libpense.kdevelop Reference Manual

0.1

Generated by Doxygen 1.4.2

Wed Jun 15 14:55:01 2005

Contents

1 libpense.kdevelop Directory Hierarchy

1.1 libpense.kdevelop Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

2 libpense.kdevelop Hierarchical Index

2.1 libpense.kdevelop Class Hierarchy

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

3 libpense.kdevelop Class Index

3.1 libpense.kdevelop Class List

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

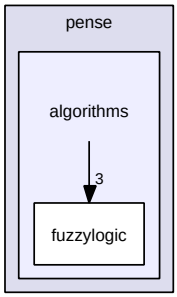
4 libpense.kdevelop File Index

4.1 libpense.kdevelop File List

Here is a list of all files with brief descriptions:

5 libpense.kdevelop Directory Documentation

5.1 /home/engin/kdev/libpense/pense/algorithms/ Directory Reference



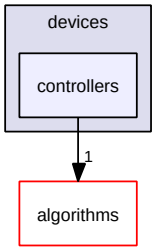
Directories

- directory **fuzzylogic**

Files

- file **algorithm.cpp**
- file **algorithm.h**
- file **fuzzylogic.cpp**
- file **fuzzylogic.h**
- file **polynomial.cpp**
- file **polynomial.h**

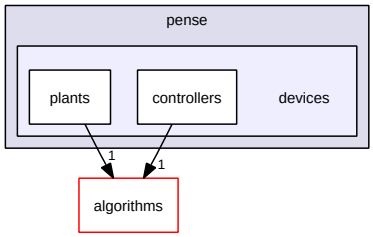
5.2 /home/engin/kdev/libpense/pense/devices/controllers/ Directory Reference



Files

- file **pwm.cpp**
- file **pwm.h**

5.3 /home/engin/kdev/libpense/pense/devices/ Directory Reference



Directories

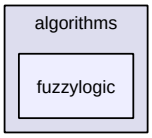
- directory **controllers**
- directory **plants**

Files

- file **device.cpp**
- file **device.h**
- file **inputnode.cpp**

- file **inputnode.h**
- file **node.cpp**
- file **node.h**
- file **outputnode.cpp**
- file **outputnode.h**

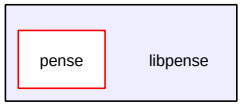
5.4 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/ Directory Reference



Files

- file **fobject.cpp**
- file **fobject.h**
- file **fuzzyvalue.cpp**
- file **fuzzyvalue.h**
- file **inputtriangleset.cpp**
- file **inputtriangleset.h**
- file **outputtriangleset.cpp**
- file **outputtriangleset.h**
- file **triangle.cpp**
- file **triangle.h**
- file **triangleset.cpp**
- file **triangleset.h**

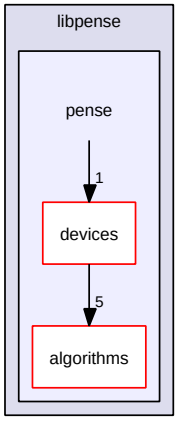
5.5 /home/engin/kdev/libpense/ Directory Reference



Directories

- directory **pense**

5.6 /home/engin/kdev/libpense/pense/ Directory Reference



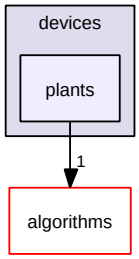
Directories

- directory **algorithms**
- directory **devices**

Files

- file **environment.cpp**
- file **environment.h**
- file **global.h**
- file **object.cpp**
- file **object.h**

5.7 /home/engin/kdev/libpense/pense/devices/plants/ Directory Reference



Files

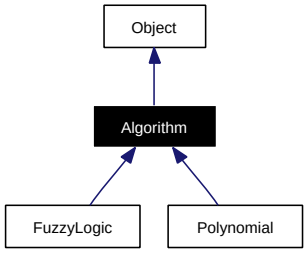
- file **dc_motor.cpp**
- file **dc_motor.h**

6 libpense.kdevelop Class Documentation

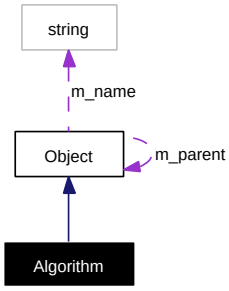
6.1 Algorithm Class Reference

`#include <algorithm.h>`

Inheritance diagram for Algorithm:



