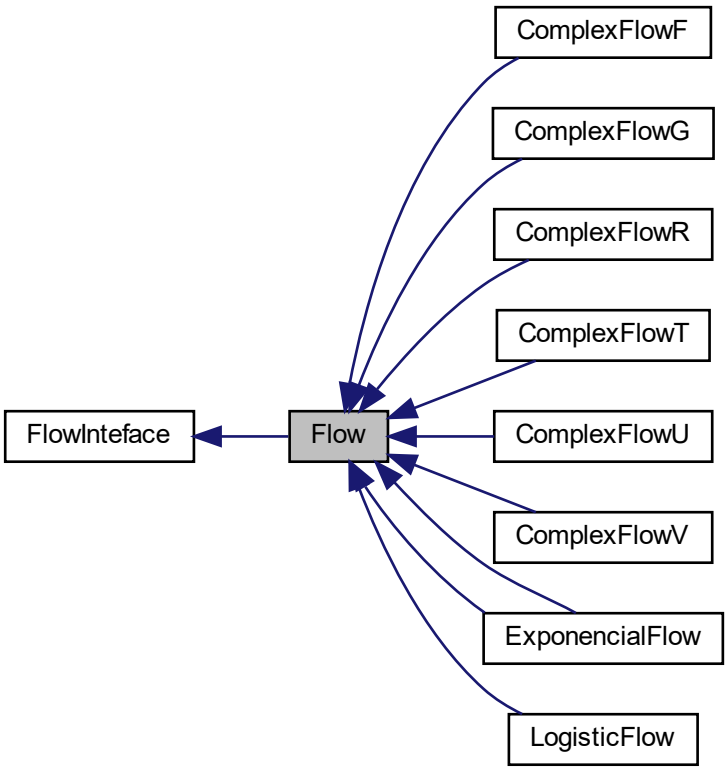
- test/funcional/funcional_tests.h
- test/unit/unit_flow.h

4.8 Flow Class Reference

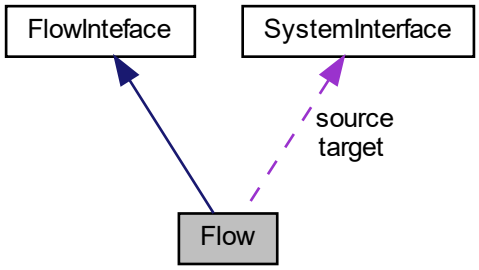
Class Flow.

```
#include <flow.h>
```

Inheritance diagram for Flow:



Collaboration diagram for Flow:



Public Member Functions

- Flow (string name="", SystemInterface *source=NULL, SystemInterface *target=NULL)
- virtual ~Flow ()
- virtual double execute ()=0
- string getName () const
- SystemInterface * getSource () const
- SystemInterface * getTarget () const
- void setName (string flowName)
- void setSource (SystemInterface *sourceSys)
- void setTarget (SystemInterface *targetSys)
- void clearSource ()
- void clearTarget ()

Protected Member Functions

- Flow (const FlowInteface &flow)
- Flow & operator= (const FlowInteface &flow)

Protected Attributes

- string name
- SystemInterface * source
- SystemInterface * target

Friends

- class ModelInterface
- class Model
- class UnitFlow

4.8.1 Detailed Description

Class Flow.

This Class represents a flow in the General Systems Theory implemented in this code.

Definition at line 11 of file flow.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Flow() [1/2]

```
Flow::Flow (
            const FlowInteface & flow )  [protected]
```

This is the copy constructor for the Flow Class.

Parameters

flow	the flow that is going to be cloned.

Definition at line 3 of file flow.cpp.

4.8.2.2 Flow() [2/2]

```
Flow::Flow (
            string name = "",
            SystemInterface * source = NULL,
            SystemInterface * target = NULL )
```

This is the default constructor for the Flow Class.

Parameters

name	the name of the Flow.
source	a pointer to the source system of the Flow.
target	a pointer to the target system of the Flow.

Returns

Flow - a Flow Class object, with it's isCopy attribute set to false.

Definition at line 21 of file flow.cpp.

4.8.2.3 ~Flow()

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

This is the default destructor for the Flow Class.

Definition at line 24 of file flow.cpp.

4.8.3 Member Function Documentation

4.8.3.1 clearSource()

```
void Flow::clearSource ( )  [virtual]
```

Sets the pointer of the source attribute as NULL.

Implements FlowInteface.

Definition at line 50 of file flow.cpp.

4.8.3.2 clearTarget()

```
void Flow::clearTarget ( )  [virtual]
```

Sets the pointer of the target attribute as NULL.

Implements FlowInteface.

Definition at line 54 of file flow.cpp.

4.8.3.3 execute()

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

A pure virtual method that will be inherited by subclasses created by the user, and that will contain an equation that will be executed by the model.

Implements FlowInteface.

Implemented in ExponencialFlow, LogisticFlow, ComplexFlowF, ComplexFlowT, ComplexFlowU, ComplexFlowV, ComplexFlowG, ComplexFlowR, and ExponencialFlow.

4.8.3.4 getName()

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

Returns the name attribute in the Flow Class.

Returns

string - the content name attribute.

Implements FlowInteface.

Definition at line 26 of file flow.cpp.

4.8.3.5 getSource()

```
SystemInterface * Flow::getSource ( ) const  [virtual]
```

Returns the source attribute in the Flow Class.

Returns

System* - the pointer of the source system.

Implements FlowInteface.

Definition at line 30 of file flow.cpp.

4.8.3.6 getTarget()

```
SystemInterface * Flow::getTarget ( ) const  [virtual]
```

Returns the target attribute in the Flow Class.

Returns

System* - the pointer of the target system.

Implements FlowInteface.

Definition at line 34 of file flow.cpp.

4.8.3.7 operator=()

```
Flow & Flow::operator= (
            const FlowInteface & flow )  [protected]
```

This is the overloaded assignment operator for the Flow Class.

Definition at line 11 of file flow.cpp.

4.8.3.8 setName()

```
void Flow::setName (
            string flowName )  [virtual]
```

Sets the name attribute in the Flow Class.

Parameters

flowName	which will be set to the current flow.

Implements FlowInteface.

Definition at line 38 of file flow.cpp.

4.8.3.9 setSource()

```
void Flow::setSource (
            SystemInterface * sourceSys )  [virtual]
```

Sets the source attribute in the Flow Class.

Parameters

sourceSys	a pointer to the source target.

Implements FlowInteface.

Definition at line 42 of file flow.cpp.

4.8.3.10 setTarget()

```
void Flow::setTarget (
            SystemInterface * targetSys )  [virtual]
```

Sets the target attribute in the Flow Class.

Parameters

targetSys	a pointer to the target system.

Implements FlowInteface.

Definition at line 46 of file flow.cpp.

4.8.4 Friends And Related Function Documentation

4.8.4.1 Model

```
friend class Model  [friend]
```

This Class represents the implementation of a model in the General Systems Theory implemented in this code.

Definition at line 30 of file flow.h.

4.8.4.2 ModelInterface

```
friend class ModelInterface  [friend]
```

This Class represents a model in the General Systems Theory implemented in this code.

Definition at line 29 of file flow.h.

4.8.4.3 UnitFlow

```
friend class UnitFlow  [friend]
```

This Class is used to test the copy constructor and assignment operator of the Flow class.

Definition at line 31 of file flow.h.

4.8.5 Member Data Documentation

4.8.5.1 name

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

This attribute contains a name for the flow.

Definition at line 13 of file flow.h.

4.8.5.2 source

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

This attribute stores a pointer to the source system of a flow.

Definition at line 14 of file flow.h.

4.8.5.3 target

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

This attribute stores a pointer to the target system of a flow.

Definition at line 15 of file flow.h.

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

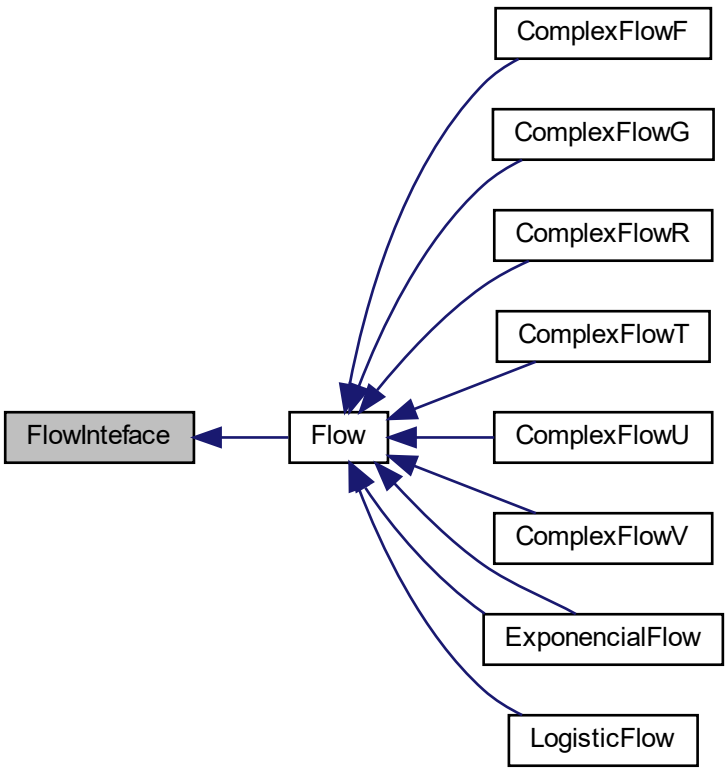
- src/flow.h
- src/flow.cpp

4.9 FlowInteface Class Reference

Class FlowInterface.

```
#include <flowinterface.h>
```

Inheritance diagram for FlowInteface:



Public Member Functions

- virtual ~FlowInteface ()
- virtual double execute ()=0
- virtual void setName (string flowName)=0
- virtual void setSource (SystemInterface *sourceSys)=0
- virtual void setTarget (SystemInterface *targetSys)=0
- virtual string getName () const =0
- virtual SystemInterface * getSource () const =0
- virtual SystemInterface * getTarget () const =0
- virtual void clearSource ()=0
- virtual void clearTarget ()=0

4.9.1 Detailed Description

Class FlowInterface.

This Class represents a flow in the General Systems Theory implemented in this code.

Definition at line 10 of file flowinterface.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 ~FlowInteface()

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

This is the default destructor for the FlowInterface Class.

Definition at line 21 of file flowinterface.h.

4.9.3 Member Function Documentation

4.9.3.1 clearSource()

```
virtual void FlowInteface::clearSource ( )  [pure virtual]
```

Sets the pointer of the source attribute as NULL.

Implemented in Flow.

4.9.3.2 clearTarget()

```
virtual void FlowInteface::clearTarget ( )  [pure virtual]
```

Sets the pointer of the target attribute as NULL.

Implemented in Flow.

4.9.3.3 execute()

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

A pure virtual method that will be inherited by subclasses created by the user, and that will contain an equation that will be executed by the model.

Implemented in ExponencialFlow, LogisticFlow, ComplexFlowF, ComplexFlowT, ComplexFlowU, ComplexFlowV, ComplexFlowG, ComplexFlowR, ExponencialFlow, and Flow.

4.9.3.4 getName()

```
virtual string FlowInteface::getName ( ) const  [pure virtual]
```

Returns the name attribute in the FlowInterface Class.

Returns

string - the content name attribute.

Implemented in Flow.

4.9.3.5 getSource()

```
virtual SystemInterface * FlowInteface::getSource ( ) const  [pure virtual]
```

Sets the source attribute in the FlowInterface Class.

Parameters

sourceSys	a pointer to the source target.

Implemented in Flow.

4.9.3.6 getTarget()

```
virtual SystemInterface * FlowInteface::getTarget ( ) const  [pure virtual]
```

Returns the target attribute in the FlowInterface Class.

Returns

System* - the pointer of the target system.

Implemented in Flow.

4.9.3.7 setName()

```
virtual void FlowInteface::setName (
            string flowName )  [pure virtual]
```

Sets the name attribute in the FlowInterface Class.

Parameters

flowName	which will be set to the current flow.

Implemented in Flow.

4.9.3.8 setSource()

```
virtual void FlowInteface::setSource (
            SystemInterface * sourceSys )  [pure virtual]
```

Sets the source attribute in the FlowInterface Class.

Parameters

sourceSys	a pointer to the source target.

Implemented in Flow.

4.9.3.9 setTarget()

```
virtual void FlowInteface::setTarget (
            SystemInterface * targetSys )  [pure virtual]
```

Sets the target attribute in the FlowInterface Class.

Parameters

targetSys	a pointer to the target system.

Implemented in Flow.

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

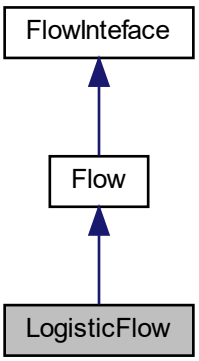
- src/flowinterface.h

4.10 LogisticFlow Class Reference

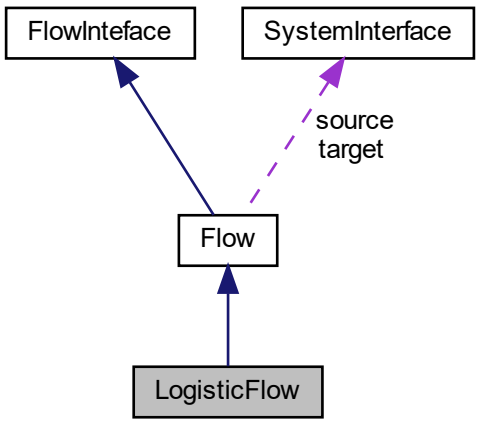
Class LogisticFlow.

```
#include <funcional_tests.h>
```

Inheritance diagram for LogisticFlow:



Collaboration diagram for LogisticFlow:



Public Member Functions

- LogisticFlow (string name, SystemInterface ∗source, SystemInterface ∗target)
- double execute ()

Additional Inherited Members

4.10.1 Detailed Description

Class LogisticFlow.

This Class represents a flow with limited growth, and is used in the Logistic Model test.

Definition at line 50 of file funcional_tests.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 LogisticFlow()

```
LogisticFlow::LogisticFlow (
            string name,
            SystemInterface * source,
            SystemInterface * target )  [inline]
```

This is a more elaborated constructor for the LogisticFlow Class.

Parameters

name	the name of the LogisticFlow Class.
source	a pointer to the source system of the LogisticFlow Class.
target	a pointer to the target system of the LogisticFlow Class.

Definition at line 58 of file funcional_tests.h.

4.10.3 Member Function Documentation

4.10.3.1 execute()

```
double LogisticFlow::execute ( )  [inline], [virtual]
```

A method created by the user, that contains an equation that will be executed by the logistic model.

Implements Flow.

Definition at line 64 of file funcional_tests.h.

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

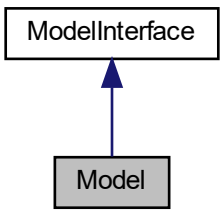
- test/funcional/funcional_tests.h

4.11 Model Class Reference

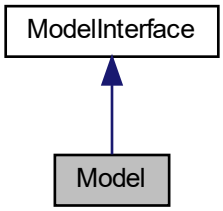
Class Model.

```
#include <model.h>
```

Inheritance diagram for Model:



Collaboration diagram for Model:



Public Types

- typedef vector< SystemInterface * >::iterator systemIterator
- typedef vector< FlowInteface * >::iterator flowIterator

Public Member Functions

- systemIterator beginSystems ()
- systemIterator endSystems ()
- flowIterator beginFlows ()
- flowIterator endFlows ()
- Model (string name="", double time=0)
- virtual ~Model ()
- void execute (double start=0, double final=0, double increment=1)
- ModelInterface * createModel (string name, double time)
- SystemInterface * createSystem (string name, double value)
- void add (SystemInterface *sys)
- void remove (SystemInterface *sys)
- void add (FlowInteface *flow)
- void remove (FlowInteface *flow)
- void setName (string modelName)
- void setTime (double currentTime)
- string getName () const
- double getTime () const
- SystemInterface * getSystem (int index)
- FlowInteface * getFlow (int index)
- void incrementTime (double increment)

Protected Attributes

- string name
- double time
- vector< SystemInterface * > systems
- vector< FlowInteface * > flows

Static Protected Attributes

- static vector< ModelInterface * > models

Additional Inherited Members

4.11.1 Detailed Description

Class Model.

This Class represents a Model in the General Systems Theory implemented in this code.

Definition at line 10 of file model.h.

4.11.2 Member Typedef Documentation

4.11.2.1 flowIterator

```
typedef vector<FlowInteface*>::iterator Model::flowIterator
```

Definition at line 33 of file model.h.

4.11.2.2 systemIterator

```
typedef vector<SystemInterface*>::iterator Model::systemIterator
```

Definition at line 32 of file model.h.

4.11.3 Constructor & Destructor Documentation

4.11.3.1 Model()

```
Model::Model (
            string name = "",
            double time = 0 )
```

This is the default constructor for the Model Class.

Parameters

name	the name of the Model Class.
time	the time for the Model Class to run.

Returns

Model - a Model Class object.

Definition at line 29 of file model.cpp.

4.11.3.2 ~Model()

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

This is the default destructor for the Model Class.

Definition at line 32 of file model.cpp.

4.11.4 Member Function Documentation

4.11.4.1 add() [1/2]

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

Adds a flow's pointer to the flows vector.

Parameters

flow	the flow to be added.

Implements ModelInterface.

Definition at line 94 of file model.cpp.

4.11.4.2 add() [2/2]

```
void Model::add (
            SystemInterface * sys )  [virtual]
```

Adds a system's pointer to the systems vector.

Parameters

sys	the system to be added.

Implements ModelInterface.

Definition at line 77 of file model.cpp.

4.11.4.3 beginFlows()

```
Model::flowIterator Model::beginFlows ( )  [virtual]
```

Returns the iterator to the beginning of flows attribute.

Implements ModelInterface.

Definition at line 21 of file model.cpp.

4.11.4.4 beginSystems()

```
Model::systemIterator Model::beginSystems ( )  [virtual]
```

Returns the iterator to the beginning of systems attribute.

Implements ModelInterface.

Definition at line 13 of file model.cpp.

4.11.4.5 createModel()

```
ModelInterface * Model::createModel (
            string name,
            double time )
```

Definition at line 65 of file model.cpp.

4.11.4.6 createSystem()

```
SystemInterface * Model::createSystem (
            string name,
            double value )  [virtual]
```

Implements ModelInterface.

Definition at line 71 of file model.cpp.

4.11.4.7 endFlows()

```
Model::flowIterator Model::endFlows ( )  [virtual]
```

Returns the iterator to the end of flows attribute.

Implements ModelInterface.

Definition at line 25 of file model.cpp.

4.11.4.8 endSystems()

```
Model::systemIterator Model::endSystems ( )  [virtual]
```

Returns the iterator to the end of systems attribute.

Implements ModelInterface.

Definition at line 17 of file model.cpp.

4.11.4.9 execute()

```
void Model::execute (
            double start = 0,
            double final = 0,
            double increment = 1 )  [virtual]
```

Executes all the flows in the Model.

Parameters

start	the initial time.
final	the final time.
increment	represents the iteration step.

Implements ModelInterface.

Definition at line 37 of file model.cpp.

4.11.4.10 getFlow()

```
FlowInteface * Model::getFlow (
            int index )  [virtual]
```

Returns a flow in the index-th position of the flows attribute Model Class.

Returns

Flow* - a flow in the index-th position of the systems attribute.

Implements ModelInterface.

Definition at line 130 of file model.cpp.

4.11.4.11 getName()

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

Returns the name attribute in the Model Class.

Returns

string - the name attribute.

Implements ModelInterface.

Definition at line 118 of file model.cpp.

4.11.4.12 getSystem()

```
SystemInterface * Model::getSystem (
            int index )  [virtual]
```

Returns a system in the index-th position of the systems attribute Model Class.

Returns

System* - a system in the index-th position of the systems attribute.

Implements ModelInterface.

Definition at line 126 of file model.cpp.

4.11.4.13 getTime()

```
double Model::getTime ( ) const  [virtual]
```

Returns the time attribute in the Model Class.

Returns

double - the time attribute.

Implements ModelInterface.

Definition at line 122 of file model.cpp.

4.11.4.14 incrementTime()

```
void Model::incrementTime (
            double increment )  [virtual]
```

This method increments the time attribute in the Model Class.

Parameters

increment	which will define by how much time should increment.

Implements ModelInterface.

Definition at line 134 of file model.cpp.

4.11.4.15 remove() [1/2]

```
void Model::remove (
            FlowInteface * flow )  [virtual]
```

Removes a flow's pointer on the flows vector.

Parameters

flow	which will be removed from the vector flows.

Implements ModelInterface.

Definition at line 98 of file model.cpp.

4.11.4.16 remove() [2/2]

```
void Model::remove (
            SystemInterface * sys )  [virtual]
```

Removes a system's pointer on the systems vector.

Parameters

sys	which will be removed from the vector flows.

Implements ModelInterface.

Definition at line 81 of file model.cpp.

4.11.4.17 setName()

```
void Model::setName (
            string modelName )  [virtual]
```

Sets the name attribute in the Model Class.

Parameters

modelName	which will be set to the current model.

Implements ModelInterface.