Collaboration diagram for Algorithm:



Public Member Functions

- **Algorithm** (const std::string &name="Default **Algorithm**", Object *p=NULL)
- ~**Algorithm** ()
- **ConstParameterList parameters** () const
- bool **contains** (std::string var)
- bool **assign** (const std::string &p, double val)
- double & **operator[]** (const std::string &p)
- double **value** (const std::string &p) const
- virtual double **evaluate** ()=0
- int **parameterCount** () const
- virtual std::string **toString** () const

Protected Member Functions

- bool **addParameter** (std::string p)
- bool **remParameter** (std::string p)

6.1.1 Detailed Description

All algorithms should be derived from this class.

For instance, polynomials, fuzzy logic algorithms.

See also:
Polynomail for an example.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Algorithm::Algorithm (const std::string & *name* = "Default Algorithm", Object * *p* = NULL)

Algorithm c-tor

Parameters:
name name of the algorithm object

p parent of this algorithm object, usually a **Device**(p. 12)

6.1.2.2 Algorithm::~Algorithm ()

Destroys the matheval object and frees resources

6.1.3 Member Function Documentation

6.1.3.1 bool Algorithm::addParameter (std::string *p*) [protected]

Adds a parameter to the algorithm's parameter list.

Parameters:
p parameter name to be added

6.1.3.2 bool Algorithm::assign (const std::string & *p*, double *val*)

Parameters:
p parameter name to assign value to

val value to assign to the variable

Returns:
true on success, false if there is no such variable

6.1.3.3 bool Algorithm::contains (std::string *var*)

Parameters:
var variable name to look up

Returns:
true if the variable exists, false otherwise

6.1.3.4 virtual double Algorithm::evaluate () [pure virtual]

Returns:
the result of the equation with the current parameter values

Implemented in **FuzzyLogic** (p. 24), and **Polynomial** (p. 46).

6.1.3.5 double & Algorithm::operator[] (const std::string & *p*)

6.1.3.6 int Algorithm::parameterCount () const

Returns:
how many parameters in the algorithm

6.1.3.7 ConstParameterList Algorithm::parameters () const

Returns:
list of parameters of this eqation

6.1.3.8 bool Algorithm::remParameter (std::string *p*) [protected]

Removes a parameter from the parameters list

Parameters:
p parameter name to be removed

Returns:
true on success, false if there is no such parameter

6.1.3.9 std::string Algorithm::toString () const [virtual]

Returns:
debug information

Reimplemented from **Object** (p. 40).

Reimplemented in **Polynomial** (p. 47).

6.1.3.10 double Algorithm::value (const std::string & *p*) const

Returns the current value of any parameter.

Note that this method doesn't do error checking.

Parameters:
p parameters name

Returns:
value of the parameter

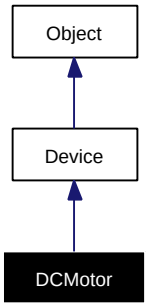
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/**algorithm.h**
- /home/engin/kdev/libpense/pense/algorithms/**algorithm.cpp**

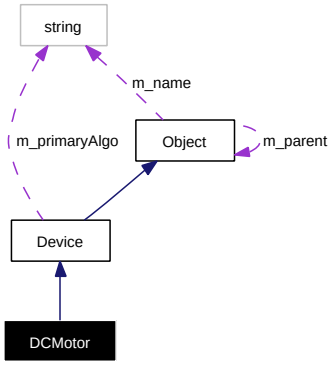
6.2 DCMotor Class Reference

`#include <dc_motor.h>`

Inheritance diagram for DCMotor:



Collaboration diagram for DCMotor:



Public Member Functions

- **DCMotor** (const std::string &n="default motor", Environment *parent=0L)
- **~DCMotor** ()
- void **setLoad** (const std::string &e)
- void **process** ()

6.2.1 Detailed Description

This class represents a DC electric motor's mathematical modelling.

Basically, in each iteration the device is accelerated by the following equation:

$$\frac{\partial \omega}{\partial t} = \frac{T_s}{J}(1 - \frac{\omega}{\omega_f})$$

Where T_s is stall torque defined as follows:

$$T_s = \frac{k_t V}{R}$$

and ω_f is final angular velocity of the motor, also defined as:

$$\omega_f = \frac{T_s}{k_e}$$

In above equations;

J refers to moment of inertia of the load which is usually rotor and the load itself

k_t is torque constant

k_e is voltage constant

Also note that motor takes **Environment**(p. 18) frequency into account.

NOTE:This motor model should be improved.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 DCMotor::DCMotor (const std::string & *n* = "default motor", Environment * *parent* = 0L)

Parameters:

n name of the motor

parent parent of this motor

6.2.2.2 DCMotor::~DCMotor ()

Frees resources

6.2.3 Member Function Documentation

6.2.3.1 void DCMotor::process () [virtual]

Iterate motor simulation

Reimplemented from **Device** (p. 16).

6.2.3.2 void DCMotor::setLoad (const std::string & *e*)

Set loads moment of inertia

Parameters:

e moment of inertia

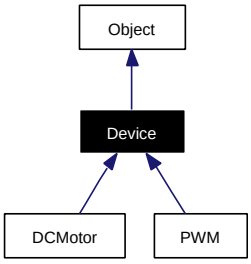
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/devices/plants/**dc_motor.h**
- /home/engin/kdev/libpense/pense/devices/plants/**dc_motor.cpp**

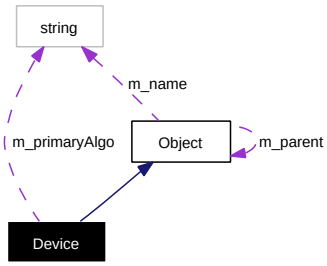
6.3 Device Class Reference

`#include <device.h>`

Inheritance diagram for Device:



Collaboration diagram for Device:



Public Member Functions

- **Device** (const std::string &name="Default **Device**", Environment *parent=0L)
- ~**Device** ()
- bool **addAlgorithm** (Algorithm::Algorithm *a, bool primary=false)
- bool **remAlgorithm** (const std::string &n)
- **OutputNode** * **outputNode** (const std::string &a) const
- **InputNodeListRange** **operator[]** (const std::string &p)
- bool **operator==** (double v) const
- bool **operator!=** (double) const
- double **value** () const
- **OutputNode** * **outputNode** (const std::string &a)
- Algorithm::Algorithm * **algorithm** (const std::string &a)
- virtual void **process** ()
- void **setErrorTolerance** (double e)
- double **errorTolerance** () const
- bool **setPrimaryAlgorithm** (const std::string &a)
- const std::string & **primaryAlgorithm** () const
- virtual std::string **toString** () const
- void **printInfo** () const

Protected Member Functions

- void **addInputNode** (**InputNode** *n)
- void **remInputNode** (const InputNodeList::iterator &it)
- void **addOutputNode** (**OutputNode** *n, bool t=false)
- void **remOutputNode** (const std::string &a)

6.3.1 Detailed Description

This is the base class of all devices in PENSE, including but not limited to plants, sources and controllers.

Note that parameter f is reserved in all plants, that represents the frequency of the system.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Device::Device (const std::string & *name* = "Default Device", Environment * *parent* = 0L)

Creates an device named name

Parameters:

name of the device

parent of the device

6.3.2.2 Device::~Device ()

Frees resources

6.3.3 Member Function Documentation

6.3.3.1 bool Device::addAlgorithm (Algorithm::Algorithm * *a*, bool *primary* = false)

Adds an algorithm to the device.

Algorithm(p. 8) names should be unique, if two identical named algorithms are provided, this method will return false on the second one. WARNING: Note that frequency f is reserved, it represents frequency. So if you're using f in your algorithms make sure that it is meant to be used as frequency.