Definition at line 110 of file model.cpp.

4.11.4.18 setTime()

```
void Model::setTime (
             double currentTime )  [virtual]
```

Sets the time attribute in the Model Class.

Parameters

currentTime	which will be set to the current model.

Implements ModelInterface.

Definition at line 114 of file model.cpp.

4.11.5 Member Data Documentation

4.11.5.1 flows

```
vector<FlowInteface*> Model::flows  [protected]
```

This attribute stores pointers to the flows contained in the model.

Definition at line 16 of file model.h.

4.11.5.2 models

```
vector< ModelInterface * > Model::models  [static], [protected]
```

Definition at line 17 of file model.h.

4.11.5.3 name

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

This attribute contains a name for the model.

Definition at line 12 of file model.h.

4.11.5.4 systems

```
vector<SystemInterface*> Model::systems  [protected]
```

This attribute stores pointers to the systems contained in the model.

Definition at line 15 of file model.h.

4.11.5.5 time

```
double Model::time  [protected]
```

This attribute contains the current time in which the operations in the model is being executed.

Definition at line 13 of file model.h.

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

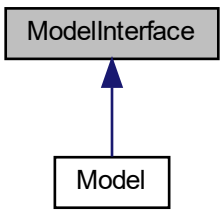
- src/model.h
- src/model.cpp

4.12 ModelInterface Class Reference

Class ModelInterface.

```
#include <modelinterface.h>
```

Inheritance diagram for ModelInterface:



Public Types

- typedef vector< SystemInterface * >::iterator systemIterator
- typedef vector< FlowInteface * >::iterator flowIterator

Public Member Functions

- virtual systemIterator beginSystems ()=0
- virtual systemIterator endSystems ()=0
- virtual flowIterator beginFlows ()=0
- virtual flowIterator endFlows ()=0
- virtual ~ModelInterface ()
- virtual void execute (double start, double final, double increment)=0
- virtual SystemInterface * createSystem (string name, double value)=0
- template<typename T_FLOW >
  FlowInteface * createFlow (SystemInterface *source=nullptr, SystemInterface *destination=nullptr)
- virtual void add (SystemInterface *sys)=0
- virtual void remove (SystemInterface *sys)=0
- virtual void add (FlowInteface *flow)=0
- virtual void remove (FlowInteface *flow)=0
- virtual string getName () const =0
- virtual double getTime () const =0
- virtual SystemInterface * getSystem (int index)=0
- virtual FlowInteface * getFlow (int index)=0
- virtual void setName (string modelName)=0
- virtual void setTime (double currentTime)=0
- virtual void incrementTime (double increment)=0

Static Public Member Functions

- static ModelInterface * createModel (string name, double time)

4.12.1 Detailed Description

Class ModelInterface.

This Class represents a model in the General Systems Theory implemented in this code.

Definition at line 11 of file modelinterface.h.

4.12.2 Member Typedef Documentation

4.12.2.1 flowIterator

```
typedef vector<FlowInteface*>::iterator ModelInterface::flowIterator
```

Definition at line 14 of file modelinterface.h.

4.12.2.2 systemIterator

```
typedef vector<SystemInterface*>::iterator ModelInterface::systemIterator
```

Definition at line 13 of file modelinterface.h.

4.12.3 Constructor & Destructor Documentation

4.12.3.1 ~ModelInterface()

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

This is the default destructor for the ModelInterface Class.

Definition at line 25 of file modelinterface.h.

4.12.4 Member Function Documentation

4.12.4.1 add() [1/2]

```
virtual void ModelInterface::add (
            FlowInteface * flow )  [pure virtual]
```

Adds a flow's pointer to the flows vector.

Parameters

flow	the flow to be added.

Implemented in Model.

4.12.4.2 add() [2/2]

```
virtual void ModelInterface::add (
            SystemInterface * sys )  [pure virtual]
```

Adds a system's pointer to the systems vector.

Parameters

sys	the system to be added.

Implemented in Model.

4.12.4.3 beginFlows()

```
virtual flowIterator ModelInterface::beginFlows ( )  [pure virtual]
```

Returns the iterator to the beginning of flows attribute.

Implemented in Model.

4.12.4.4 beginSystems()

```
virtual systemIterator ModelInterface::beginSystems ( )  [pure virtual]
```

Returns the iterator to the beginning of systems attribute.

Implemented in Model.

4.12.4.5 createFlow()

```
template<typename T_FLOW >
FlowInteface * ModelInterface::createFlow (
            SystemInterface * source = nullptr,
            SystemInterface * destination = nullptr )  [inline]
```

Definition at line 38 of file modelinterface.h.

4.12.4.6 createModel()

```
static ModelInterface * ModelInterface::createModel (
            string name,
            double time )  [static]
```

4.12.4.7 createSystem()

```
virtual SystemInterface * ModelInterface::createSystem (
            string name,
            double value )  [pure virtual]
```

Implemented in Model.

4.12.4.8 endFlows()

```
virtual flowIterator ModelInterface::endFlows ( )  [pure virtual]
```

Returns the iterator to the end of flows attribute.

Implemented in Model.

4.12.4.9 endSystems()

```
virtual systemIterator ModelInterface::endSystems ( )  [pure virtual]
```

Returns the iterator to the end of systems attribute.

Implemented in Model.

4.12.4.10 execute()

```
virtual void ModelInterface::execute (
            double start,
            double final,
            double increment )  [pure virtual]
```

Executes all the flows in the model.

Parameters

start	the initial time.
final	the final time.
increment	represents the iteration step.

Implemented in Model.

4.12.4.11 getFlow()

```
virtual FlowInteface * ModelInterface::getFlow (
            int index )  [pure virtual]
```

Returns a flow in the index-th position of the flows attribute ModelInterface Class.

Returns

Flow* - a flow in the index-th position of the systems attribute.

Implemented in Model.

4.12.4.12 getName()

```
virtual string ModelInterface::getName ( ) const  [pure virtual]
```

Returns the name attribute in the ModelInterface Class.

Returns

string - the name attribute.

Implemented in Model.

4.12.4.13 getSystem()

```
virtual SystemInterface * ModelInterface::getSystem (
            int index )  [pure virtual]
```

Returns a system in the index-th position of the systems attribute ModelInterface Class.

Returns

System* - a system in the index-th position of the systems attribute.

Implemented in Model.

4.12.4.14 getTime()

```
virtual double ModelInterface::getTime ( ) const  [pure virtual]
```

Returns the time attribute in the ModelInterface Class.

Returns

double - the time attribute.

Implemented in Model.

4.12.4.15 incrementTime()

```
virtual void ModelInterface::incrementTime (
            double increment )  [pure virtual]
```

This method increments the time attribute in the ModelInterface Class.

Parameters

increment	which will define by how much time should increment.

Implemented in Model.

4.12.4.16 remove() [1/2]

```
virtual void ModelInterface::remove (
            FlowInteface * flow )  [pure virtual]
```

Removes a flow's pointer on the flows vector.

Parameters

flow	which will be removed from the vector flows.

Implemented in Model.

4.12.4.17 remove() [2/2]

```
virtual void ModelInterface::remove (
            SystemInterface * sys )  [pure virtual]
```

Removes a system's pointer on the systems vector.

Parameters

sys	which will be removed from the vector flows.

Implemented in Model.

4.12.4.18 setName()

```
virtual void ModelInterface::setName (
            string modelName )  [pure virtual]
```

Sets the name attribute in the ModelInterface Class.

Parameters

modelName	which will be set to the current model.

Implemented in Model.

4.12.4.19 setTime()

```
virtual void ModelInterface::setTime (
            double currentTime )  [pure virtual]
```

Sets the time attribute in the ModelInterface Class.

Parameters

currentTime	which will be set to the current model.

Implemented in Model.

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

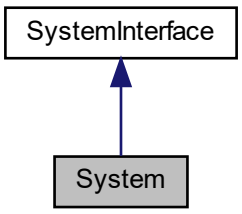
- src/modelinterface.h

4.13 System Class Reference

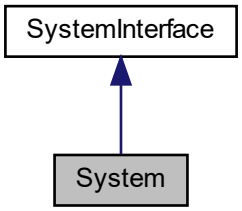
Class System.

```
#include <system.h>
```

Inheritance diagram for System:



Collaboration diagram for System:



Public Member Functions

- System (const System &sys)
- System (string name="", double value=0)
- virtual ~System ()
- void setName (string sysName)
- void setValue (double sysValue)
- string getName () const
- double getValue () const
- double operator+ (const SystemInterface &sys)
- double operator+ (const double &valueSys)
- double operator- (const SystemInterface &sys)
- double operator- (const double &valueSys)
- double operator* (const SystemInterface &sys)
- double operator* (const double &valueSys)
- double operator/ (const SystemInterface &sys)
- double operator/ (const double &valueSys)

Protected Member Functions

- System & operator= (const System &sys)

Protected Attributes

- string name
- double value

Friends

- class Flow
- class Model
- class UnitSystem

4.13.1 Detailed Description

Class System.

This Class represents a System in the General Systems Theory implemented in this code.

Definition at line 10 of file system.h.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 System() [1/2]

```
System::System (
            const System & sys )
```

This is the copy constructor for the System Class.

Parameters

sys	the System that is going to be cloned.

Definition at line 3 of file system.cpp.

4.13.2.2 System() [2/2]

```
System::System (
            string name = "",
            double value = 0 )
```

This is the default constructor for the System Class.

Parameters

name	the name of the System.
value	the initial value of the System.

Returns

System - a System Class object.

Definition at line 18 of file system.cpp.

4.13.2.3 ~System()

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

This is the default destructor for the System Class.

Definition at line 21 of file system.cpp.

4.13.3 Member Function Documentation

4.13.3.1 getName()

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

Returns the name attribute in the System Class.

Returns

string - the content name attribute.

Implements SystemInterface.

Definition at line 32 of file system.cpp.

4.13.3.2 getValue()

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

Returns the value attribute in the System Class.

Returns

double - the content value attribute.

Implements SystemInterface.

Definition at line 36 of file system.cpp.

4.13.3.3 operator∗() [1/2]

```
double System::operator* (
            const double & valueSys )  [virtual]
```

This is the overloaded "∗" operator for the System Class.

Implements SystemInterface.

Definition at line 73 of file system.cpp.

4.13.3.4 operator∗() [2/2]

```
double System::operator* (
            const SystemInterface & sys )  [virtual]
```

This is the overloaded "∗" operator for the System Class.

Implements SystemInterface.

Definition at line 65 of file system.cpp.

4.13.3.5 operator+() [1/2]

```
double System::operator+ (
            const double & valueSys )  [virtual]
```

This is the overloaded "-" operator for the System Class.

Implements SystemInterface.

Definition at line 49 of file system.cpp.

4.13.3.6 operator+() [2/2]

```
double System::operator+ (
            const SystemInterface & sys )  [virtual]
```

This is the overloaded "+" operator for the System Class.

Implements SystemInterface.

Definition at line 41 of file system.cpp.

4.13.3.7 operator-() [1/2]

```
double System::operator- (
            const double & valueSys )  [virtual]
```

This is the overloaded "-" operator for the System Class.

Implements SystemInterface.

Definition at line 61 of file system.cpp.

4.13.3.8 operator-() [2/2]

```
double System::operator- (
            const SystemInterface & sys )  [virtual]
```

This is the overloaded "-" operator for the System Class.

Implements SystemInterface.

Definition at line 53 of file system.cpp.

4.13.3.9 operator/() [1/2]

```
double System::operator/ (
            const double & valueSys )  [virtual]
```

This is the overloaded "/" operator for the System Class.

Implements SystemInterface.

Definition at line 86 of file system.cpp.

4.13.3.10 operator/() [2/2]

```
double System::operator/ (
            const SystemInterface & sys )  [virtual]
```

This is the overloaded "/" operator for the System Class.

Implements SystemInterface.

Definition at line 78 of file system.cpp.

4.13.3.11 operator=()

```
System & System::operator= (
            const System & sys )  [protected]
```

This is the overloaded assignment operator for the System Class.

Definition at line 10 of file system.cpp.

4.13.3.12 setName()

```
void System::setName (
            string sysName )  [virtual]
```

Sets the name attribute in the System Class.

Parameters

sysName	which will be set to the current System.

Implements SystemInterface.

Definition at line 24 of file system.cpp.

4.13.3.13 setValue()

```
void System::setValue (
            double sysValue )  [virtual]
```

Sets the value attribute in the System Class.

Parameters

sysValue	which will be set to the current System.

Implements SystemInterface.

Definition at line 28 of file system.cpp.

4.13.4 Friends And Related Function Documentation

4.13.4.1 Flow

```
friend class Flow  [friend]
```

This Class represents a flow in the General Systems Theory implemented in this code.

Definition at line 22 of file system.h.

4.13.4.2 Model

```
friend class Model  [friend]
```

This Class represents a model in the General Systems Theory implemented in this code.

Definition at line 23 of file system.h.

4.13.4.3 UnitSystem

```
friend class UnitSystem  [friend]
```

This Class is used to test the copy constructor and assignment operator of the System class.

Definition at line 24 of file system.h.

4.13.5 Member Data Documentation

4.13.5.1 name

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

This attribute contains a name for the System.

Definition at line 12 of file system.h.

4.13.5.2 value

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

This attribute contains the actual value of the System.

Definition at line 13 of file system.h.

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

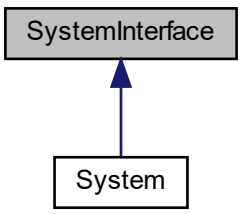
- src/system.h
- src/system.cpp

4.14 SystemInterface Class Reference

Class SystemInterface.

```
#include <systeminterface.h>
```

Inheritance diagram for SystemInterface:



Public Member Functions

- virtual ~SystemInterface ()
- virtual void setName (string sysName)=0
- virtual void setValue (double sysValue)=0
- virtual string getName () const =0
- virtual double getValue () const =0
- virtual double operator+ (const SystemInterface &sys)=0
- virtual double operator+ (const double &valueSys)=0
- virtual double operator- (const SystemInterface &sys)=0
- virtual double operator- (const double &valueSys)=0
- virtual double operator* (const SystemInterface &sys)=0
- virtual double operator* (const double &valueSys)=0
- virtual double operator/ (const SystemInterface &sys)=0
- virtual double operator/ (const double &valueSys)=0

4.14.1 Detailed Description

Class SystemInterface.

This Class represents a system in the General Systems Theory implemented in this code.

Definition at line 16 of file systeminterface.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 ~SystemInterface()

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

This is the default destructor for the SystemInterface Class.

Definition at line 23 of file systeminterface.h.

4.14.3 Member Function Documentation

4.14.3.1 getName()

```
virtual string SystemInterface::getName ( ) const  [pure virtual]
```

Returns the name attribute in the SystemInterface Class.

Returns

string - the content name attribute.

Implemented in System.

4.14.3.2 getValue()

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

Returns the value attribute in the SystemInterface Class.

Returns

double - the content value attribute.

Implemented in System.

4.14.3.3 operator∗() [1/2]

```
virtual double SystemInterface::operator* (
            const double & valueSys )  [pure virtual]
```

This is the overloaded "∗"" operator for the SystemInterface Class.

Implemented in System.

4.14.3.4 operator∗() [2/2]

```
virtual double SystemInterface::operator* (
            const SystemInterface & sys )  [pure virtual]
```

This is the overloaded "∗" operator for the SystemInterface Class.

Implemented in System.

4.14.3.5 operator+() [1/2]

```
virtual double SystemInterface::operator+ (
            const double & valueSys )  [pure virtual]
```

This is the overloaded "+" operator for the SystemInterface Class.

Implemented in System.

4.14.3.6 operator+() [2/2]

```
virtual double SystemInterface::operator+ (
            const SystemInterface & sys )  [pure virtual]
```

This is the overloaded "+" operator for the SystemInterface Class.

Implemented in System.

4.14.3.7 operator-() [1/2]

```
virtual double SystemInterface::operator- (
            const double & valueSys )  [pure virtual]
```

This is the overloaded "-" operator for the SystemInterface Class.

Implemented in System.

4.14.3.8 operator-() [2/2]

```
virtual double SystemInterface::operator- (
            const SystemInterface & sys )  [pure virtual]
```

This is the overloaded "-" operator for the SystemInterface Class.

Implemented in System.

4.14.3.9 operator/() [1/2]

```
virtual double SystemInterface::operator/ (
            const double & valueSys )  [pure virtual]
```

This is the overloaded "/" operator for the SystemInterface Class.

Implemented in System.

4.14.3.10 operator/() [2/2]

```
virtual double SystemInterface::operator/ (
            const SystemInterface & sys )  [pure virtual]
```

This is the overloaded "/" operator for the SystemInterface Class.

Implemented in System.

4.14.3.11 setName()

```
virtual void SystemInterface::setName (
            string sysName )  [pure virtual]
```

Sets the name attribute in the SystemInterface Class.

Parameters

sysName	which will be set to the current system.

Implemented in System.

4.14.3.12 setValue()

```
virtual void SystemInterface::setValue (
            double sysValue )  [pure virtual]
```

Sets the value attribute in the SystemInterface Class.

Parameters

sysValue	which will be set to the current system.

Implemented in System.

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

- src/systeminterface.h

4.15 UnitFlow Class Reference

Unit tests.

```
#include <unit_flow.h>
```

Public Member Functions

- UnitFlow ()
- ~UnitFlow ()
- void unit_flow_copy_constructor ()
- void unit_flow_assingmentOperator ()

4.15.1 Detailed Description

Unit tests.

Creation of the unit tests for the Flow class.

Definition at line 19 of file unit_flow.h.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 UnitFlow()

```
UnitFlow::UnitFlow ( )  [inline]
```

Definition at line 22 of file unit_flow.h.

4.15.2.2 ~UnitFlow()

```
UnitFlow::~UnitFlow ( )  [inline]
```

Definition at line 23 of file unit_flow.h.

4.15.3 Member Function Documentation

4.15.3.1 unit_flow_assingmentOperator()

```
void UnitFlow::unit_flow_assingmentOperator ( )
```

Function prototype for the Flow class' assingment operator unit test.

Definition at line 181 of file unit_flow.cpp.

4.15.3.2 unit_flow_copy_constructor()

```
void UnitFlow::unit_flow_copy_constructor ( )
```

Function prototype for the Flow class' copy constructor unit test.

Definition at line 21 of file unit_flow.cpp.

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

- test/unit/unit_flow.h
- test/unit/unit_flow.cpp

4.16 UnitSystem Class Reference

Unit tests.

`#include <unit_system.h>`

Public Member Functions

- UnitSystem ()
- ~UnitSystem ()
- void unit_system_copy_constructor ()
- void unit_system_assingmentOperator ()

4.16.1 Detailed Description

Unit tests.

Creation of the unit tests for the System class.

Definition at line 14 of file unit_system.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 UnitSystem()

`UnitSystem::UnitSystem ( )  [inline]`

Definition at line 16 of file unit_system.h.

4.16.2.2 ~UnitSystem()

`UnitSystem::~UnitSystem ( )  [inline]`

Definition at line 17 of file unit_system.h.

4.16.3 Member Function Documentation

4.16.3.1 unit_system_assingmentOperator()

```
void UnitSystem::unit_system_assingmentOperator ( )
```

Function prototype for the System class' assingment operator unit test.

Definition at line 123 of file unit_system.cpp.

4.16.3.2 unit_system_copy_constructor()

```
void UnitSystem::unit_system_copy_constructor ( )
```

Function prototype for the System class' copy constructor unit test.

Definition at line 30 of file unit_system.cpp.

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

- test/unit/unit_system.h
- test/unit/unit_system.cpp

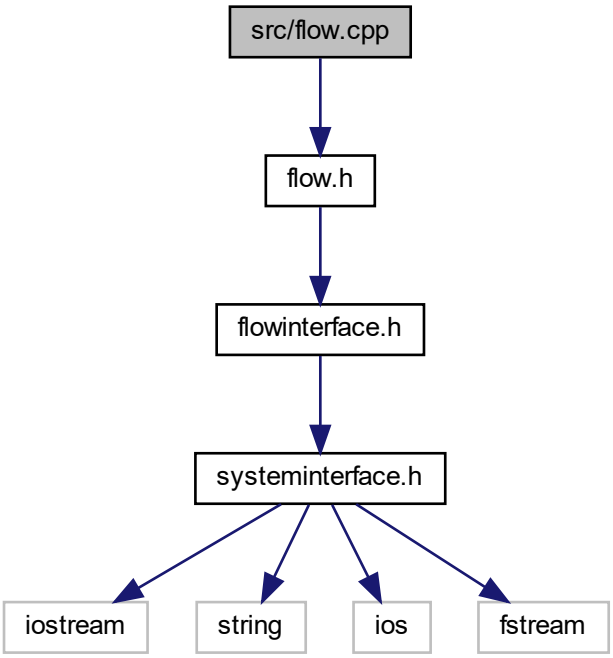
Chapter 5

File Documentation

5.1 src/flow.cpp File Reference

```
#include "flow.h"
```

Include dependency graph for flow.cpp:



5.2 flow.cpp

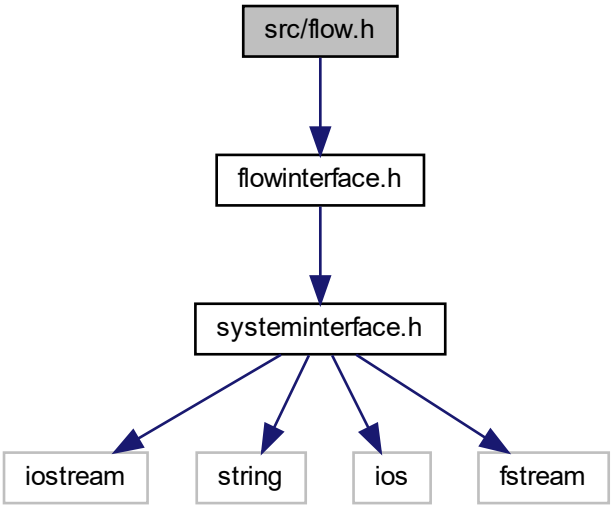
Go to the documentation of this file.

```
00001 #include "flow.h"
00002
00003 Flow :: Flow (const FlowInteface& flow){
00004     if (this != &flow) {
00005         this->name = flow.getName();
00006         this->source = NULL;
00007         this->target = NULL;
00008     }
00009 }
00010
00011 Flow& Flow :: operator=(const FlowInteface& flow){
00012     if (this != &flow){
00013         this->name = flow.getName();
00014         this->source = NULL;
00015         this->target = NULL;
00016     }
00017
00018     return *this;
00019 }
00020
00021 Flow :: Flow (string name, SystemInterface *source, SystemInterface *target):
00022     name (name), source (source), target (target) {}
00023
00024 Flow :: ~Flow () {}
00025
00026 string Flow :: getName() const {
00027     return this->name;
00028 }
00029
00030 SystemInterface* Flow :: getSource () const {
00031     return this->source;
00032 }
00033
00034 SystemInterface* Flow :: getTarget () const {
00035     return this->target;
00036 }
00037
00038 void Flow :: setName (string flowName) {
00039     this->name = flowName;
00040 }
00041
00042 void Flow :: setSource (SystemInterface* sourceSys) {
00043     this->source = sourceSys;
00044 }
00045
00046 void Flow :: setTarget (SystemInterface* targetSys) {
00047     this->target = targetSys;
00048 }
00049
00050 void Flow :: clearSource () {
00051     this->source = NULL;
00052 }
00053
00054 void Flow :: clearTarget () {
00055     this->target = NULL;
00056 }
```

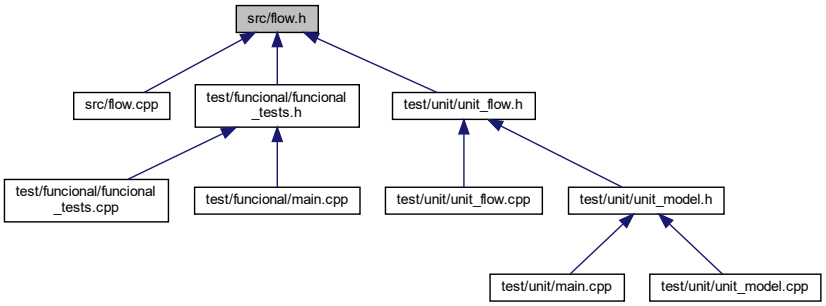
5.3 src/flow.h File Reference

```
#include "flowinterface.h"
```

Include dependency graph for flow.h:



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



Classes

- class Flow

  Class Flow.

5.4 flow.h

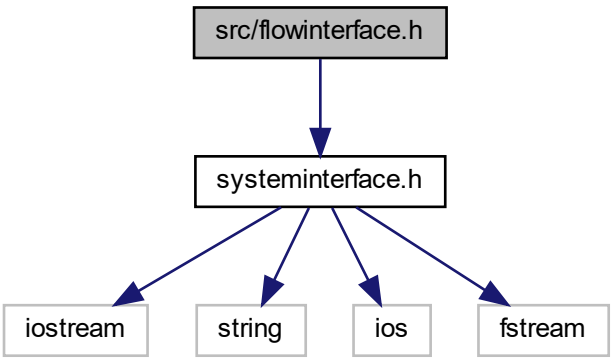
Go to the documentation of this file.

```
00001 #ifndef FLOW_H
00002 #define FLOW_H
00003
00004 #include "flowinterface.h"
00005
00007
00011 class Flow : public FlowInteface {
00012     protected:
00013         string name;
00014         SystemInterface *source;
00015         SystemInterface *target;
00021         Flow (const FlowInteface &flow);
00022
00026         Flow& operator= (const FlowInteface &flow);
00027
00028     public:
00029         friend class ModelInterface;
00030         friend class Model;
00031         friend class UnitFlow;
00040         Flow (string name = "", SystemInterface *source = NULL, SystemInterface *target = NULL);
00041
00045         virtual ~Flow ();
00046
00051         virtual double execute () = 0;
00052
00057         string getName () const;
00062         SystemInterface* getSource () const;
00067         SystemInterface* getTarget () const;
00068
00073         void setName (string flowName);
00078         void setSource (SystemInterface *sourceSys);
00083         void setTarget (SystemInterface *targetSys);
00084
00088         void clearSource ();
00092         void clearTarget ();
00093
00094 };
00095
00096
00097 #endif
```

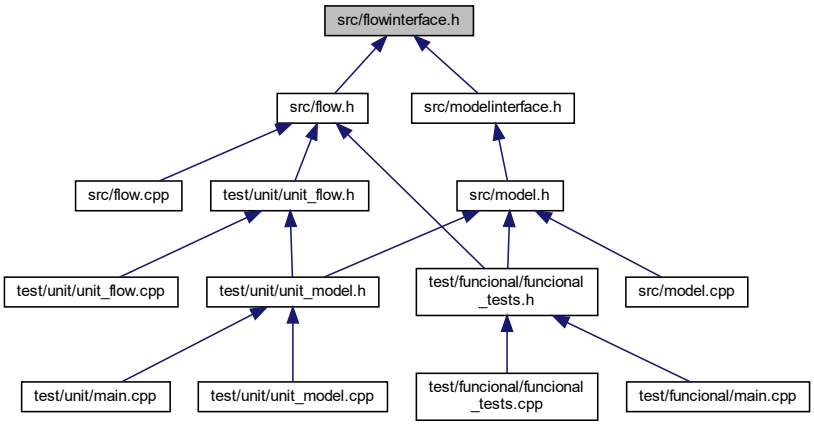
5.5 src/flowinterface.h File Reference

`#include "systeminterface.h"`

Include dependency graph for flowinterface.h:



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



Classes

- class FlowInteface

  Class FlowInterface.

5.6 flowinterface.h

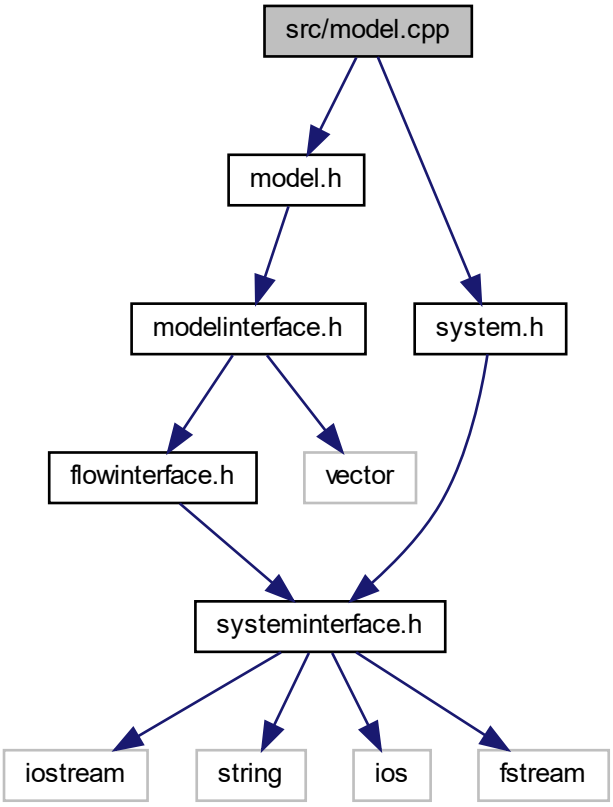
Go to the documentation of this file.

```
00001 #ifndef FLOW_INTERFACE_H
00002 #define FLOW_INTERFACE_H
00003
00004 #include "systeminterface.h"
00005
00007
00010 class FlowInteface{
00011     public:
00012         // Classes Amigas
00013         // friend class ModelInterface; /*!< This Class represents a model in the General Systems
       Theory implemented in this code. */
00014         // friend class Model; /*!< This Class represents a model in the General Systems Theory
       implemented in this code. */
00015         // friend class UnitFlow; /*!< This Class is used to test the copy constructor and assignment
       operator of the FlowInterface class. */
00016
00017         // Destrutor
00021         virtual ~FlowInteface() {}
00022
00023         // Funçaõ que executa a equação
00028         virtual double execute() = 0;
00029
00030         // Setters
00035         virtual void setName(string flowName) = 0;
00036
00041         virtual void setSource(SystemInterface* sourceSys) = 0;
00042
00047         virtual void setTarget(SystemInterface* targetSys) = 0;
00048
00049         // Getters
00054         virtual string getName() const = 0;
00059         virtual SystemInterface* getSource() const = 0;
00064         virtual SystemInterface* getTarget() const = 0;
00065
00066         // Funções que limpam os fluxos de entrada e saída
00070         virtual void clearSource() = 0;
00074         virtual void clearTarget() = 0;
00075 };
00076
00077
00078 #endif
```

5.7 src/model.cpp File Reference

```
#include "model.h"
#include "system.h"
```

Include dependency graph for model.cpp:



5.8 model.cpp

Go to the documentation of this file.

```
00001 #include "model.h"
00002 #include "system.h"
00003
00004 vector<ModelInterface*> Model :: models;
00005
00006 Model :: Model (const Model &model) {}
00007
00008 void Model :: operator= (const Model &model) {
00009
00010 }
00011
00012
00013 Model :: systemIterator Model :: beginSystems () {
00014     return systems.begin();
00015 }
00016
00017 Model :: systemIterator Model :: endSystems () {
00018     return systems.end();
```

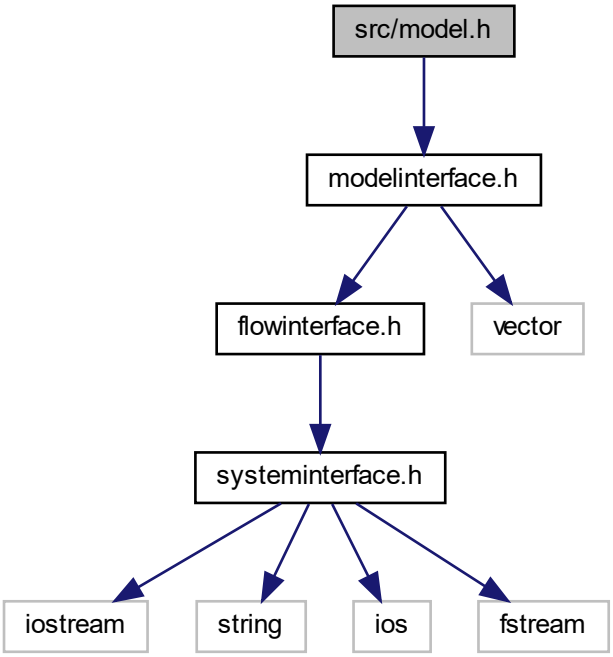
```
00019 }
00020
00021 Model :: flowIterator Model :: beginFlows () {
00022     return flows.begin();
00023 }
00024
00025 Model :: flowIterator Model :: endFlows () {
00026     return flows.end();
00027 }
00028
00029 Model :: Model (string name, double time) :
00030     name (name), time(time) {}
00031
00032 Model :: ~Model() {
00033     flows.clear();
00034     systems.clear();
00035 }
00036
00037 void Model :: execute (double start, double final, double increment) {
00038     vector <double> results;
00039     int j = 0;
00040
00041     for (double k = start; k < final; k += increment) {
00042         for (FlowInteface *item : flows) {
00043             results.push_back(item->execute());
00044         }
00045
00046         j = 0;
00047
00048         for (FlowInteface *item : flows) {
00049             if (item->getSource() != NULL)
00050                 item->getSource()->setValue(item->getSource()->getValue() - results[j]);
00051
00052             if (item->getTarget() != NULL)
00053                 item->getTarget()->setValue(item->getTarget()->getValue() + results[j]);
00054
00055             j++;
00056         }
00057
00058         for (auto item = beginFlows(); item != endFlows(); ++item)
00059             results.pop_back();
00060
00061         time += increment;
00062     }
00063 }
00064
00065 ModelInterface* Model :: createModel(string name, double time){
00066     ModelInterface* m = new Model(name, time);
00067     models.push_back(m);
00068     return m;
00069 }
00070
00071 SystemInterface* Model :: createSystem(string name, double value){
00072     SystemInterface* s = new System(name,value);
00073     this->add(s);
00074     return s;
00075 }
00076
00077 void Model :: add (SystemInterface *sys) {
00078     systems.insert(endSystems(), sys);
00079 }
00080
00081 void Model :: remove (SystemInterface *sys) {
00082     auto i = beginSystems();
00083
00084     for (SystemInterface *item : systems) {
00085         if (item == sys) {
00086             systems.erase(i);
00087             break;
00088         }
00089         ++i;
00090     }
00091 }
00092
00093
00094 void Model :: add (FlowInteface *flow) {
00095     flows.insert(endFlows(), flow);
00096 }
00097
00098 void Model :: remove (FlowInteface *flow) {
00099     auto i = beginFlows();
00100
00101     for (FlowInteface *item : flows) {
00102         if (item == flow) {
00103             flows.erase(i);
00104             break;
00105         }
```

```
00106         ++i;
00107     }
00108 }
00109
00110 void Model :: setName (string modelName) {
00111     this->name = modelName;
00112 }
00113
00114 void Model :: setTime (double currentTime) {
00115     this->time = currentTime;
00116 }
00117
00118 string Model :: getName () const {
00119     return this->name;
00120 }
00121
00122 double Model :: getTime () const {
00123     return this->time;
00124 }
00125
00126 SystemInterface* Model :: getSystem (int index) {
00127     return this->systems[index];
00128 }
00129
00130 FlowInteface* Model :: getFlow (int index) {
00131     return this->flows[index];
00132 }
00133
00134 void Model :: incrementTime (double increment) {
00135     this->time += increment;
00136 }
```

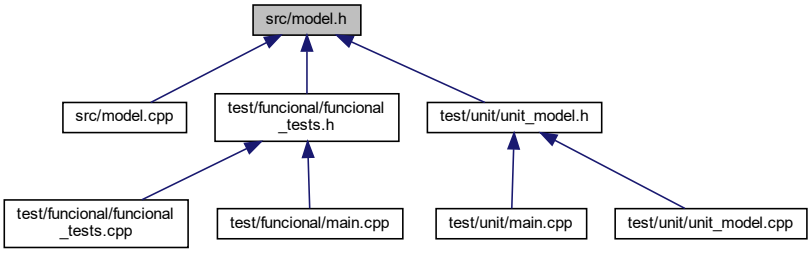
5.9 src/model.h File Reference

`#include "modelinterface.h"`

Include dependency graph for model.h:



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



Classes

- class Model

  Class Model.

5.10 model.h

Go to the documentation of this file.

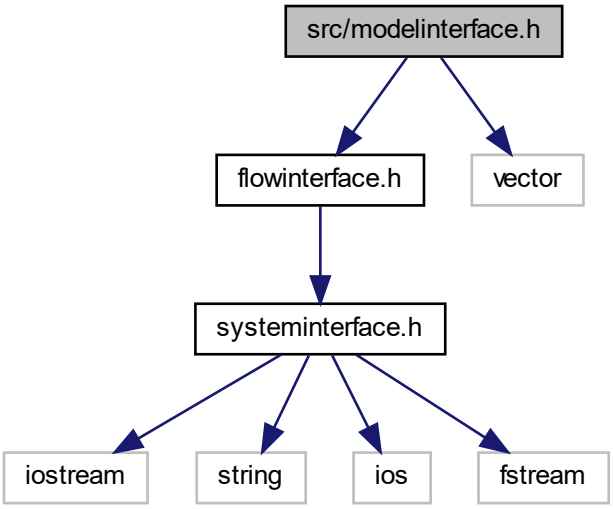
```
00001 #ifndef MODEL_H
00002 #define MODEL_H
00003
00004 #include "modelinterface.h"
00005
00007
00010 class Model : public ModelInterface{
00011     protected:
00012         string name;
00013         double time;
00015         vector <SystemInterface*> systems;
00016         vector <FlowInteface*> flows;
00017         static vector<ModelInterface*> models;
00018
00019     private:
00024         Model (const Model &model);
00025
00029         void operator= (const Model &model);
00030
00031     public:
00032         typedef vector<SystemInterface*> :: iterator systemIterator;
00033         typedef vector<FlowInteface*> :: iterator flowIterator;
00034
00035         systemIterator beginSystems();
00036         systemIterator endSystems();
00037         flowIterator beginFlows();
00038         flowIterator endFlows();
00046         Model (string name = "", double time = 0);
00047
00051         virtual ~Model();
00052
00059         void execute (double start = 0, double final = 0, double increment = 1);
00060
00061
00062         ModelInterface* createModel(string name, double time);
00063
00064         SystemInterface* createSystem(string name, double value);
00065
00066
00071         void add (SystemInterface *sys);
00076         void remove (SystemInterface *sys);
00077
00082         void add (FlowInteface *flow);
00087         void remove (FlowInteface *flow);
00088
00093         void setName(string modelName);
```

```
00098         void setTime(double currentTime);
00099
00104         string getName() const;
00109         double getTime() const;
00114         SystemInterface* getSystem(int index);
00119         FlowInteface* getFlow(int index);
00120
00125         void incrementTime(double increment);
00126
00127
00128 };
00129
00130 #endif
```

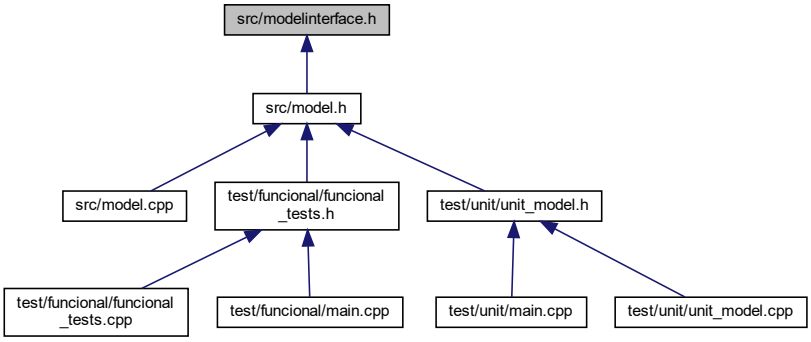
5.11 src/modelinterface.h File Reference

```
#include "flowinterface.h"
#include <vector>
```

Include dependency graph for modelinterface.h:



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



Classes

- class ModelInterface

  Class ModelInterface.

5.12 modelinterface.h

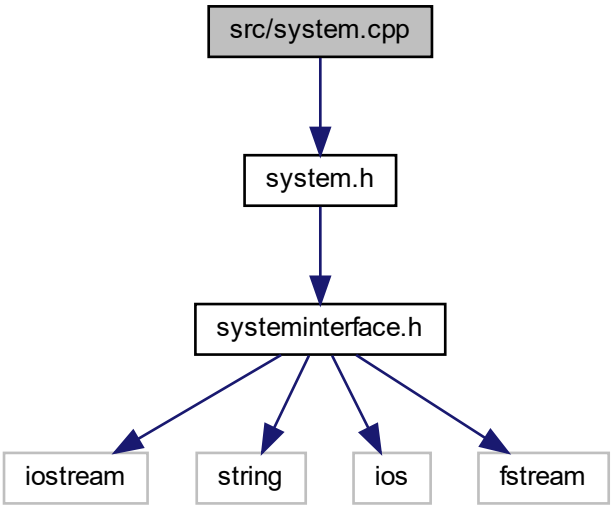
Go to the documentation of this file.

```
00001 #ifndef MODEL_INTERFACE_H
00002 #define MODEL_INTERFACE_H
00003
00004 #include "flowinterface.h"
00005 #include <vector>
00006
00008
00011 class ModelInterface{
00012     public:
00013         typedef vector <SystemInterface*> :: iterator systemIterator;
00014         typedef vector <FlowInteface*> :: iterator flowIterator;
00015
00016         virtual systemIterator beginSystems() = 0;
00017         virtual systemIterator endSystems() = 0;
00019         virtual flowIterator beginFlows() = 0;
00020         virtual flowIterator endFlows() = 0;
00025         virtual ~ModelInterface() {}
00026
00033         virtual void execute (double start, double final, double increment) = 0;
00034
00035         static ModelInterface* createModel(string name, double time);
00036         virtual SystemInterface* createSystem(string name, double value) = 0;
00037         template <typename T_FLOW>
00038         FlowInteface* createFlow(SystemInterface* source = nullptr, SystemInterface* destination =
      nullptr){
00039             FlowInteface* flow = new T_FLOW();
00040             flow->setSource(source);
00041             flow->setTarget(destination);
00042             add(flow);
00043             return flow;
00044         }
00045
00046
00051         virtual void add (SystemInterface *sys) = 0;
00056         virtual void remove (SystemInterface *sys) = 0;
00057
00062         virtual void add (FlowInteface *flow) = 0;
00067         virtual void remove (FlowInteface *flow) = 0;
00068
00069         // Getters
00074         virtual string getName() const = 0;
00079         virtual double getTime() const = 0;
```

```
00084         virtual SystemInterface* getSystem (int index) = 0;
00089         virtual FlowInteface* getFlow (int index) = 0;
00090
00091         // Setters
00096         virtual void setName (string modelName) = 0;
00101         virtual void setTime(double currentTime) = 0;
00102
00107         virtual void incrementTime(double increment) = 0;
00108 };
00109 #endif
```

5.13 src/system.cpp File Reference

`#include "system.h"`
Include dependency graph for system.cpp:



Functions

- double operator+ (const double &valueSys, const SystemInterface &sys)
- double operator- (const double &valueSys, const SystemInterface &sys)
- double operator∗ (const double &valueSys, const SystemInterface &sys)
- double operator/ (const double &valueSys, const SystemInterface &sys)

5.13.1 Function Documentation

5.13.1.1 operator*()

```
double operator* (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 102 of file system.cpp.

5.13.1.2 operator+()

```
double operator+ (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 94 of file system.cpp.