Parameters:

a algorithm to be added

primary if true algorithm a will be also marked as primary

Returns:

true on success, false on failure

6.3.3.2 void Device::addInputNode (InputNode * *n*) [protected]

Adds node n to the input node list

Parameters:

n node to be added to the input node list

6.3.3.3 void Device::addOutputNode (OutputNode * *n*, bool *t* = false) [protected]

Adds node n to the the output node list

Parameters:
n node to be added to the output node list

t if true this is primary output node

6.3.3.4 Algorithm::Algorithm * Device::algorithm (const std::string & *a*)

Parameters:
a algorithm's name to get

Returns:
Algorithm::Algorithm(p. 9) pointer

6.3.3.5 double Device::errorTolerance () const

Returns:
the current error tolerance value

6.3.3.6 bool Device::operator!= (double) const

Returns:
!operator==(v)

6.3.3.7 bool Device::operator== (double *v*) const

This operator can be used to to see if the device's primary algorithm is around a specific value. The error tolerance value has dramatic effect on this method. Here is a little demonstration.

```
// Create a motor with error tolerance 10.0
Device::Device motor( "my motor", 10.0 );
motor.addAlgorithm( new Algorithm::Polynomial( "x*1000", "algo" ) );
motor["x"] = 10.005;
// This will return true
cout << ( motor == 10000 ? "true" : "false" ) << endl;
// Because the motors actual value is 10005 at the moment, and the
// error tolerance is 10, so any value between 9995 and 10015
// evaluates to true.
```

Parameters:
v value to be evaluate

Returns:
true if the value v is close enough to the primary algorithm's output node's value.

See also:
setErrorTolerance(double)(p. 17)
Device(const std::string&, double)

6.3.3.8 InputNodeListRange Device::operator[] (const std::string & *p*)

This is a convenience method to make it easy to assign a value to parameter p of all algorithms in this device.

```
Device foo;
foo.addAlgorithm( new Polynomial( "x*y*z" ) );
foo.addAlgorithm( new Polynomial( "x*k*l" ) );
foo["x"] = 4;
```

The above code makes both algorithms' parameter x = 4.

Parameters:
p

Returns:

6.3.3.9 OutputNode * Device::outputNode (const std::string & *a*)

Parameters:
a algorithm's name which this node belongs to

Returns:
the appropriate **OutputNode**(p. 40) object pointer

6.3.3.10 OutputNode* Device::outputNode (const std::string & *a*) const

Parameters:
a algorithm name i.e. "sin(x)" or "fuzzyMotorController"

Returns:
the output node which represents the algorithm a

6.3.3.11 const std::string & Device::primaryAlgorithm () const

Returns:
name of the current primary algorithm

6.3.3.12 void Device::printInfo () const

This prints verbose debug information to stdout

6.3.3.13 void Device::process () [virtual]

Process the device one step, which means; evaluating all the algorithms this device has, so the output nodes will be updated with new generated data. The other devices connected to this device should also be explicitly processed.

Reimplemented in **PWM** (p. 49), and **DCMotor** (p. 12).

6.3.3.14 bool Device::remAlgorithm (const std::string & *n*)

Removes an algorithm

Parameters:
n name of the algorithm

Returns:
true on success, false on failure

6.3.3.15 void Device::remInputNode (const InputNodeList::iterator & *it*) [protected]

Removes input node which is represented by InputNodeList::iterator it

Parameters:
it iterator pointing to the input node to be removed

6.3.3.16 void Device::remOutputNode (const std::string & *a*) [protected]

Removes output node which represents algorithm a

Parameters:
a algorithm's name

6.3.3.17 void Device::setErrorTolerance (double *e*)

The error tolerance value is used in == operator of this class.

For instance, assume the current value of the primary output node is 2500.0001, and if our target value is 2500.0, computer will think these values are not equal (and he's right). To overcome such situation we tolerate error. For this particular case, let us use default error tolerance 0.01

When error tolerance is 0.01 any number between 2500.01 and 2499.09 will be interpreted as equal to 2500.

Parameters:
e error tolerance

See also:
==(double)

6.3.3.18 bool Device::setPrimaryAlgorithm (const std::string & *a*)

Sets the primaty algorithm.

This sets the algorithm to be used when operator == called on this class.

Parameters:
a

Returns:
true on success, false on failure

6.3.3.19 std::string Device::toString () const [virtual]

This is supposed to be used with output stream, std::cout, it prints an information defining this object.

```
Device::Device foo( "foo" );
cout << foo << endl;
```

Returns:

Reimplemented from **Object** (p. 40).