5.13.1.3 operator-()

```
double operator- (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 98 of file system.cpp.

5.13.1.4 operator/()

```
double operator/ (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 106 of file system.cpp.

5.14 system.cpp

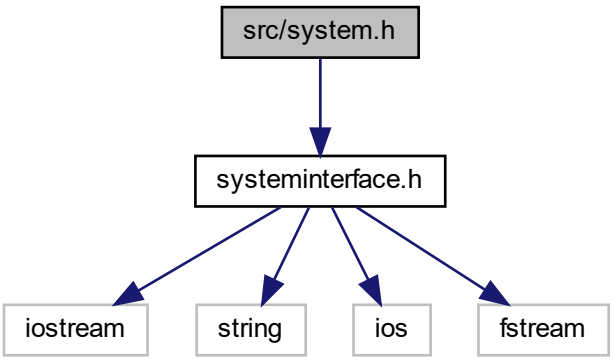
Go to the documentation of this file.

```
00001 #include "system.h"
00002
00003 System :: System (const System& sys){
00004     if (this != &sys){
00005         name = sys.getName();
00006         value = sys.getValue();
00007     }
00008 }
00009
00010 System& System :: operator= (const System& sys){
00011     if (this != &sys){
00012         setName(sys.getName());
00013         setValue(sys.getValue());
00014     }
00015     return *this;
00016 }
00017
00018 System :: System(string name, double value):
00019     name(name), value(value){}
00020
00021 System :: ~System() {}
00022
00023
00024 void System :: setName(string sysName){
00025     name = sysName;
00026 }
00027
00028 void System :: setValue(double sysValue){
00029     value = sysValue;
00030 }
00031
00032 string System :: getName() const {
00033     return name;
00034 }
00035
00036 double System :: getValue() const{
00037     return value;
00038 }
00039
00040
00041 double System :: operator+(const SystemInterface& sys){
00042     if (this == &sys){
00043         return 2.0 * this->value;
00044     }
00045
00046     return this->value + sys.getValue();
00047 }
00048
00049 double System :: operator+(const double& valueSys){
00050     return valueSys + this->value;
00051 }
00052
00053 double  System :: operator-(const SystemInterface& sys){
00054     if (this == &sys){
00055         return 0.0;
00056     }
00057
00058     return this->value - sys.getValue();
00059 }
00060
00061 double System :: operator-(const double& valueSys){
00062     return this->value - valueSys;
00063 }
00064
00065 double System :: operator*(const SystemInterface& sys){
00066     if (this == &sys){
00067         return this->value * sys.getValue();
00068     }
00069
00070     return this->value * sys.getValue();
00071 }
00072
00073 double System :: operator*(const double& valueSys){
00074     return this->value * valueSys;
00075 }
00076
00077
00078 double System :: operator/(const SystemInterface& sys){
00079     if (this == &sys){
00080         return 1.0;
00081     }
00082
```

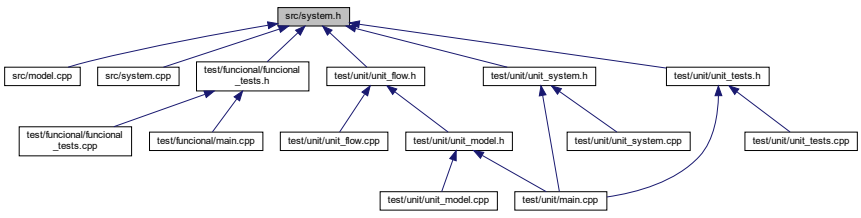
```
00083     return this->value / sys.getValue();
00084 }
00085
00086 double System :: operator/(const double& valueSys){
00087     return this->value / valueSys;
00088 }
00089
00090
00091 //-------------------------------------------------
00092
00093
00094 double operator+(const double& valueSys, const SystemInterface& sys){
00095     return sys.getValue() + valueSys;
00096 }
00097
00098 double operator-(const double& valueSys, const SystemInterface& sys){
00099     return valueSys - sys.getValue();
00100 }
00101
00102 double operator*(const double& valueSys, const SystemInterface& sys){
00103     return valueSys * sys.getValue();
00104 }
00105
00106 double operator/(const double& valueSys, const SystemInterface& sys){
00107     return valueSys / sys.getValue();
00108 }
```

5.15 src/system.h File Reference

`#include "systeminterface.h"`
Include dependency graph for system.h:



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



Classes

- class System

  Class System.

Functions

- double operator+ (const double &valueSys, const SystemInterface &sys)
- double operator- (const double &valueSys, const SystemInterface &sys)
- double operator∗ (const double &valueSys, const SystemInterface &sys)
- double operator/ (const double &valueSys, const SystemInterface &sys)

5.15.1 Function Documentation

5.15.1.1 operator∗()

```
double operator* (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 102 of file system.cpp.

5.15.1.2 operator+()

```
double operator+ (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 94 of file system.cpp.

5.15.1.3 operator-()

```
double operator- (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 98 of file system.cpp.

5.15.1.4 operator/()

```
double operator/ (
            const double & valueSys,
            const SystemInterface & sys )
```

Definition at line 106 of file system.cpp.

5.16 system.h

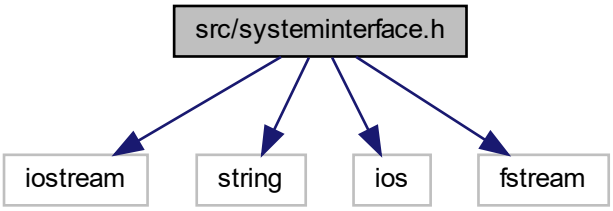
Go to the documentation of this file.

```
00001 #ifndef SYSTEM_H
00002 #define SYSTEM_H
00003
00004 #include "systeminterface.h"
00005
00007
00010 class System : public SystemInterface{
00011     protected:
00012         string name;
00013         double value;
00018         System& operator=(const System& sys);
00019
00020     public:
00021
00022         friend class Flow;
00023         friend class Model;
00024         friend class UnitSystem;
00026         // Construtor e Destrutor
00031         System (const System& sys);
00032
00039         System (string name = "", double value = 0);
00040
00044         virtual ~System();
00045
00046         // Setters
00051         void setName(string sysName);
00056         void setValue(double sysValue);
00057
00058         // Getters
00063         string getName() const;
00068         double getValue() const;
00069
00070         // Sobrecarga de Operadores
00071         // Operador +
00075         double operator+(const SystemInterface& sys);
00079         double operator+(const double& valueSys);
00080
00081         //Operador -
00085         double operator-(const SystemInterface& sys);
00089         double operator-(const double& valueSys);
00090
00091         // Operador *
00095         double operator*(const SystemInterface& sys);
00099         double operator*(const double& valueSys);
00100
00101         // Operador /
00105         double operator/(const SystemInterface& sys);
00109         double operator/(const double& valueSys);
00110
00111
00112
00113 };
00114
00115 double operator+(const double& valueSys, const SystemInterface& sys);
00116 double operator-(const double& valueSys, const SystemInterface& sys);
00117 double operator*(const double& valueSys, const SystemInterface& sys);
00118 double operator/(const double& valueSys, const SystemInterface& sys);
00119
00120 #endif
```

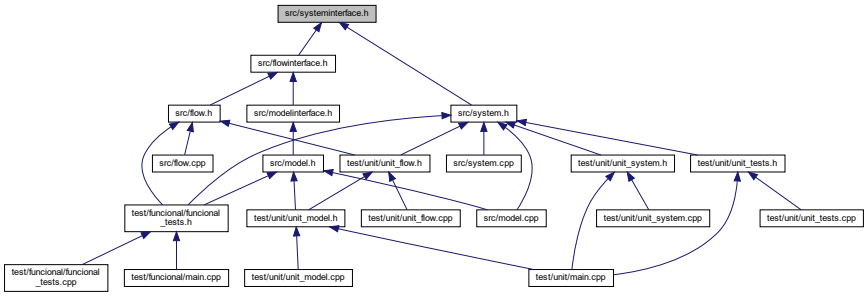
5.17 src/systeminterface.h File Reference

```
#include <iostream>
#include <string>
#include <ios>
#include <fstream>
```

Include dependency graph for systeminterface.h:



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



Classes

- class SystemInterface

  Class SystemInterface.

5.18 systeminterface.h

Go to the documentation of this file.

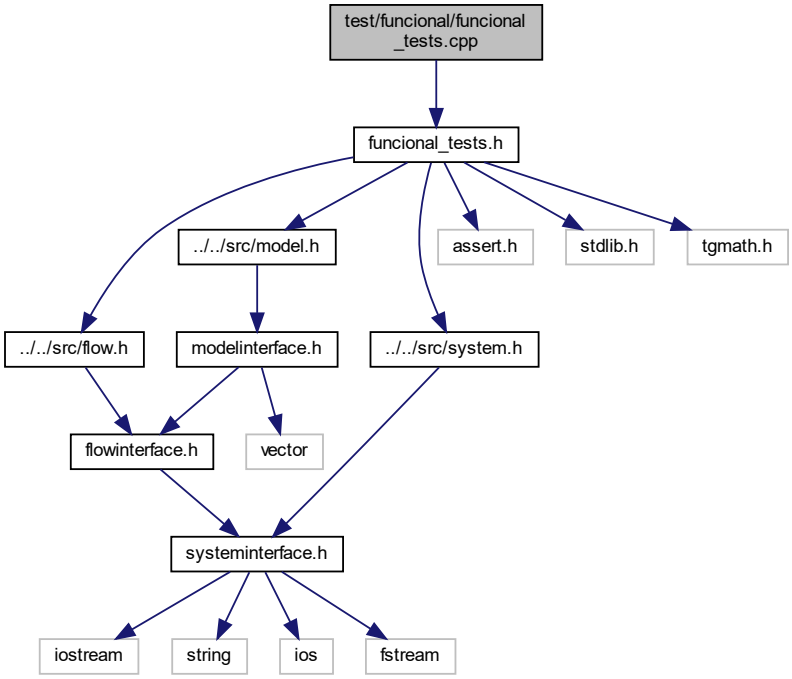
```
00001 #ifndef SYSTEM_INTERFACE_H
00002 #define SYSTEM_INTERFACE_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <ios>
00007 #include <fstream>
00008
00009 using namespace std;
```

```
00010
00011
00013
00016 class SystemInterface {
00017
00018     public:
00019         // Destrutor
00023         virtual ~SystemInterface(){}
00024
00025         //Setters
00030         virtual void setName(string sysName) = 0;
00035         virtual void setValue(double sysValue) = 0;
00036
00037         // Getters
00042         virtual string getName() const = 0;
00047         virtual double getValue() const = 0;
00048
00049         // Sobrecargas
00050         //Operador +
00054         virtual double operator+(const SystemInterface& sys) = 0;
00058         virtual double operator+(const double& valueSys) = 0;
00059
00060         // Operador -
00064         virtual double operator-(const SystemInterface& sys) = 0;
00068         virtual double operator-(const double& valueSys) = 0;
00069
00070         // Operador *
00074         virtual double operator*(const SystemInterface& sys) = 0;
00078         virtual double operator*(const double& valueSys) = 0;
00079
00080         // Operador /
00084         virtual double operator/(const SystemInterface& sys) = 0;
00088         virtual double operator/(const double& valueSys) = 0;
00089
00090
00091 };
00092
00093
00094 #endif
```

5.19 test/funcional/funcional_tests.cpp File Reference

```
#include "funcional_tests.h"
```

Include dependency graph for funcional_tests.cpp:



Functions

- void exponentialFuncionalTest ()
- void logisticalFuncionalTest ()
- void complexFuncionalTest ()

5.19.1 Function Documentation

5.19.1.1 complexFuncionalTest()

```
void complexFuncionalTest ( )
```

Function prototype for the complex flows functional tests.

Definition at line 79 of file funcional_tests.cpp.

5.19.1.2 exponentialFuncionalTest()

```
void exponentialFuncionalTest ( )
```

Function prototype for exponential flow functional test.

Definition at line 3 of file funcional_tests.cpp.

5.19.1.3 logisticalFuncionalTest()

```
void logisticalFuncionalTest ( )
```

Function prototype for logistical flow functional test.

Definition at line 43 of file funcional_tests.cpp.

5.20 funcional_tests.cpp

Go to the documentation of this file.

```
00001 #include "funcional_tests.h"
00002
00003 void exponentialFuncionalTest() {
00004     SystemInterface *population1 = new System ("Population 1", 100);
00005     SystemInterface *population2 = new System ("Population 2", 0);
00006
00007     ExponencialFlow* exponentialFlow = new ExponencialFlow("Unlimited Growth", population1,
      population2);
00008
00009     ModelInterface *exponentialModel = new Model ("Exponential Model", 0.0);
00010
00011     exponentialModel->add(population1);
00012     exponentialModel->add(population2);
00013     exponentialModel->add(exponentialFlow);
00014
00015
00016     assert(population1->getName() == "Population 1");
00017     assert(population2->getName() == "Population 2");
00018     assert(exponentialFlow->getName() == "Unlimited Growth");
00019     assert(exponentialModel->getName() == "Exponential Model");
00020
00021     cout « "NAMES PASSED" « endl;
00022
00023
00024     assert(abs(population1->getValue() - 100.0) < 0.0001);
00025     assert(abs(population2->getValue() - 0.0) < 0.0001);
00026     assert(abs(exponentialModel->getTime() - 0.0) < 0.0001);
00027
00028     cout « "VALUES PASSED" « endl;
00029
00030     exponentialModel->execute(0, 100, 1);
00031
00032
00033     assert(abs(population1->getValue() - 36.6032) < 0.0001);
00034     assert(abs(population2->getValue() - 63.3968) < 0.0001);
00035     assert(abs(exponentialModel->getTime() - 100) < 0.0001);
00036
00037     cout « "EXECUTE PASSED" « endl;
00038
00039     delete exponentialModel;
00040 }
00041
00042
00043 void logisticalFuncionalTest() {
00044     SystemInterface *population1 = new System ("Population 1", 100);
00045     SystemInterface *population2 = new System ("Population 2", 10);
00046
00047     LogisticFlow* logisticalFlow = new LogisticFlow("Limited Growth", population1, population2);
00048     ModelInterface *logisticalModel = new Model ("Logistic Model", 0);
```

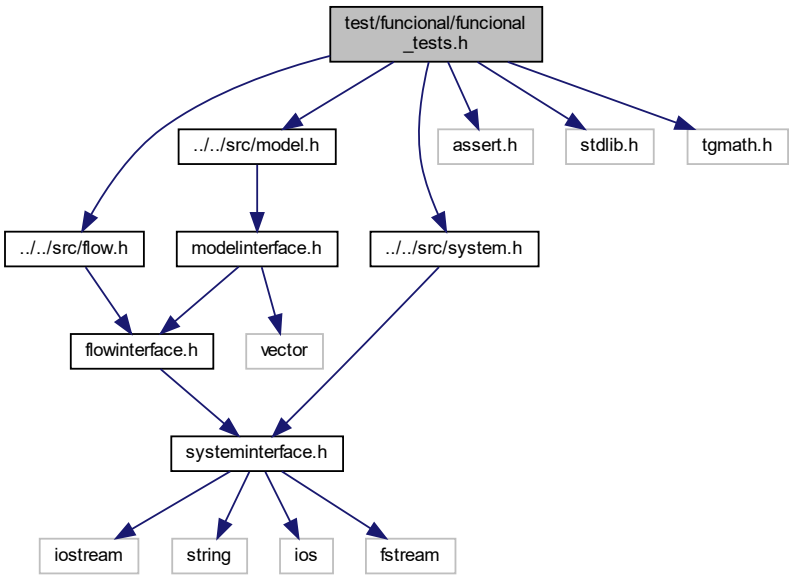
```
00049
00050     logisticalModel->add(population1);
00051     logisticalModel->add(population2);
00052     logisticalModel->add(logisticalFlow);
00053
00054
00055     assert(population1->getName() == "Population 1");
00056     assert(population2->getName() == "Population 2");
00057     assert(logisticalFlow->getName() == "Limited Growth");
00058     assert(logisticalModel->getName() == "Logistic Model");
00059     cout « "NAMES PASSED" « endl;
00060
00061
00062     assert(abs(population1->getValue() - 100.0) < 0.0001);
00063     assert(abs(population2->getValue() - 10.0) < 0.0001);
00064     assert(abs(logisticalModel->getTime() - 0.0) < 0.0001);
00065     cout « "VALUES PASSED" « endl;
00066
00067
00068     logisticalModel->execute(0, 100, 1);
00069
00070     assert(abs(population1->getValue() - 88.2167) < 0.0001);
00071     assert(abs(population2->getValue() - 21.7833) < 0.0001);
00072     assert(abs(logisticalModel->getTime() - 100.0) < 0.0001);
00073     cout « "EXECUTE PASSED" « endl;
00074
00075     delete logisticalModel;
00076 }
00077
00078
00079 void complexFuncionalTest() {
00080     SystemInterface *Q1 = new System ("Q1", 100);
00081     SystemInterface *Q2 = new System ("Q2", 0);
00082     SystemInterface *Q3 = new System ("Q3", 100);
00083     SystemInterface *Q4 = new System ("Q4", 0);
00084     SystemInterface *Q5 = new System ("Q5", 0);
00085
00086     ComplexFlowF* complexFlowF = new ComplexFlowF("Flow f", Q1, Q2);
00087     ComplexFlowT* complexFlowT = new ComplexFlowT("Flow t", Q2, Q3);
00088     ComplexFlowU* complexFlowU = new ComplexFlowU("Flow u", Q3, Q4);
00089     ComplexFlowV* complexFlowV = new ComplexFlowV("Flow v", Q4, Q1);
00090     ComplexFlowG* complexFlowG = new ComplexFlowG("Flow g", Q1, Q3);
00091     ComplexFlowR* complexFlowR = new ComplexFlowR("Flow r", Q2, Q5);
00092
00093     ModelInterface *complexModel = new Model ("Complex Model", 0);
00094
00095     complexModel->add(Q1);
00096     complexModel->add(Q2);
00097     complexModel->add(Q3);
00098     complexModel->add(Q4);
00099     complexModel->add(Q5);
00100     complexModel->add(complexFlowF);
00101     complexModel->add(complexFlowT);
00102     complexModel->add(complexFlowU);
00103     complexModel->add(complexFlowV);
00104     complexModel->add(complexFlowG);
00105     complexModel->add(complexFlowR);
00106
00107     assert(Q1->getName() == "Q1");
00108     assert(Q2->getName() == "Q2");
00109     assert(Q3->getName() == "Q3");
00110     assert(Q4->getName() == "Q4");
00111     assert(Q5->getName() == "Q5");
00112     assert(complexFlowF->getName() == "Flow f");
00113     assert(complexFlowT->getName() == "Flow t");
00114     assert(complexFlowU->getName() == "Flow u");
00115     assert(complexFlowV->getName() == "Flow v");
00116     assert(complexFlowG->getName() == "Flow g");
00117     assert(complexFlowR->getName() == "Flow r");
00118     assert(complexModel->getName() == "Complex Model");
00119     cout « "NAMES PASSED" « endl;
00120
00121     assert(abs(Q1->getValue() - 100.0) < 0.0001);
00122     assert(abs(Q2->getValue() - 0.0) < 0.0001);
00123     assert(abs(Q3->getValue() - 100.0) < 0.0001);
00124     assert(abs(Q4->getValue() - 0.0) < 0.0001);
00125     assert(abs(Q5->getValue() - 0.0) < 0.0001);
00126     assert(abs(complexModel->getTime() - 0.0) < 0.0001);
00127     cout « "VALUES PASSED" « endl;
00128
00129     complexModel->execute(0, 100, 1);
00130
00131     assert(abs(Q1->getValue() - 31.8513) < 0.0001);
00132     assert(abs(Q2->getValue() - 18.4003) < 0.0001);
00133     assert(abs(Q3->getValue() - 77.1143) < 0.0001);
00134     assert(abs(Q4->getValue() - 56.1728) < 0.0001);
00135     assert(abs(Q5->getValue() - 16.4612) < 0.0001);
```