6.3.3.20 double Device::value () const

Returns:
the value of the primary algorithm

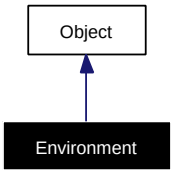
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/devices/**device.h**
- /home/engin/kdev/libpense/pense/devices/**device.cpp**

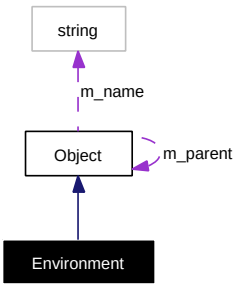
6.4 Environment Class Reference

`#include <environment.h>`

Inheritance diagram for Environment:



Collaboration diagram for Environment:



Public Member Functions

- **Environment** (uint freq=1000000, const std::string &name="default system", Object *parent=0L)
- ~**Environment** ()
- void **addDevice** (Device::Device *d)
- bool **remDevice** (const std::string &n)
- Device::Device * **device** (const std::string &n)
- bool **contains** (const std::string &n)
- int **order** (const std::string &n)
- void **swap** (int x, int y)
- void **printInfo** () const
- virtual std::string **toString** () const
- void **process** ()
- Device::Device & **operator[]** (const std::string &n)
- void **setFrequency** (uint f)
- uint **frequency** () const

6.4.1 Detailed Description

Environment is the largest container in the PENSE framework. It can host virtually unlimited devices in it. It provides shared parameters for all devices in it, those are including but not limited to frequency of the iteration, environment temperature, humudity etc.

Additionally through the Environment object you can set a value to a parameter in all the devices in this environment. Assume it contains 3 devices. Calling 'env->assign("foo", "bar");' will set parameters foos' values to "bar" on all 3 devices.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Environment::Environment (uint *freq* = 1000000, const std::string & *name* = "default system", Object * *parent* = 0L)

Here the most important parameter is frequency parameter. It's where you define

Parameters:

freq frequency of the environment

name name of the environment

parent parent of the environment

Returns:

6.4.2.2 Environment::~Environment ()

6.4.3 Member Function Documentation

6.4.3.1 void Environment::addDevice (Device::Device * *d*)

Parameters:

d **Device**(p. 12) to add

6.4.3.2 bool Environment::contains (const std::string & *n*)

Checks if a device exists in the environment or not. Looks up by name, since devices in an Environment should have unique names.

Parameters:
n name of the device to loop up

Returns:
true if it exists

6.4.3.3 Device::Device * Environment::device (const std::string & *n*)

Returns pointer to a specific device according to it's name.

Parameters:
n name of the requested device

Returns:
pointer to the requested device

6.4.3.4 uint Environment::frequency () const

Returns:
the current frequency of the system

6.4.3.5 Device::Device & Environment::operator[] (const std::string & *n*)

This is a utulity method to do easy operations on the devices.

```
Environment env;
Device plant( "my plant", &env );
...
// Sets parameter V of device "my plant" in the environment to 23.5.
env["my plant"]["V"] = 23.5;
```

Parameters:
n name of the device

Returns:
reference to the device

6.4.3.6 int Environment::order (const std::string & *n*)

Returns the order of the given device.

Parameters:
n name of the device

Returns:
order of the device

6.4.3.7 void Environment::printInfo () const

Just for debugging purposes.

6.4.3.8 void Environment::process ()

Processes the devices in the environment in given order.

6.4.3.9 bool Environment::remDevice (const std::string & *n*)

Parameters:
n device name to be deleted

Returns:
true on success

6.4.3.10 void Environment::setFrequency (uint *f*)

This is where you define how many iterations you're going to do in one second, hence the frequency of the system. This parameter will will be provided to all devices in the Environment by parameter *f*.

Parameters:
f frequency of the system

6.4.3.11 void Environment::swap (int *x*, int *y*)

Swaps the given devices' orders.

Parameters:
x device to be swapped
y device to be swapped

6.4.3.12 std::string Environment::toString () const [virtual]

Returns:
debug information

Reimplemented from **Object** (p. 40).