```
00136     assert(abs(complexModel->getTime() - 100.0) < 0.0001);
00137     cout « "EXECUTE PASSED" « endl;
00138
00139     delete complexModel;
00140 }
```

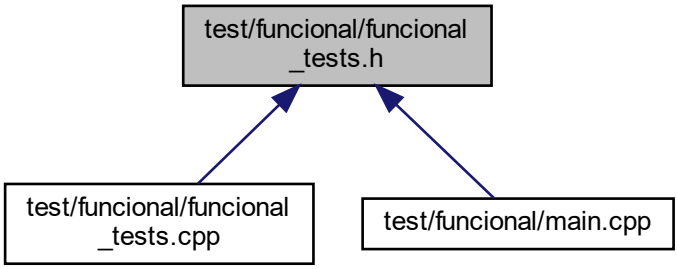
5.21 test/funcional/funcional_tests.h File Reference

```
#include "../../src/flow.h"
#include "../../src/system.h"
#include "../../src/model.h"
#include <assert.h>
#include <stdlib.h>
#include <tgmath.h>
```

Include dependency graph for funcional_tests.h:



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



Classes

- class ExponencialFlow

  Functional tests.
- class LogisticFlow

  Class LogisticFlow.
- class ComplexFlowF

  Class ComplexFlowF.
- class ComplexFlowT

  Class ComplexFlowT.
- class ComplexFlowU

  Class ComplexFlowU.
- class ComplexFlowV

  Class ComplexFlowV.
- class ComplexFlowG

  Class ComplexFlowG.
- class ComplexFlowR

  Class ComplexFlowR.

Functions

- void exponentialFuncionalTest ()
- void logisticalFuncionalTest ()
- void complexFuncionalTest ()

5.21.1 Function Documentation

5.21.1.1 complexFuncionalTest()

```
void complexFuncionalTest ( )
```

Function prototype for the complex flows functional tests.

Definition at line 79 of file funcional_tests.cpp.

5.21.1.2 exponentialFuncionalTest()

```
void exponentialFuncionalTest ( )
```

Function prototype for exponential flow functional test.

Definition at line 3 of file funcional_tests.cpp.

5.21.1.3 logisticalFuncionalTest()

```
void logisticalFuncionalTest ( )
```

Function prototype for logistical flow functional test.

Definition at line 43 of file funcional_tests.cpp.

5.22 funcional_tests.h

Go to the documentation of this file.

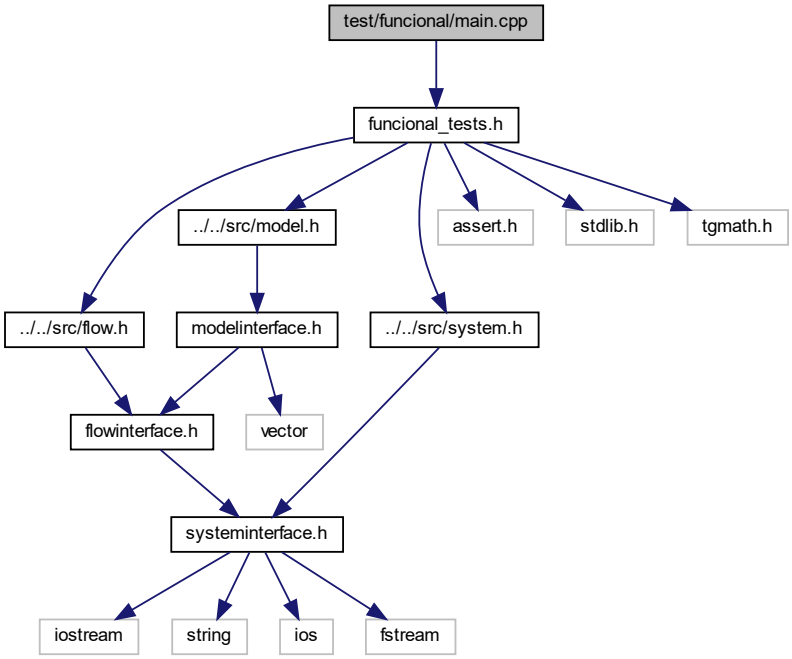
```
00001 #ifndef FUNCTIONAL_TEST
00002 #define FUNCTIONAL_TEST
00003
00004 #include "../../src/flow.h"
00005 #include "../../src/system.h"
00006 #include "../../src/model.h"
00007
00008 #include <assert.h>
00009 #include <stdlib.h>
00010 #include <tgmath.h>
00011
00013
00019
00022 class ExponencialFlow : public Flow {
00023     public:
00030         ExponencialFlow (string name, SystemInterface *source, SystemInterface *target):
00031             Flow (name, source, target) {}
00032
00036         double execute () {
00037             if (getSource() != NULL)
00038                 return 0.01 * getSource()->getValue();
00039             else
00040                 return 0;
00041         }
00042
00043 };
00044
00045
00047
00050 class LogisticFlow : public Flow {
00051     public:
00058         LogisticFlow (string name, SystemInterface *source, SystemInterface *target):
00059             Flow (name, source, target) {}
00060
00064         double execute(){
00065             if (getTarget() != NULL)
00066                 return 0.01 * getTarget()->getValue() * (1 - getTarget()->getValue() / 70);
00067             else
00068                 return 0;
00069         }
00070 };
00071
00072
00074
00077 class ComplexFlowF : public Flow{
00078     public:
00085         ComplexFlowF(string name, SystemInterface *source, SystemInterface *target):
00086             Flow (name, source, target) {}
00087
00091         double execute() {
00092             if (getSource() != NULL)
00093                 return 0.01 * getSource()->getValue();
00094             else
00095                 return 0;
00096         }
00097 };
00098
00100
00103 class ComplexFlowT : public Flow{
00104     public:
00111         ComplexFlowT(string name, SystemInterface *source, SystemInterface *target):
00112             Flow (name, source, target) {}
00113
00117         double execute() {
00118             if (getSource() != NULL)
00119                 return 0.01 * getSource()->getValue();
```

```
00120             else
00121                 return 0;
00122         }
00123 };
00124
00126
00129 class ComplexFlowU : public Flow {
00130     public:
00137         ComplexFlowU(string name, SystemInterface *source, SystemInterface *target):
00138             Flow (name, source, target) {}
00139
00143         double execute(){
00144             if (getSource() != NULL)
00145                 return 0.01 * getSource()->getValue();
00146             else
00147                 return 0;
00148         }
00149 };
00150
00151
00153
00156 class ComplexFlowV : public Flow {
00157     public:
00164         ComplexFlowV(string name, SystemInterface *source, SystemInterface *target):
00165             Flow (name, source, target) {}
00166
00170         double execute(){
00171             if (getSource() != NULL)
00172                 return 0.01 * getSource()->getValue();
00173             else
00174                 return 0;
00175         }
00176 };
00177
00178
00180
00183 class ComplexFlowG : public Flow {
00184     public:
00191         ComplexFlowG(string name, SystemInterface *source, SystemInterface *target):
00192             Flow (name, source, target) {}
00193
00197         double execute(){
00198             if (getSource() != NULL)
00199                 return 0.01 * getSource()->getValue();
00200             else
00201                 return 0;
00202         }
00203 };
00204
00205
00207
00210 class ComplexFlowR: public Flow {
00211     public:
00218         ComplexFlowR(string name, SystemInterface *source, SystemInterface *target):
00219             Flow (name, source, target) {}
00220
00224         double execute(){
00225             if (getSource() != NULL)
00226                 return 0.01 * getSource()->getValue();
00227             else
00228                 return 0;
00229         }
00230 };
00231
00235 void exponentialFuncionalTest ();
00239 void logisticalFuncionalTest ();
00243 void complexFuncionalTest ();
00244
00245 #endif
```

5.23 test/funcional/main.cpp File Reference

```
#include "funcional_tests.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

5.23.1 Function Documentation

5.23.1.1 main()

```
int main ( )
```

Definition at line 3 of file main.cpp.

5.24 main.cpp

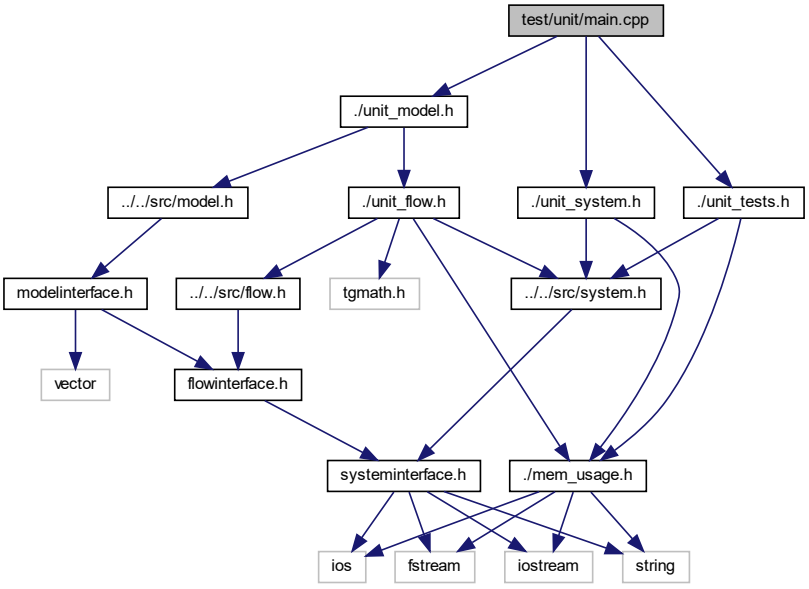
Go to the documentation of this file.

```
00001 #include "funcional_tests.h"
00002
00003 int main() {
00004     cout « "-----Exponential Test-----" « endl;
00005     exponentialFuncionalTest();
00006     cout « "-----Logistical Test-----" « endl;
00007     logisticalFuncionalTest();
00008     cout « "-----Complex Test-----" « endl;
00009     complexFuncionalTest();
00010
00011     return 0;
00012 }
```

5.25 test/unit/main.cpp File Reference

```
#include "./unit_system.h"
#include "./unit_model.h"
#include "./unit_tests.h"
```

Include dependency graph for main.cpp:



Macros

- #define MAIN_UNIT_TESTS

Functions

- int main ()

5.25.1 Macro Definition Documentation

5.25.1.1 MAIN_UNIT_TESTS

```
#define MAIN_UNIT_TESTS
```

Definition at line 2 of file main.cpp.

5.25.2 Function Documentation

5.25.2.1 main()

```
int main ( )
```

Definition at line 8 of file main.cpp.

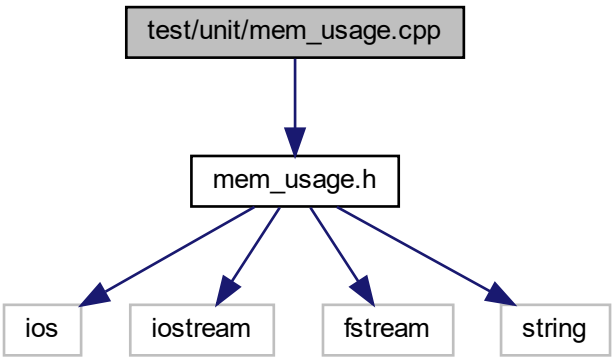
5.26 main.cpp

Go to the documentation of this file.

```
00001 #ifndef MAIN_UNIT_TESTS
00002 #define MAIN_UNIT_TESTS
00003
00004 #include "./unit_system.h"
00005 #include "./unit_model.h"
00006 #include "./unit_tests.h"
00007
00008 int main(){
00009
00010     cout « "\n========== Testes unitarios da Classe System ==========\n" « endl;
00011     run_unit_tests_system();
00012     cout « "\n========== Testes unitarios da Classe Flow ==========\n" « endl;
00013     run_unit_tests_flow();
00014     cout « "\n========== Testes unitarios da Classe Model ==========\n"« endl;
00015     run_unit_tests_model();
00016     cout « "\n========== Testes unitarios de Funcoes Globais ==========\n"« endl;
00017     run_unit_tests_globals();
00018     cout « "\n=========================================================\n" « endl;
00019
00020     return 0;
00021 }
00022
00023 #endif
```

5.27 test/unit/mem_usage.cpp File Reference

`#include "mem_usage.h"`
Include dependency graph for mem_usage.cpp:



Functions

- void memory_usage (double &vm_usage, double &resident_set)

5.27.1 Function Documentation

5.27.1.1 memory_usage()

```
void memory_usage (
            double & vm_usage,
            double & resident_set )
```

Definition at line 5 of file mem_usage.cpp.

5.28 mem_usage.cpp

Go to the documentation of this file.

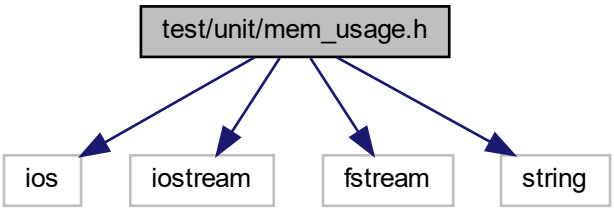
```
00001 #include "mem_usage.h"
00002
00003 using namespace std;
00004
00005 void memory_usage(double& vm_usage, double& resident_set) {
00006     vm_usage = 0.0;
00007     resident_set = 0.0;
00008
00009     ifstream stat_stream("/proc/self/stat", ios_base::in); //get info from proc directory
```

```
00010
00011     // Create variables to get info
00012     string pid, comm, state, ppid, pgrp, session, tty_nr;
00013     string tpgid, flags, minflt, cminflt, majflt, cmajflt;
00014     string utime, stime, cutime, cstime, priority, nice;
00015     string O, itrealvalue, starttime;
00016     unsigned long vsize;
00017     long rss;
00018
00019     stat_stream » pid » comm » state » ppid » pgrp » session » tty_nr
00020                 » tpgid » flags » minflt » cminflt » majflt » cmajflt
00021                 » utime » stime » cutime » cstime » priority » nice
00022                 » O » itrealvalue » starttime » vsize » rss;
00023     stat_stream.close();
00024     vm_usage = vsize / 1024.0;
00025 }
```

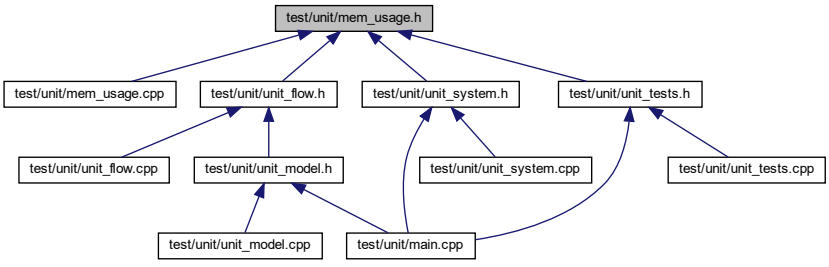
5.29 test/unit/mem_usage.h File Reference

```
#include <ios>
#include <iostream>
#include <fstream>
#include <string>
```

Include dependency graph for mem_usage.h:



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



Functions

- void memory_usage (double &vm_usage, double &resident_set)

5.29.1 Function Documentation

5.29.1.1 memory_usage()

```
void memory_usage (
            double & vm_usage,
            double & resident_set )
```

Definition at line 5 of file mem_usage.cpp.

5.30 mem_usage.h

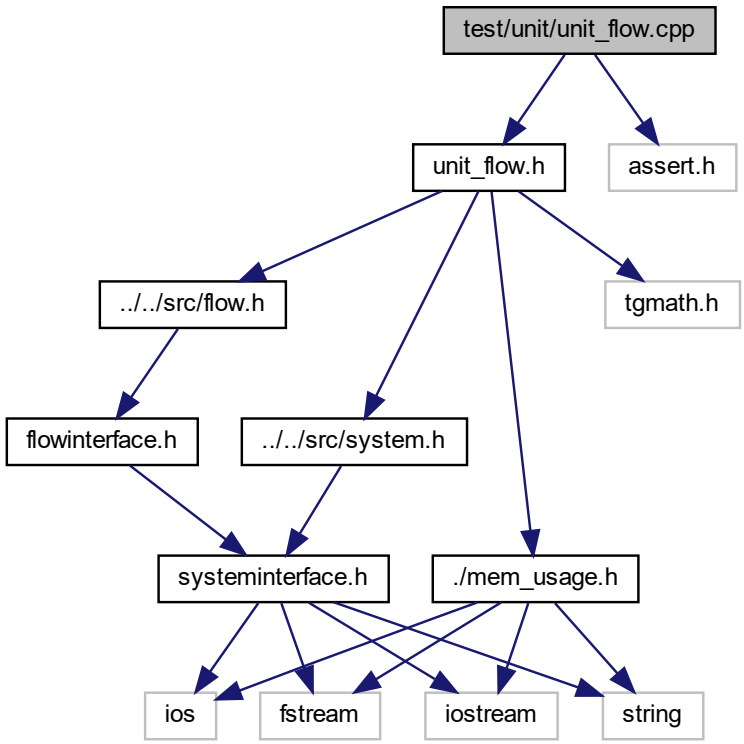
Go to the documentation of this file.

```
00001 #include <ios>
00002 #include <iostream>
00003 #include <fstream>
00004 #include <string>
00005
00006 void memory_usage(double& vm_usage, double& resident_set);
```

5.31 test/unit/unit_flow.cpp File Reference

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

Include dependency graph for unit_flow.cpp:



Functions

- void unit_flow_constructor ()
- void unit_flow_destructor ()
- void unit_flow_getName ()
- void unit_flow_setName ()
- void unit_flow_getSource ()
- void unit_flow_setSource ()
- void unit_flow_clearSource ()
- void unit_flow_getTarget ()
- void unit_flow_setTarget ()
- void unit_flow_clearTarget ()
- void unit_flow_execute ()
- void run_unit_tests_flow ()

5.31.1 Function Documentation

5.31.1.1 run_unit_tests_flow()

```
void run_unit_tests_flow ( )
```

Function prototype for the function that runs all the unit tests of the Flow class.

Definition at line 223 of file unit_flow.cpp.

5.31.1.2 unit_flow_clearSource()

```
void unit_flow_clearSource ( )
```

Function prototype for the Flow class' method clearSource() unit test.

Definition at line 122 of file unit_flow.cpp.

5.31.1.3 unit_flow_clearTarget()

```
void unit_flow_clearTarget ( )
```

Function prototype for the Flow class' method clearTarget() unit test.

Definition at line 166 of file unit_flow.cpp.

5.31.1.4 unit_flow_constructor()

```
void unit_flow_constructor ( )
```

Function prototype for the Flow class' constructor unit test.

Definition at line 5 of file unit_flow.cpp.

5.31.1.5 unit_flow_destructor()

```
void unit_flow_destructor ( )
```

Function prototype for the Flow class' destructor unit test.

Definition at line 45 of file unit_flow.cpp.

5.31.1.6 unit_flow_execute()

```
void unit_flow_execute ( )
```

Function prototype for the Flow class' method execute() unit test.

Definition at line 206 of file unit_flow.cpp.

5.31.1.7 unit_flow_getName()

```
void unit_flow_getName ( )
```

Function prototype for the Flow class' method getName() unit test.

Definition at line 67 of file unit_flow.cpp.

5.31.1.8 unit_flow_getSource()

```
void unit_flow_getSource ( )
```

Function prototype for the Flow class' method getSource() unit test.

Definition at line 93 of file unit_flow.cpp.

5.31.1.9 unit_flow_getTarget()

```
void unit_flow_getTarget ( )
```

Function prototype for the Flow class' method getTarget() unit test.

Definition at line 137 of file unit_flow.cpp.

5.31.1.10 unit_flow_setName()

```
void unit_flow_setName ( )
```

Function prototype for the Flow class' method setName() unit test.

Definition at line 80 of file unit_flow.cpp.

5.31.1.11 unit_flow_setSource()

```
void unit_flow_setSource ( )
```

Function prototype for the Flow class' method setSource() unit test.

Definition at line 108 of file unit_flow.cpp.

5.31.1.12 unit_flow_setTarget()

```
void unit_flow_setTarget ( )
```

Function prototype for the Flow class' method setTarget() unit test.

Definition at line 152 of file unit_flow.cpp.

5.32 unit_flow.cpp

Go to the documentation of this file.

```
00001 #include "unit_flow.h"
00002 #include <assert.h>
00003
00004 // Function for flow's constructor unit test.
00005 void unit_flow_constructor() {
00006     cout « "TEST 1 - Default constructor of the Flow class without passing parameters" « endl;
00007
00008     ExponencialFlow* flow1 = new ExponencialFlow();
00009
00010     // Making assertion to verify if the name property was initialized with the default data.
00011     assert(flow1->getName() == "");
00012     // Making assertion to verify if there isn't a source system in this flow.
00013     assert(flow1->getSource() == NULL);
00014     // Making assertion to verify if there isn't a target system in this flow.
00015     assert(flow1->getTarget() == NULL);
00016
00017     cout « GREEN « "OK!" « RESET « endl;
00018 }
00019
00020 // Function for Flow class's copy constructor unit test.
00021 void UnitFlow::unit_flow_copy_constructor() {
00022     cout « "TEST 2 - Copy constructor of the Flow class" « endl;
00023
00024     SystemInterface* sys1 = new System("Sys 1", 5.0);
00025     SystemInterface* sys2 = new System("Sys 2", 6.0);
00026
00027     ExponencialFlow* flow1 = new ExponencialFlow("Flow 1");
00028     Flow* flow2 = new ExponencialFlow(*flow1);
00029
00030     flow1->setName("Original Flow 1");
00031     flow1->setSource(sys1);
00032     flow1->setTarget(sys2);
00033
00034     // Making assertion to verify if the name property was copied.
00035     assert(flow2->getName() == "Flow 1");
00036     // Making assertion to verify if there isn't a source system in this flow.
00037     assert(flow2->getSource() == NULL);
00038     // Making assertion to verify if there isn't a target system in this flow.
00039     assert(flow2->getTarget() == NULL);
00040
00041     cout « GREEN « "OK!" « RESET « endl;
00042 }
00043
00044 // Function for the Flow class' destructor unit test.
00045 void unit_flow_destructor() {
00046     cout « "TEST 3 - Default destructor of the Flow class" « endl;
00047
00048     double vmBefore, vmAfter, rss;
00049
00050     // Getting the memory usage previous to the creation of a flow.
00051     memory_usage(vmBefore, rss);
00052
00053     ExponencialFlow* flow = new ExponencialFlow("Flow");
00054     delete(flow);
00055
00056     // Getting the memory usage after the creation and destruction of a Flow object.
00057     memory_usage(vmAfter, rss);
00058
00059     // Making assertion to verify if the memory usage after the creation and deletion
00060     // is the same as before the creation of Flow object.
00061     assert(vmBefore == vmAfter);
00062
00063     cout « GREEN « "OK!" « RESET « endl;
00064 }
00065
00066 // Function for Flow class' method getName() unit test.
00067 void unit_flow_getName() {
00068     cout « "TEST 4 - Flow class' getName() method" « endl;
00069
00070     ExponencialFlow* flow = new ExponencialFlow("Flow 1");
00071
00072     // Making assertion to verify if the method returns the Flow class name and if it's
00073     // equal to the parameter previously passed.
00074     assert(flow->getName() == "Flow 1");
00075
00076     cout « GREEN « "OK!" « RESET « endl;
00077 }
00078
00079 // Function for Flow class' method setName() unit test.
00080 void unit_flow_setName() {
00081     cout « "TEST 5 - Flow class' setName() method" « endl;
00082
```

```
00083     ExponencialFlow* flow = new ExponencialFlow();
00084     flow->setName("Test Flow");
00085 
00086     // Making assertion to verify if the data of the name property has been altered.
00087     assert(flow->getName() == "Test Flow");
00088 
00089     cout « GREEN « "OK!" « RESET « endl;
00090 }
00091 
00092 // Function for Flow class' method getSource() unit test.
00093 void unit_flow_getSource(){
00094     cout « "TEST 6 - Flow class' getSource() method" « endl;
00095 
00096     SystemInterface* system = new System("Test System");
00097     ExponencialFlow* flow = new ExponencialFlow("Flow 1");
00098     flow->setSource(system);
00099 
00100     // Making assertion to verify if the method returns the Flow class source system and if it's
00101     // equal to the parameter previously passed.
00102     assert(flow->getSource()->getName() == "Test System");
00103 
00104     cout « GREEN « "OK!" « RESET « endl;
00105 }
00106 
00107 // Function for Flow class' method setSource() unit test.
00108 void unit_flow_setSource(){
00109     cout « "TEST 7 - Flow class' setSource() method" « endl;
00110 
00111     SystemInterface* system = new System("Test System");
00112     ExponencialFlow* flow1 = new ExponencialFlow("Flow 1");
00113     flow1->setSource(system);
00114 
00115     // Making assertion to verify if the data of the source system property has been altered.
00116     assert(flow1->getSource()->getName() == "Test System");
00117 
00118     cout « GREEN « "OK!" « RESET « endl;
00119 }
00120 
00121 // Function for Flow class' method clearSource() unit test.
00122 void unit_flow_clearSource(){
00123     cout « "TEST 8 - Flow class' clearSource() method" « endl;
00124 
00125     SystemInterface* system = new System("Test System");
00126     ExponencialFlow* flow1 = new ExponencialFlow("Flow 1");
00127     flow1->setSource(system);
00128     flow1->clearSource();
00129 
00130     // Making assertion to verify if the data of the source system property has been altered.
00131     assert(flow1->getSource() == NULL);
00132 
00133     cout « GREEN « "OK!" « RESET « endl;
00134 }
00135 
00136 // Function for Flow class' method getTarget() unit test.
00137 void unit_flow_getTarget(){
00138     cout « "TEST 9 - Flow class' getTarget() method" « endl;
00139 
00140     SystemInterface* system = new System("Test System");
00141     ExponencialFlow* flow = new ExponencialFlow("Flow 1");
00142     flow->setTarget(system);
00143 
00144     // Making assertion to verify if the method returns the Flow class target system and if it's
00145     // equal to the parameter previously passed.
00146     assert(flow->getTarget()->getName() == "Test System");
00147 
00148     cout « GREEN « "OK!" « RESET « endl;
00149 }
00150 
00151 // Function for Flow class' method setTarget() unit test.
00152 void unit_flow_setTarget(){
00153     cout « "TEST 10 - Flow class' setTarget() method" « endl;
00154 
00155     SystemInterface* system = new System("Test System");
00156     ExponencialFlow* flow = new ExponencialFlow("Flow 1");
00157     flow->setTarget(system);
00158 
00159     // Making assertion to verify if the data of the target system property has been altered.
00160     assert(flow->getTarget()->getName() == "Test System");
00161 
00162     cout « GREEN « "OK!" « RESET « endl;
00163 }
00164 
00165 // Function for Flow class' method clearTarget() unit test.
00166 void unit_flow_clearTarget(){
00167     cout « "TEST 11 - Flow class' clearTarget() method" « endl;
00168 
00169     SystemInterface* system = new System("Test System");
```

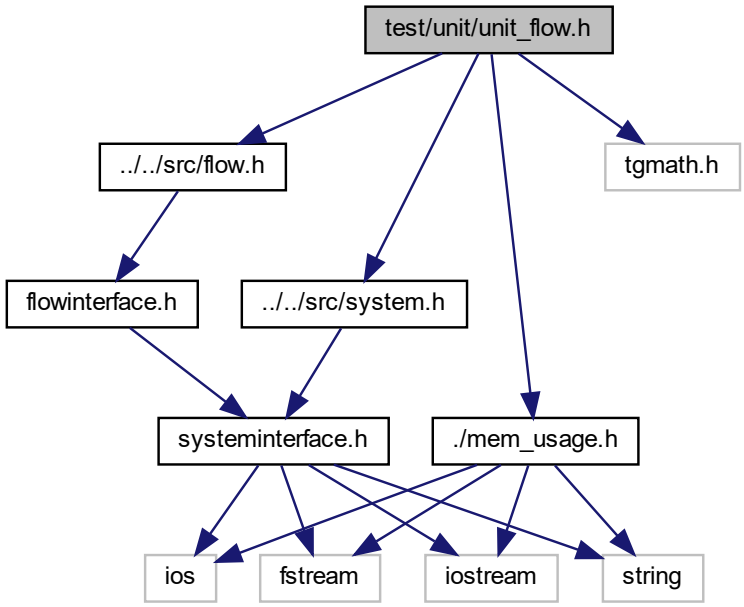
```
00170     ExponencialFlow* flow = new ExponencialFlow("Flow");
00171     flow->setTarget(system);
00172     flow->clearTarget();
00173 
00174     // Making assertion to verify if the data of the target system property has been altered.
00175     assert(flow->getTarget() == NULL);
00176 
00177     cout « GREEN « "OK!" « RESET « endl;
00178 }
00179 
00180 // Function for Flow class' assingment operator unit test.
00181 void UnitFlow::unit_flow_assingmentOperator(){
00182     cout « "TEST 12 - Flow class' assignment operator" « endl;
00183 
00184     SystemInterface* sys1 = new System("Sys 1", 5.0);
00185     SystemInterface* sys2 = new System("Sys 2", 6.0);
00186 
00187     ExponencialFlow* flow1 = new ExponencialFlow("Flow 1");
00188     ExponencialFlow* flow2 = new ExponencialFlow();
00189     *flow2 = *flow1;
00190 
00191     flow1->setName("Original Flow 1");
00192     flow1->setSource(sys1);
00193     flow1->setTarget(sys2);
00194 
00195     // Making assertion to verify if the name property was assigned.
00196     assert(flow2->getName() == "Flow 1");
00197     // Making assertion to verify if the source system property was assigned.
00198     assert(flow2->getSource() == NULL);
00199     // Making assertion to verify if the target system property was assigned.
00200     assert(flow2->getTarget() == NULL);
00201 
00202     cout « GREEN « "OK!" « RESET « endl;
00203 }
00204 
00205 // Function for Flow class' execute method unit test.
00206 void unit_flow_execute(){
00207     cout « "TEST 13 - Flow class' execute() method" « endl;
00208 
00209     SystemInterface* system1 = new System("Test System 1", 10.0);
00210     SystemInterface* system2 = new System("Test System 2", 0.0);
00211     ExponencialFlow* flow = new ExponencialFlow("Flow");
00212     flow->setSource(system1);
00213     flow->setTarget(system2);
00214     system2->setValue(flow->execute());
00215 
00216     // Making assertion to verify if the execute method has been completed successfully.
00217     assert(abs(flow->getTarget()->getValue() - 0.1) < 0.01);
00218 
00219     cout « GREEN « "OK!" « RESET « endl;
00220 }
00221 
00222 // Function to run all the Flow class' unit tests.
00223 void run_unit_tests_flow() {
00224 
00225     UnitFlow* unit_flow = new UnitFlow();
00226 
00227     // Calling all the Flow class' unit test functions.
00228     unit_flow_constructor();
00229     unit_flow->unit_flow_copy_constructor();
00230     unit_flow_destructor();
00231     unit_flow_getName();
00232     unit_flow_setName();
00233     unit_flow_getSource();
00234     unit_flow_setSource();
00235     unit_flow_clearSource();
00236     unit_flow_getTarget();
00237     unit_flow_setTarget();
00238     unit_flow_clearTarget();
00239     unit_flow->unit_flow_assingmentOperator();
00240     unit_flow_execute();
00241 
00242     delete(unit_flow);
00243 }
```

5.33 test/unit/unit_flow.h File Reference

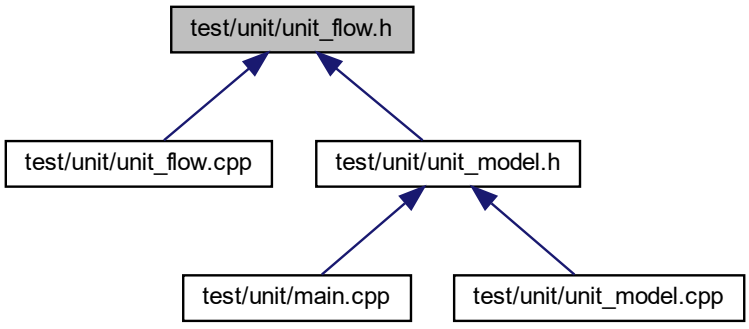
```
#include "../../src/system.h"
#include "../../src/flow.h"
#include "./mem_usage.h"
```

```
#include <tgmath.h>
```

Include dependency graph for unit_flow.h:



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



Classes

- class UnitFlow

  Unit tests.
- class ExponencialFlow

  Functional tests.

Macros

- #define RESET "\033[0m" /∗! Escape sequence to reset color output to default. ∗/
- #define GREEN "\033[32m" /∗! Escape sequence to a green color output. ∗/

Functions

- void unit_flow_constructor ()
- void unit_flow_destructor ()
- void unit_flow_getName ()
- void unit_flow_setName ()
- void unit_flow_getSource ()
- void unit_flow_setSource ()
- void unit_flow_clearSource ()
- void unit_flow_getTarget ()
- void unit_flow_setTarget ()
- void unit_flow_clearTarget ()
- void unit_flow_execute ()
- void run_unit_tests_flow ()

5.33.1 Macro Definition Documentation

5.33.1.1 GREEN

```
#define GREEN "\033[32m" /*!  Escape sequence to a green color output.  */
```

Definition at line 11 of file unit_flow.h.

5.33.1.2 RESET

```
#define RESET "\033[0m" /*!  Escape sequence to reset color output to default.  */
```

Definition at line 10 of file unit_flow.h.

5.33.2 Function Documentation

5.33.2.1 run_unit_tests_flow()

```
void run_unit_tests_flow ( )
```

Function prototype for the function that runs all the unit tests of the Flow class.

Definition at line 223 of file unit_flow.cpp.

5.33.2.2 unit_flow_clearSource()

```
void unit_flow_clearSource ( )
```

Function prototype for the Flow class' method clearSource() unit test.

Definition at line 122 of file unit_flow.cpp.

5.33.2.3 unit_flow_clearTarget()

```
void unit_flow_clearTarget ( )
```

Function prototype for the Flow class' method clearTarget() unit test.

Definition at line 166 of file unit_flow.cpp.

5.33.2.4 unit_flow_constructor()

```
void unit_flow_constructor ( )
```

Function prototype for the Flow class' constructor unit test.

Definition at line 5 of file unit_flow.cpp.

5.33.2.5 unit_flow_destructor()

```
void unit_flow_destructor ( )
```

Function prototype for the Flow class' destructor unit test.

Definition at line 45 of file unit_flow.cpp.

5.33.2.6 unit_flow_execute()

```
void unit_flow_execute ( )
```

Function prototype for the Flow class' method execute() unit test.

Definition at line 206 of file unit_flow.cpp.

5.33.2.7 unit_flow_getName()

```
void unit_flow_getName ( )
```

Function prototype for the Flow class' method getName() unit test.

Definition at line 67 of file unit_flow.cpp.

5.33.2.8 unit_flow_getSource()

```
void unit_flow_getSource ( )
```

Function prototype for the Flow class' method getSource() unit test.

Definition at line 93 of file unit_flow.cpp.

5.33.2.9 unit_flow_getTarget()

```
void unit_flow_getTarget ( )
```

Function prototype for the Flow class' method getTarget() unit test.

Definition at line 137 of file unit_flow.cpp.

5.33.2.10 unit_flow_setName()

```
void unit_flow_setName ( )
```

Function prototype for the Flow class' method setName() unit test.

Definition at line 80 of file unit_flow.cpp.

5.33.2.11 unit_flow_setSource()

```
void unit_flow_setSource ( )
```

Function prototype for the Flow class' method setSource() unit test.

Definition at line 108 of file unit_flow.cpp.

5.33.2.12 unit_flow_setTarget()

```
void unit_flow_setTarget ( )
```

Function prototype for the Flow class' method setTarget() unit test.

Definition at line 152 of file unit_flow.cpp.

5.34 unit_flow.h

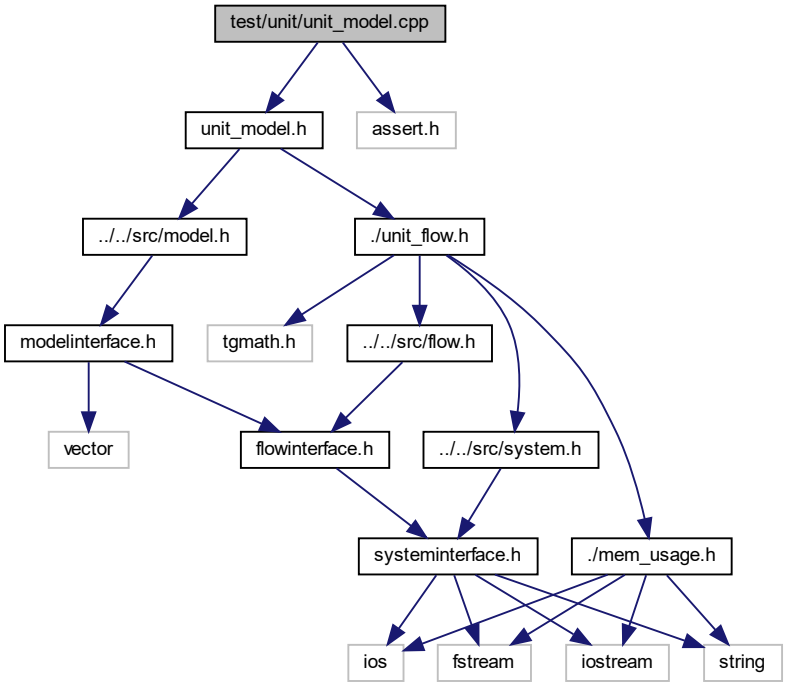
Go to the documentation of this file.

```
00001 #ifndef UNIT_FLOW
00002 #define UNIT_FLOW
00003
00004 #include "../../src/system.h"
00005 #include "../../src/flow.h"
00006 #include "./mem_usage.h"
00007
00008 #include <tgmath.h>
00009
00010 #define RESET "\033[0m"
00011 #define GREEN "\033[32m"
00012 using namespace std;
00013
00015
00019 class UnitFlow{
00020
00021     public:
00022         UnitFlow(){}
00023         ~UnitFlow(){}
00024
00028         void unit_flow_copy_constructor();
00032         void unit_flow_assingmentOperator();
00033
00034 };
00035
00037
00040 class ExponencialFlow : public Flow{
00041     public:
00048         ExponencialFlow(string name = "", System* source = NULL, System* target = NULL): Flow(name,
      source, target){}
00049
00053         double execute(){
00054             if (getSource() != NULL){
00055                 return (0.01 * getSource()->getValue());
00056             }
00057             else{
00058                 return 0;
00059             }
00060         }
00061 };
00062
00066 void unit_flow_constructor();
00067
00071 void unit_flow_destructor();
00072
00076 void unit_flow_getName();
00077
00081 void unit_flow_setName();
00082
00086 void unit_flow_getSource();
00087
00091 void unit_flow_setSource();
00092
00096 void unit_flow_clearSource();
00097
00101 void unit_flow_getTarget();
00102
00106 void unit_flow_setTarget();
00107
00111 void unit_flow_clearTarget();
00112
00116 void unit_flow_execute();
00117
00121 void run_unit_tests_flow();
00122
00123 #endif
```

5.35 test/unit/unit_model.cpp File Reference

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

Include dependency graph for unit_model.cpp:



Functions

- void unit_model_constructor ()

  Unit tests.
- void unit_model_destructor ()
- void unit_model_getName ()
- void unit_model_setName ()
- void unit_model_getTime ()
- void unit_model_setTime ()
- void unit_model_incrementTime ()
- void unit_model_addSystem ()
- void unit_model_removeSystem ()
- void unit_model_addFlow ()
- void unit_model_removeFlow ()
- void unit_model_execute ()
- void run_unit_tests_model ()

5.35.1 Function Documentation

5.35.1.1 run_unit_tests_model()

```
void run_unit_tests_model ( )
```

Function prototype for the function that runs all the unit tests of the Model class.

Definition at line 243 of file unit_model.cpp.

5.35.1.2 unit_model_addFlow()

```
void unit_model_addFlow ( )
```

Function prototype for the Model class' method addFlow() unit test.

Definition at line 179 of file unit_model.cpp.

5.35.1.3 unit_model_addSystem()

```
void unit_model_addSystem ( )
```

Function prototype for the Model class' method addSystem() unit test.

Definition at line 148 of file unit_model.cpp.

5.35.1.4 unit_model_constructor()

```
void unit_model_constructor ( )
```

Unit tests.

Creation of the unit tests for the Model class.

Function prototype for the Model class' constructor unit test.

Definition at line 7 of file unit_model.cpp.

5.35.1.5 unit_model_destructor()

```
void unit_model_destructor ( )
```

Function prototype for the Model class' destructor unit test.

Definition at line 53 of file unit_model.cpp.

5.35.1.6 unit_model_execute()

```
void unit_model_execute ( )
```

Definition at line 208 of file unit_model.cpp.

5.35.1.7 unit_model_getName()

```
void unit_model_getName ( )
```

Function prototype for the Model class' method getName() unit test.

Definition at line 83 of file unit_model.cpp.

5.35.1.8 unit_model_getTime()

```
void unit_model_getTime ( )
```

Function prototype for the Model class' method getTime() unit test.

Definition at line 109 of file unit_model.cpp.

5.35.1.9 unit_model_incrementTime()

```
void unit_model_incrementTime ( )
```

Function prototype for the Model class' method incrementTime() unit test.

Definition at line 135 of file unit_model.cpp.

5.35.1.10 unit_model_removeFlow()

```
void unit_model_removeFlow ( )
```

Function prototype for the Model class' method removeFlow() unit test.

Definition at line 193 of file unit_model.cpp.

5.35.1.11 unit_model_removeSystem()

```
void unit_model_removeSystem ( )
```

Function prototype for the Model class' method removeSystem() unit test.

Definition at line 163 of file unit_model.cpp.

5.35.1.12 unit_model_setName()

```
void unit_model_setName ( )
```

Function prototype for the Model class' method setName() unit test.

Definition at line 96 of file unit_model.cpp.

5.35.1.13 unit_model_setTime()

```
void unit_model_setTime ( )
```

Function prototype for the Model class' method setTime() unit test.

Definition at line 122 of file unit_model.cpp.

5.36 unit_model.cpp

Go to the documentation of this file.

```
00001 #include "unit_model.h"
00002 #include <assert.h>
00003
00004 using namespace std;
00005
00006 // Function for Model class's constructor unit test.
00007 void unit_model_constructor(){
00008     cout « "TEST 1 - Default constructor of the Model class without passing parameters" « endl;
00009
00010     ModelInterface* model1 =  new Model();
00011
00012     // Making assertion to verify if the name property was initialized with the default data.
00013     assert(model1->getName() == "");
00014     // Making assertion to verify if the time property was initialized with the default data.
00015     assert(model1->getTime() == 0.0);
00016
00017     cout « GREEN « "OK!" « RESET « endl;
00018     cout « "TEST 2 - Default constructor of the Model class with passing parameters" « endl;
00019
00020     SystemInterface* system1 = new System("System 1");
00021     SystemInterface* system2 = new System("System 2");
00022
00023     vector<SystemInterface*> systems;
00024
00025     systems.push_back(system1);
00026     systems.push_back(system2);
00027
00028     ExponencialFlow* flow1 = new ExponencialFlow("Flow 1");
00029     ModelInterface* model2 = new Model("Test Model", 1.0);
00030
00031     model2->add(system1);
00032     model2->add(system2);
```

```
00033     model2->add(flow1);
00034
00035     // Making assertion to verify if the name property was initialized with the parameter specified.
00036     assert(model2->getName() == "Test Model");
00037     // Making assertion to verify if the time property was initialized with the parameter specified.
00038     assert(model2->getTime() == 1.0);
00039     // Making assertion to verify if the Flow object was added to the Model's flows parameter.
00040     assert((*(model2->beginFlows()))->getName() == "Flow 1");
00041
00042     int counter = 0;
00043     for (auto sys = model2->beginSystems(); sys != model2->endSystems(); ++sys){
00044         // Making assertion to verify if the systems were added to the systems property.
00045         assert((*sys)->getName() == systems[counter]->getName());
00046         counter++;
00047     }
00048
00049     cout << GREEN << "OK!" << RESET << endl;
00050 }
00051
00052 // Function for the Model class' destructor unit test.
00053 void unit_model_destructor(){
00054     cout << "TEST 3 - Default destructor of the Model class" << endl;
00055
00056     double vmBefore, vmAfter, rss;
00057
00058     // Getting the memory usage previous to the creation of a model.
00059     memory_usage(vmBefore, rss);
00060
00061     SystemInterface* system1 = new System("System 1");
00062     SystemInterface* system2 = new System("System 2");
00063     ExponencialFlow* flow1 = new ExponencialFlow("Flow 1");
00064     ModelInterface* model = new Model("Test Model", 1.0);
00065
00066     model->add(system1);
00067     model->add(system2);
00068     model->add(flow1);
00069
00070     delete model;
00071
00072     // Getting the memory usage after the creation and destruction of a Flow object.
00073     memory_usage(vmAfter, rss);
00074
00075     // Making assertion to verify if the memory usage after the creation and deletion
00076     // is the same as before the creation of Model object.
00077     assert(vmBefore == vmAfter);
00078
00079     cout << GREEN << "OK!" << RESET << endl;
00080 }
00081
00082 // Function for the Model class' getName() method unit test.
00083 void unit_model_getName(){
00084     cout << "TEST 4 - Model class' getName() method" << endl;
00085
00086     ModelInterface* model = new Model("Test Model", 0.0);
00087
00088     // Making assertion to verify if the method returns the Model class name and if it's
00089     // equal to the parameter previously passed.
00090     assert(model->getName() == "Test Model");
00091
00092     cout << GREEN << "OK!" << RESET << endl;
00093 }
00094
00095 // Function for the Model class' setName() method unit test.
00096 void unit_model_setName(){
00097     cout << "TEST 5 - Model class' setName() method" << endl;
00098
00099     ModelInterface* model = new Model();
00100     model->setName("Test Model");
00101
00102     // Making assertion to verify if the data of the name property has been altered.
00103     assert(model->getName() == "Test Model");
00104
00105     cout << GREEN << "OK!" << RESET << endl;
00106 }
00107
00108 // Function for the Model class' getTime() method unit test.
00109 void unit_model_getTime(){
00110     cout << "TEST 6 - Model class' getTime() method" << endl;
00111
00112     ModelInterface* model = new Model("Test Model", 0.0);
00113
00114     // Making assertion to verify if the method returns the Model class time and if it's
00115     // equal to the parameter previously passed.
00116     assert(model->getTime() == 0.0);
00117
00118     cout << GREEN << "OK!" << RESET << endl;
00119 }
```

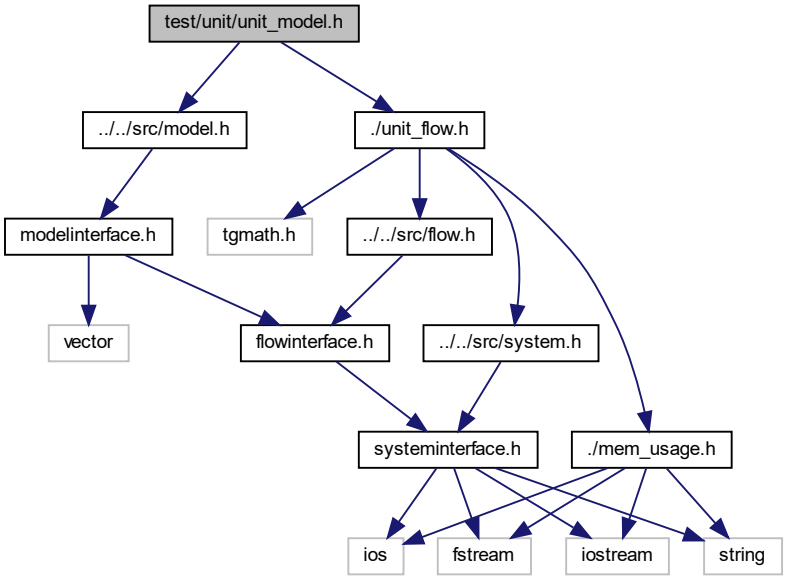
```
00120
00121 // Function for the Model class' setTime() method unit test.
00122 void unit_model_setTime(){
00123     cout « "TEST 7 - Model class' setTime() method" « endl;
00124
00125     ModelInterface* model = new Model("Test Model", 0.0);
00126     model->setTime(1.0);
00127
00128     // Making assertion to verify if the data of the time property has been altered.
00129     assert(model->getTime() == 1.0);
00130
00131     cout « GREEN « "OK!" « RESET « endl;
00132 }
00133
00134 // Function for the Model class' incrementTime() method unit test.
00135 void unit_model_incrementTime(){
00136     cout « "TEST 8 - Model class' incrementTime() method" « endl;
00137
00138     ModelInterface* model = new Model("Test Model", 1.0);
00139     model->incrementTime(1.0);
00140
00141     // Making assertion to verify if the data of the time property has been incremented.
00142     assert(model->getTime() == 2.0);
00143
00144     cout « GREEN « "OK!" « RESET « endl;
00145 }
00146
00147 // Function for the Model class' addSystem() method unit test.
00148 void unit_model_addSystem(){
00149     cout « "TEST 9 - Model class' addSystem() method" « endl;
00150
00151     SystemInterface* system = new System("System 1");
00152
00153     ModelInterface* model = new Model("Test Model", 1.0);
00154     model->add(system);
00155
00156     // Making assertion to verify if the system has been added to the systems property.
00157     assert((*(model->beginSystems()))->getName() == system->getName());
00158
00159     cout « GREEN « "OK!" « RESET « endl;
00160 }
00161
00162 // Function for the Model class' removeSystem() method unit test.
00163 void unit_model_removeSystem(){
00164     cout « "TEST 10 - Model class' removeSystem() method" « endl;
00165
00166     SystemInterface* system = new System("System 1");
00167
00168     ModelInterface* model = new Model("Test Model", 1.0);
00169     model->add(system);
00170     model->remove(system);
00171
00172     // Making assertion to verify if the system has been removed from the systems property.
00173     assert(model->beginSystems() == model->endSystems());
00174
00175     cout « GREEN « "OK!" « RESET « endl;
00176 }
00177
00178 // Function for the Model class' addFlow() method unit test.
00179 void unit_model_addFlow(){
00180     cout « "TEST 11 - Model class' addFlow() method" « endl;
00181
00182     ExponencialFlow* flow = new ExponencialFlow("Flow 1");
00183     ModelInterface* model = new Model("Test Model", 1.0);
00184     model->add(flow);
00185
00186     // Making assertion to verify if the flow has been added to the flows property.
00187     assert((*(model->beginFlows()))->getName() == flow->getName());
00188
00189     cout « GREEN « "OK!" « RESET « endl;
00190 }
00191
00192 // Function for the Model class' removeFlow() method unit test.
00193 void unit_model_removeFlow(){
00194     cout « "TEST 12 - Model class' removeFlow() method" « endl;
00195
00196     ExponencialFlow* flow = new ExponencialFlow("Flow 1");
00197     ModelInterface* model = new Model("Test Model", 1.0);
00198     model->add(flow);
00199     model->remove(flow);
00200
00201     // Making assertion to verify if the system has been removed from the systems property.
00202     assert(model->beginFlows() == model->endFlows());
00203
00204     cout « GREEN « "OK!" « RESET « endl;
00205 }
00206
```

```
00207 // Function for the Model class' execute() method unit test.
00208 void unit_model_execute(){
00209     cout « "TEST 13 - Model class' execute() method" « endl;
00210 
00211     SystemInterface* pop1 = new System("Population 1", 100.0);
00212     SystemInterface* pop2 = new System("Population 2", 0.0);
00213     ExponencialFlow* expFlow = new ExponencialFlow("Unlimited Growth");
00214     expFlow->setSource (pop1);
00215     expFlow->setTarget (pop2);
00216     ModelInterface* expModel = new Model("Exponential Model", 0.0);
00217 
00218     expModel->add(pop1);
00219     expModel->add(pop2);
00220     expModel->add(expFlow);
00221 
00222     // Making assertions for the unit test before execution
00223     assert(pop1->getName() == "Population 1");
00224     assert(pop2->getName() == "Population 2");
00225     assert(expFlow->getName() == "Unlimited Growth");
00226     assert(expModel->getName() == "Exponential Model");
00227 
00228     assert(abs(pop1->getValue() - 100.0) < 0.0001);
00229     assert(abs(pop2->getValue() - 0.0) < 0.0001);
00230     assert(abs(expModel->getTime() - 0.0) < 0.0001);
00231 
00232     expModel->execute(0, 100, 1);
00233 
00234     // Making assertions for the unit test after execution
00235     assert(abs(pop1->getValue() - 36.6032) < 0.0001);
00236     assert(abs(pop2->getValue() - 63.3968) < 0.0001);
00237     assert(abs(expModel->getTime() - 100.0) < 0.0001);
00238 
00239     cout « GREEN « "OK!" « RESET « endl;
00240 }
00241 
00242 // Function to run all the Model class' unit tests.
00243 void run_unit_tests_model(){
00244 
00245     // Calling all the Model class' unit test functions.
00246     unit_model_constructor();
00247     unit_model_destructor();
00248     unit_model_getName();
00249     unit_model_setName();
00250     unit_model_getTime();
00251     unit_model_setTime();
00252     unit_model_incrementTime();
00253     unit_model_addSystem();
00254     unit_model_removeSystem();
00255     unit_model_addFlow();
00256     unit_model_removeFlow();
00257     unit_model_execute();
00258 
00259 }
```

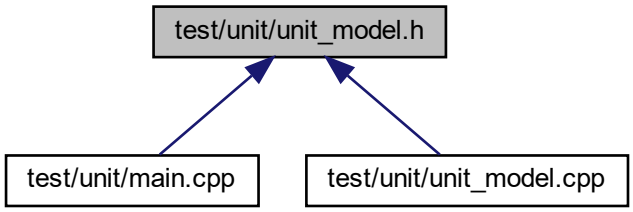
5.37 test/unit/unit_model.h File Reference

```
#include "../../src/model.h"
#include "./unit_flow.h"
```

Include dependency graph for unit_model.h:



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



Macros

- #define RESET "\033[0m" /∗! Escape sequence to reset color output to default. ∗/
- #define GREEN "\033[32m" /∗! Escape sequence to a green color output. ∗/

Functions

- void unit_model_constructor ()

  Unit tests.
- void unit_model_destructor ()

- void unit_model_addSystem ()
- void unit_model_removeSystem ()
- void unit_model_addFlow ()
- void unit_model_removeFlow ()
- void unit_model_getName ()
- void unit_model_setName ()
- void unit_model_getTime ()
- void unit_model_setTime ()
- void unit_model_incrementTime ()
- void run_unit_tests_model ()

5.37.1 Macro Definition Documentation

5.37.1.1 GREEN

```
#define GREEN "\033[32m" /*!  Escape sequence to a green color output.  */
```

Definition at line 8 of file unit_model.h.

5.37.1.2 RESET

```
#define RESET "\033[0m" /*!  Escape sequence to reset color output to default.  */
```

Definition at line 7 of file unit_model.h.

5.37.2 Function Documentation

5.37.2.1 run_unit_tests_model()

```
void run_unit_tests_model ( )
```

Function prototype for the function that runs all the unit tests of the Model class.

Definition at line 243 of file unit_model.cpp.

5.37.2.2 unit_model_addFlow()

```
void unit_model_addFlow ( )
```

Function prototype for the Model class' method addFlow() unit test.

Definition at line 179 of file unit_model.cpp.

5.37.2.3 unit_model_addSystem()

```
void unit_model_addSystem ( )
```

Function prototype for the Model class' method addSystem() unit test.

Definition at line 148 of file unit_model.cpp.

5.37.2.4 unit_model_constructor()

```
void unit_model_constructor ( )
```

Unit tests.

Creation of the unit tests for the Model class.

Function prototype for the Model class' constructor unit test.

Definition at line 7 of file unit_model.cpp.

5.37.2.5 unit_model_destructor()

```
void unit_model_destructor ( )
```

Function prototype for the Model class' destructor unit test.

Definition at line 53 of file unit_model.cpp.

5.37.2.6 unit_model_getName()

```
void unit_model_getName ( )
```

Function prototype for the Model class' method getName() unit test.

Definition at line 83 of file unit_model.cpp.

5.37.2.7 unit_model_getTime()

```
void unit_model_getTime ( )
```

Function prototype for the Model class' method getTime() unit test.

Definition at line 109 of file unit_model.cpp.

5.37.2.8 unit_model_incrementTime()

```
void unit_model_incrementTime ( )
```

Function prototype for the Model class' method incrementTime() unit test.

Definition at line 135 of file unit_model.cpp.

5.37.2.9 unit_model_removeFlow()

```
void unit_model_removeFlow ( )
```

Function prototype for the Model class' method removeFlow() unit test.

Definition at line 193 of file unit_model.cpp.

5.37.2.10 unit_model_removeSystem()

```
void unit_model_removeSystem ( )
```

Function prototype for the Model class' method removeSystem() unit test.

Definition at line 163 of file unit_model.cpp.

5.37.2.11 unit_model_setName()

```
void unit_model_setName ( )
```

Function prototype for the Model class' method setName() unit test.

Definition at line 96 of file unit_model.cpp.

5.37.2.12 unit_model_setTime()

```
void unit_model_setTime ( )
```

Function prototype for the Model class' method setTime() unit test.

Definition at line 122 of file unit_model.cpp.

5.38 unit_model.h

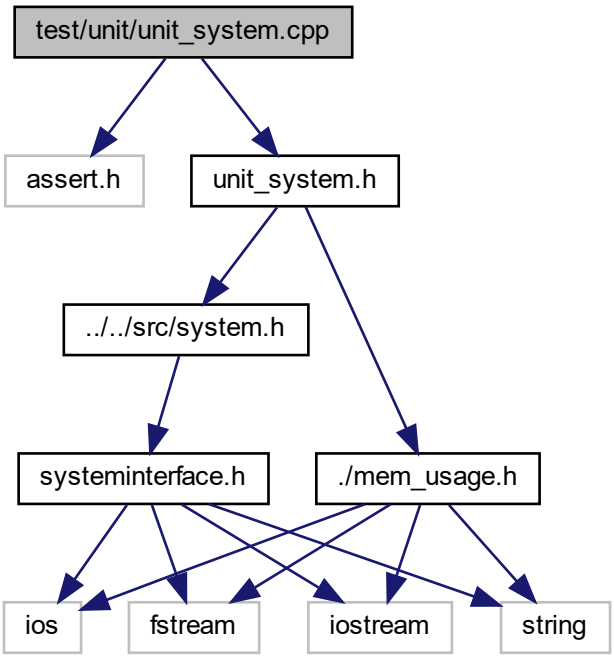
Go to the documentation of this file.

```
00001 #ifndef UNIT_MODEL
00002 #define UNIT_MODEL
00003
00004 #include "../../src/model.h"
00005 #include "./unit_flow.h"
00006
00007 #define RESET "\033[0m"
00008 #define GREEN "\033[32m"
00011
00018 void unit_model_constructor();
00019
00023 void unit_model_destructor();
00024
00028 void unit_model_addSystem();
00029
00033 void unit_model_removeSystem();
00034
00038 void unit_model_addFlow();
00039
00043 void unit_model_removeFlow();
00044
00048 void unit_model_getName();
00049
00053 void unit_model_setName();
00054
00058 void unit_model_getTime();
00059
00063 void unit_model_setTime();
00064
00068 void unit_model_incrementTime();
00069
00073 void run_unit_tests_model();
00074
00075 #endif
```

5.39 test/unit/unit_system.cpp File Reference

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

Include dependency graph for unit_system.cpp:



Functions

- void unit_system_constructor ()
- void unit_system_destructor ()
- void unit_system_getName ()
- void unit_system_setName ()
- void unit_system_getValue ()
- void unit_system_setValue ()
- void unit_system_sumOperator ()
- void unit_system_minusOperator ()
- void unit_system_timesOperator ()
- void unit_system_divisionOperator ()
- void run_unit_tests_system ()

5.39.1 Function Documentation

5.39.1.1 run_unit_tests_system()

```
void run_unit_tests_system ( )
```

Function prototype for the function that runs all the unit tests of the System class.

Definition at line 190 of file unit_system.cpp.

5.39.1.2 unit_system_constructor()

```
void unit_system_constructor ( )
```

Function prototype for the System class' constructor unit test.

Definition at line 7 of file unit_system.cpp.

5.39.1.3 unit_system_destructor()

```
void unit_system_destructor ( )
```

Function prototype for the System class' destructor unit test.

Definition at line 49 of file unit_system.cpp.

5.39.1.4 unit_system_divisionOperator()

```
void unit_system_divisionOperator ( )
```

Function prototype for the System class' "/" operator unit test.

Definition at line 178 of file unit_system.cpp.

5.39.1.5 unit_system_getName()

```
void unit_system_getName ( )
```

Function prototype for the System class' method getName() unit test.

Definition at line 71 of file unit_system.cpp.

5.39.1.6 unit_system_getValue()

```
void unit_system_getValue ( )
```

Function prototype for the System class' method getValue() unit test.

Definition at line 97 of file unit_system.cpp.

5.39.1.7 unit_system_minusOperator()

```
void unit_system_minusOperator ( )
```

Function prototype for the System class' "-" operator unit test.

Definition at line 154 of file unit_system.cpp.

5.39.1.8 unit_system_setName()

```
void unit_system_setName ( )
```

Function prototype for the System class' method setName() unit test.

Definition at line 84 of file unit_system.cpp.

5.39.1.9 unit_system_setValue()

```
void unit_system_setValue ( )
```

Function prototype for the System class' method setValue() unit test.

Definition at line 110 of file unit_system.cpp.

5.39.1.10 unit_system_sumOperator()

```
void unit_system_sumOperator ( )
```

Function prototype for the System class' "+" operator unit test.

Definition at line 142 of file unit_system.cpp.

5.39.1.11 unit_system_timesOperator()

```
void unit_system_timesOperator ( )
```

Function prototype for the System class' "$*$" operator unit test.

Definition at line 166 of file unit_system.cpp.

5.40 unit_system.cpp

Go to the documentation of this file.

```
00001 #include <assert.h>
00002 #include "unit_system.h"
00003
00004 using namespace std;
00005
00006 // Function for System class's constructor unit test.
00007 void unit_system_constructor(){
00008     cout « "TEST 1 - Default constructor of the System class without passing parameters" « endl;
00009
00010     SystemInterface* system1 = new System();
00011     // Making assertion to verify if the name property was initialized with the default data.
00012     assert(system1->getName() == "");
00013     // Making assertion to verify if the value property was initialized with the default data.
00014     assert(system1->getValue() == 0.0);
00015
00016     cout « GREEN « "OK!" « RESET « endl;
00017
00018     cout « "TEST 2 - Default constructor of the System class with passing parameters" « endl;
00019
00020     SystemInterface* system2 = new System("Test System", 10.0);
00021     // Making assertion to verify if the name property was initialized with the parameter specified.
00022     assert(system2->getName() == "Test System");
00023     // Making assertion to verify if the value property was initialized with the parameter specified.
00024     assert(system2->getValue() == 10.0);
00025
00026     cout « GREEN « "OK!"« RESET « endl;
00027 }
00028
00029 // Function for System class's copy constructor unit test.
00030 void UnitSystem::unit_system_copy_constructor(){
00031     cout « "TEST 3 - Copy constructor of the System class" « endl;
00032
00033     System* system1 = new System("Test System", 10.0);
00034     SystemInterface* system2 = new System(*system1);
00035     system1->setName("Original Test System");
00036     system1->setValue(20.0);
00037
00038     // Making assertion to verify if the name property was copied before the original's alteration.
00039     assert(system2->getName() == "Test System");
00040     // Making assertion to verify if the value property was copied before the original's alteration.
00041     assert(system2->getValue() == 10.0);
00042     // Making assertion to verify if the System objects aren't pointing to the same memory.
00043     assert(system1 != system2);
00044
00045     cout « GREEN « "OK!" « RESET « endl;
00046 }
00047
00048 // Function for the System class' destructor unit test.
00049 void unit_system_destructor(){
00050     cout « "TEST 4 - Default destructor of the System class" « endl;
00051
00052     double vmBefore, vmAfter, rss;
00053
00054     // Getting the memory usage previous to the creation of a system.
00055     memory_usage(vmBefore, rss);
00056
00057     SystemInterface* system = new System("Test System", 10.0);
00058     delete system;
00059
00060     // Getting the memory usage after the creation and destruction of a System object.
00061     memory_usage(vmAfter, rss);
00062
00063     // Making assertion to verify if the memory usage after the creation and deletion
00064     // is the same as before the creation of System object.
00065     assert(vmBefore == vmAfter);
00066
00067     cout « GREEN « "OK!" « RESET « endl;
00068 }
00069
00070 // Function for System class' method getName() unit test.
00071 void unit_system_getName(){
00072     cout « "TEST 5 - System class's getName() method" « endl;
00073
00074     SystemInterface* system = new System("Test System", 10.0);
00075
00076     // Making assertion to verify if the method returns the System class name and if it's
00077     // equal to the parameter previously passed.
00078     assert(system->getName() == "Test System");
00079
00080     cout « GREEN « "OK!" « RESET « endl;
00081 }
00082
```

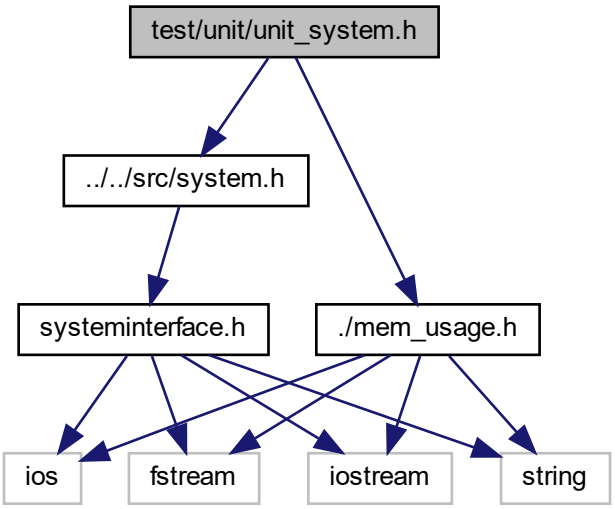
```
00083 // Function for System class' method setName() unit test.
00084 void unit_system_setName(){
00085     cout « "TEST 6 - System class's setName() method" « endl;
00086 
00087     SystemInterface* system = new System("Test System", 10.0);
00088     system->setName("Altered Name");
00089 
00090     // Making assertion to verify if the data of the name property has been altered.
00091     assert(system->getName() == "Altered Name");
00092 
00093     cout « GREEN « "OK!" « RESET « endl;
00094 }
00095 
00096 // Function for System class' method getValue() unit test.
00097 void unit_system_getValue(){
00098     cout « "TEST 7 - System class's getValue() method" « endl;
00099 
00100     SystemInterface* system = new System("Test System", 10.0);
00101 
00102     // Making assertion to verify if the method returns the System class value and if it's
00103     // equal to the parameter previously passed.
00104     assert(system->getValue() == 10.0);
00105 
00106     cout « GREEN « "OK!" « RESET « endl;
00107 }
00108 
00109 // Function for System class' method setValue() unit test.
00110 void unit_system_setValue(){
00111     cout « "TEST 8 - System class's setValue() method" « endl;
00112 
00113     SystemInterface* system = new System("Test System", 10.0);
00114     system->setValue(20.0);
00115 
00116     // Making assertion to verify if the data of the value property has been altered.
00117     assert(system->getValue() == 20.0);
00118 
00119     cout « GREEN « "OK!" « RESET « endl;
00120 }
00121 
00122 // Function for System class' assingment operator unit test.
00123 void UnitSystem::unit_system_assingmentOperator(){
00124     cout « "TEST 9 - System class assignment operator" « endl;
00125 
00126     System* system1 = new System("Test System", 10.0);
00127     System* system2 = new System();
00128     *system2 = *system1;
00129 
00130     system1->setName("Original Test System");
00131     system1->setValue(100.0);
00132 
00133     // Making assertion to verify if the name property was assigned.
00134     assert(system2->getName() == "Test System");
00135     // Making assertion to verify if the value property was assigned.
00136     assert(system2->getValue() == 10.0);
00137 
00138     cout « GREEN « "OK!" « RESET « endl;
00139 }
00140 
00141 // Function for System class' "+" operator unit test.
00142 void unit_system_sumOperator(){
00143     SystemInterface* sys1 = new System("Population 1", 100.0);
00144     SystemInterface* sys2 = new System("Population 2", 10.0);
00145 
00146     cout « "TEST 10 - System class sum operator" « endl;
00147     assert(((*sys1) + (*sys2)) == 110);
00148     assert(((*sys1) + 20.0) == 120);
00149 
00150     cout « GREEN « "OK!" « RESET « endl;
00151 }
00152 
00153 // Function for System class' "-" operator unit test.
00154 void unit_system_minusOperator(){
00155     SystemInterface* sys1 = new System("Population 1", 100.0);
00156     SystemInterface* sys2 = new System("Population 2", 10.0);
00157 
00158     cout « "TEST 11 - System class subtraction operator" « endl;
00159     assert(((*sys1) - (*sys2)) == 90);
00160     assert(((*sys1) - 20.0) == 80);
00161 
00162     cout « GREEN « "OK!" « RESET « endl;
00163 }
00164 
00165 // Function for System class' "*" operator unit test.
00166 void unit_system_timesOperator(){
00167     SystemInterface* sys1 = new System("Population 1", 100.0);
00168     SystemInterface* sys2 = new System("Population 2", 10.0);
00169 
```

```
00170     cout « "TEST 12 - System class multiplication operator" « endl;
00171     assert(((*sys1) * (*sys2)) == 1000);
00172     assert(((*sys1) * 20.0) == 2000);
00173
00174     cout « GREEN « "OK!" « RESET « endl;
00175 }
00176
00177 // Function for System class' "/" operator unit test.
00178 void unit_system_divisionOperator(){
00179     SystemInterface* sys1 = new System("Population 1", 100.0);
00180     SystemInterface* sys2 = new System("Population 2", 10.0);
00181
00182     cout « "TEST 13 - System class division operator" « endl;
00183     assert(((*sys1) / (*sys2)) == 10);
00184     assert(((*sys1) / 20.0) == 5);
00185
00186     cout « GREEN « "OK!" « RESET « endl;
00187 }
00188
00189 // Function to run all the System class' unit tests.
00190 void run_unit_tests_system(){
00191
00192     UnitSystem* unit_sys = new UnitSystem();
00193
00194     // Calling all the System class' unit test functions.
00195     unit_system_constructor();
00196     unit_sys->unit_system_copy_constructor();
00197     unit_system_destructor();
00198     unit_system_getName();
00199     unit_system_setName();
00200     unit_system_getValue();
00201     unit_system_setValue();
00202     unit_sys->unit_system_assingmentOperator();
00203     unit_system_sumOperator();
00204     unit_system_minusOperator();
00205     unit_system_timesOperator();
00206     unit_system_divisionOperator();
00207
00208     delete(unit_sys);
00209 }
```

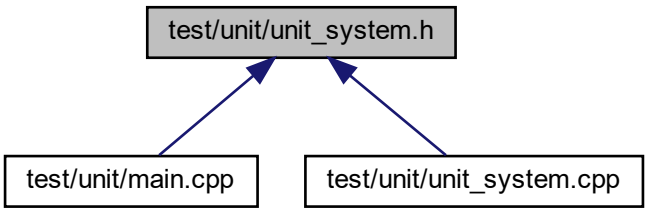
5.41 test/unit/unit_system.h File Reference

```
#include "../../src/system.h"
#include "./mem_usage.h"
```

Include dependency graph for unit_system.h:



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



Classes

- class UnitSystem

  Unit tests.

Macros

- #define RESET "\033[0m" /∗! Escape sequence to reset color output to default. ∗/
- #define GREEN "\033[32m" /∗! Escape sequence to a green color output. ∗/

Functions

- void unit_system_constructor ()
- void unit_system_destructor ()
- void unit_system_getName ()
- void unit_system_setName ()
- void unit_system_getValue ()
- void unit_system_setValue ()
- void unit_system_sumOperator ()
- void unit_system_minusOperator ()
- void unit_system_timesOperator ()
- void unit_system_divisionOperator ()
- void run_unit_tests_system ()

5.41.1 Macro Definition Documentation

5.41.1.1 GREEN

```
#define GREEN "\033[32m" /*!  Escape sequence to a green color output.  */
```

Definition at line 8 of file unit_system.h.

5.41.1.2 RESET

```
#define RESET "\033[0m" /*!  Escape sequence to reset color output to default.  */
```

Definition at line 7 of file unit_system.h.

5.41.2 Function Documentation

5.41.2.1 run_unit_tests_system()

```
void run_unit_tests_system ( )
```

Function prototype for the function that runs all the unit tests of the System class.

Definition at line 190 of file unit_system.cpp.

5.41.2.2 unit_system_constructor()

```
void unit_system_constructor ( )
```

Function prototype for the System class' constructor unit test.

Definition at line 7 of file unit_system.cpp.

5.41.2.3 unit_system_destructor()

```
void unit_system_destructor ( )
```

Function prototype for the System class' destructor unit test.

Definition at line 49 of file unit_system.cpp.

5.41.2.4 unit_system_divisionOperator()

```
void unit_system_divisionOperator ( )
```

Function prototype for the System class' "/" operator unit test.

Definition at line 178 of file unit_system.cpp.

5.41.2.5 unit_system_getName()

```
void unit_system_getName ( )
```

Function prototype for the System class' method getName() unit test.

Definition at line 71 of file unit_system.cpp.

5.41.2.6 unit_system_getValue()

```
void unit_system_getValue ( )
```

Function prototype for the System class' method getValue() unit test.

Definition at line 97 of file unit_system.cpp.

5.41.2.7 unit_system_minusOperator()

```
void unit_system_minusOperator ( )
```

Function prototype for the System class' "-" operator unit test.

Definition at line 154 of file unit_system.cpp.

5.41.2.8 unit_system_setName()

```
void unit_system_setName ( )
```

Function prototype for the System class' method setName() unit test.

Definition at line 84 of file unit_system.cpp.

5.41.2.9 unit_system_setValue()

```
void unit_system_setValue ( )
```

Function prototype for the System class' method setValue() unit test.

Definition at line 110 of file unit_system.cpp.

5.41.2.10 unit_system_sumOperator()

```
void unit_system_sumOperator ( )
```

Function prototype for the System class' "+" operator unit test.

Definition at line 142 of file unit_system.cpp.

5.41.2.11 unit_system_timesOperator()

```
void unit_system_timesOperator ( )
```

Function prototype for the System class' "∗" operator unit test.

Definition at line 166 of file unit_system.cpp.

5.42 unit_system.h

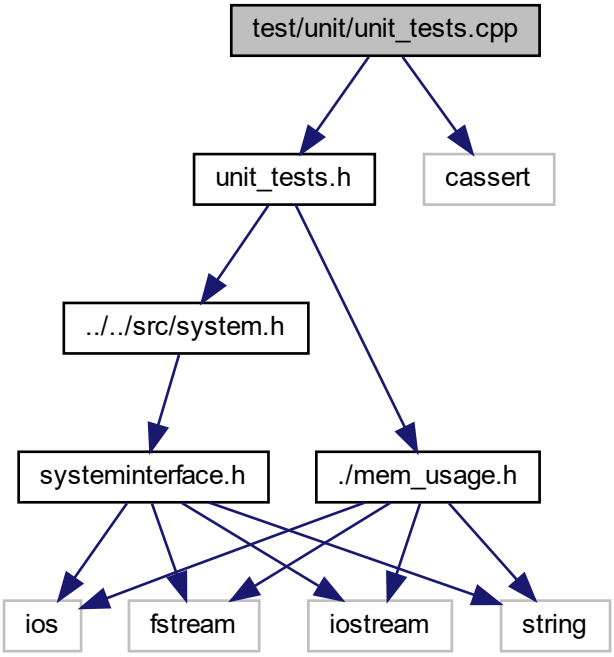
Go to the documentation of this file.

```
00001 #ifndef UNIT_SYSTEM
00002 #define UNIT_SYSTEM
00003
00004 #include "../../src/system.h"
00005 #include "./mem_usage.h"
00006
00007 #define RESET "\033[0m"
00008 #define GREEN "\033[32m"
00011
00014 class UnitSystem{
00015     public:
00016         UnitSystem(){}
00017         ~UnitSystem(){}
00018
00022         void unit_system_copy_constructor();
00023
00027         void unit_system_assingmentOperator();
00028 };
00029
00033 void unit_system_constructor();
00034
00038 void unit_system_destructor();
00039
00043 void unit_system_getName();
00044
00048 void unit_system_setName();
00049
00053 void unit_system_getValue();
00054
00058 void unit_system_setValue();
00059
00063 void unit_system_sumOperator();
00064
00068 void unit_system_minusOperator();
00069
00073 void unit_system_timesOperator();
00074
00078 void unit_system_divisionOperator();
00079
00083 void run_unit_tests_system();
00084
00085 #endif
```

5.43 test/unit/unit_tests.cpp File Reference

```
#include "unit_tests.h"
#include <cassert>
```

Include dependency graph for unit_tests.cpp:



Functions

- void unit_test_global_sumOperator ()
- void unit_test_global_minusOperator ()
- void unit_test_global_timesOperator ()
- void unit_test_global_divisionOperator ()
- void run_unit_tests_globals ()

5.43.1 Function Documentation

5.43.1.1 run_unit_tests_globals()

```
void run_unit_tests_globals ( )
```

Function prototype for the function that runs all the global unit tests.

Definition at line 47 of file unit_tests.cpp.

5.43.1.2 unit_test_global_divisionOperator()

```
void unit_test_global_divisionOperator ( )
```

Function prototype for the System class' "/" global operator unit test.

Definition at line 37 of file unit_tests.cpp.

5.43.1.3 unit_test_global_minusOperator()

```
void unit_test_global_minusOperator ( )
```

Function prototype for the System class' "-" global operator unit test.

Definition at line 17 of file unit_tests.cpp.

5.43.1.4 unit_test_global_sumOperator()

```
void unit_test_global_sumOperator ( )
```

Function prototype for the System class' "+" global operator unit test.

Definition at line 7 of file unit_tests.cpp.

5.43.1.5 unit_test_global_timesOperator()

```
void unit_test_global_timesOperator ( )
```

Function prototype for the System class' "∗" global operator unit test.

Definition at line 27 of file unit_tests.cpp.

5.44 unit_tests.cpp

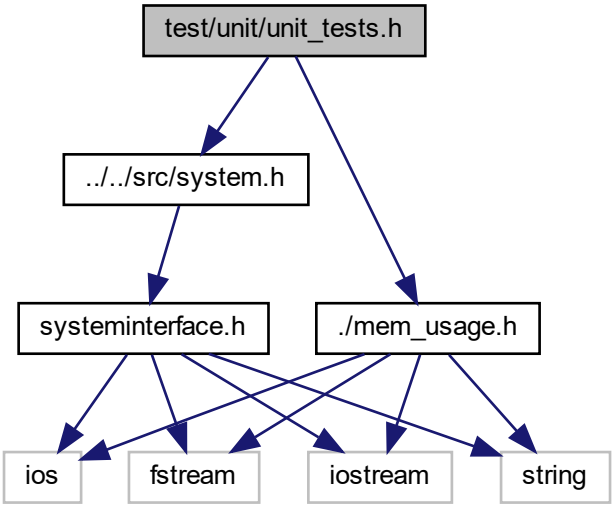
Go to the documentation of this file.

```
00001 #include "unit_tests.h"
00002 #include <cassert>
00003
00004 using namespace std;
00005
00006 // Function for System class' "+" global operator unit test.
00007 void unit_test_global_sumOperator(){
00008     SystemInterface* sys = new System("Population", 10.0);
00009
00010     cout « "TEST 1 - System class sum global operator" « endl;
00011     assert((20.0 + (*sys)) == 30);
00012
00013     cout « GREEN « "OK!" « RESET « endl;
00014 }
00015
00016 // Function for System class' "-" global operator unit test.
00017 void unit_test_global_minusOperator(){
00018     SystemInterface* sys = new System("Population", 10.0);
00019
00020     cout « "TEST 2 - System class subtraction global operator" « endl;
00021     assert((20.0 - (*sys)) == 10);
00022
00023     cout « GREEN « "OK!" « RESET « endl;
00024 }
00025
00026 // Function for System class' "*" global operator unit test.
00027 void unit_test_global_timesOperator(){
00028     SystemInterface* sys = new System("Population", 10.0);
00029
00030     cout « "TEST 3 - System class multiplication global operator" « endl;
00031     assert((20.0 * (*sys)) == 200);
00032
00033     cout « GREEN « "OK!" « RESET « endl;
00034 }
00035
00036 // Function for System class' "/" global operator unit test.
00037 void unit_test_global_divisionOperator(){
00038     SystemInterface* sys = new System("Population", 10.0);
00039
00040     cout « "TEST 4 - System class division global operator" « endl;
00041     assert((20.0 / (*sys)) == 2);
00042
00043     cout « GREEN « "OK!" « RESET « endl;
00044 }
00045
00046 // Function to run all the global unit tests.
00047 void run_unit_tests_globals(){
00048     unit_test_global_sumOperator();
00049     unit_test_global_minusOperator();
00050     unit_test_global_timesOperator();
00051     unit_test_global_divisionOperator();
00052 }
```

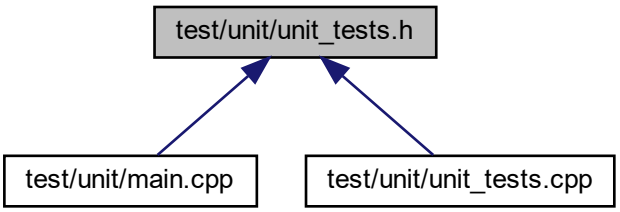
5.45 test/unit/unit_tests.h File Reference

```
#include "../../src/system.h"
#include "./mem_usage.h"
```

Include dependency graph for unit_tests.h:



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



Macros

- #define RESET "\033[0m" /∗! Escape sequence to reset color output to default. ∗/
- #define GREEN "\033[32m" /∗! Escape sequence to a green color output. ∗/

Functions

- void unit_test_global_sumOperator ()
- void unit_test_global_minusOperator ()
- void unit_test_global_timesOperator ()
- void unit_test_global_divisionOperator ()
- void run_unit_tests_globals ()

5.45.1 Macro Definition Documentation

5.45.1.1 GREEN

```
#define GREEN "\033[32m" /*!  Escape sequence to a green color output.  */
```

Definition at line 5 of file unit_tests.h.

5.45.1.2 RESET

```
#define RESET "\033[0m" /*!  Escape sequence to reset color output to default.  */
```

Definition at line 4 of file unit_tests.h.

5.45.2 Function Documentation

5.45.2.1 run_unit_tests_globals()

```
void run_unit_tests_globals ( )
```

Function prototype for the function that runs all the global unit tests.

Definition at line 47 of file unit_tests.cpp.

5.45.2.2 unit_test_global_divisionOperator()

```
void unit_test_global_divisionOperator ( )
```

Function prototype for the System class' "/" global operator unit test.

Definition at line 37 of file unit_tests.cpp.

5.45.2.3 unit_test_global_minusOperator()

```
void unit_test_global_minusOperator ( )
```

Function prototype for the System class' "-" global operator unit test.

Definition at line 17 of file unit_tests.cpp.

5.45.2.4 unit_test_global_sumOperator()

```
void unit_test_global_sumOperator ( )
```

Function prototype for the System class' "+" global operator unit test.

Definition at line 7 of file unit_tests.cpp.

5.45.2.5 unit_test_global_timesOperator()

```
void unit_test_global_timesOperator ( )
```

Function prototype for the System class' "∗" global operator unit test.

Definition at line 27 of file unit_tests.cpp.

5.46 unit_tests.h

Go to the documentation of this file.

```
00001 #include "../../src/system.h"
00002 #include "./mem_usage.h"
00003
00004 #define RESET "\033[0m"
00005 #define GREEN "\033[32m"
00010 void unit_test_global_sumOperator();
00011
00015 void unit_test_global_minusOperator();
00016
00020 void unit_test_global_timesOperator();
00021
00025 void unit_test_global_divisionOperator();
00026
00030 void run_unit_tests_globals();
```

Index