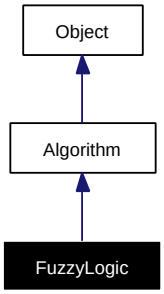
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/**environment.h**
- /home/engin/kdev/libpense/pense/**environment.cpp**

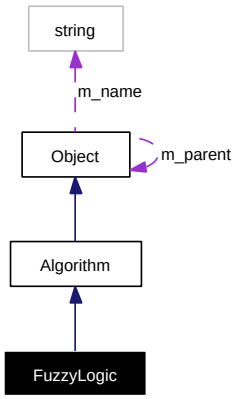
6.5 FuzzyLogic Class Reference

`#include <fuzzylogic.h>`

Inheritance diagram for FuzzyLogic:



Collaboration diagram for FuzzyLogic:



Public Member Functions

- **FuzzyLogic** (const std::string &name="default fuzzy logic algorithm", Object *parent=0L)
- ~**FuzzyLogic** ()
- bool **addInputTriangle** (double start, double end, const std::string &name)
- bool **remInputTriangle** (const std::string &name)
- bool **addOutputTriangle** (double start, double end, const std::string &name, const std::string &partner)
- bool **remOutputTriangle** (const std::string &name)
- double **evaluate** ()
- double **evaluate** (double val)

6.5.1 Detailed Description

This class makes use of the FuzzyLogic framework which is in $PENSE_-SRC/pense/algorithms/fuzzylogic

One can say that this is a wrapper class which derives from **Algorithm**(p. 8) to form a valid PENSE algorithm.

This **Algorithm**(p. 8) has a fixed parameter "input", and that's the only input which is being evaluated. So, what programmer should do is to create a fuzzy logic object, add input triangles, add output triangles, assign an input value and evaluate it. Alternatively you can assign a value within **evaluate(double)**(p. 24) too.

```
Algorithm::FuzzyLogic f;
f.addInputTriangle( 10.0, 20.0, "too slow" );
...
f["input"] = 15.0;
f.evaluate();
```

This **Algorithm**(p. 8) can be used for fuzzy logic controllers.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 FuzzyLogic::FuzzyLogic (const std::string & *name* = "default fuzzy logic algorithm", Object * *parent* = 0L)

Creates a fuzzy logic algorithm.

Parameters:

name name of this algorithm

parent parent of this algorithm

6.5.2.2 FuzzyLogic::~FuzzyLogic ()

Frees resources

6.5.3 Member Function Documentation

6.5.3.1 bool FuzzyLogic::addInputTriangle (double *start*, double *end*, const std::string & *name*)

Adds an input triangle.

You can define a triangle by it's start point, end point and a name. You can think of these triangles as the triangles on the vertical axis.

Parameters:

start start point of the triangle

end end point of the triangle

name name of the triangle

Returns:

true on success

6.5.3.2 bool FuzzyLogic::addOutputTriangle (double *start*, double *end*, const std::string & *name*, const std::string & *partner*)

Adds an output triangle.

You can think of ouput triangles as the triangles on the horizontal axis.

Parameters:
start start point of the triangle
end end point of the triangle
name name of the triangle
partner name of the input triangle which this output triangle will be partner with.

Returns:
true on success

6.5.3.3 double FuzzyLogic::evaluate (double *val*)

This updates the "input" parameters value and calls **evaluate()**(p. 24).

See also:
evaluate()(p. 24)

Parameters:
val value to be evaluated

Returns:
the crisp output value

6.5.3.4 double FuzzyLogic::evaluate () [virtual]

This fuzifies the input internally according to input triangles. Then defuzifies this fuzified value according to the output triangles and gives the crisp output.

The input value in this case is value of parameter "input".

Returns:
the crisp output value

Implements **Algorithm** (p. 10).

6.5.3.5 bool FuzzyLogic::remInputTriangle (const std::string & *name*)

Parameters:
name of the triangle to be removed

Returns:
true on sucecss

6.5.3.6 bool FuzzyLogic::remOutputTriangle (const std::string & *name*)

Parameters:
name output triangles name to be removed

Returns:
true on success

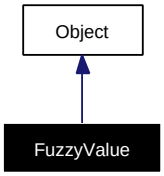
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/**fuzzylogic.h**
- /home/engin/kdev/libpense/pense/algorithms/**fuzzylogic.cpp**

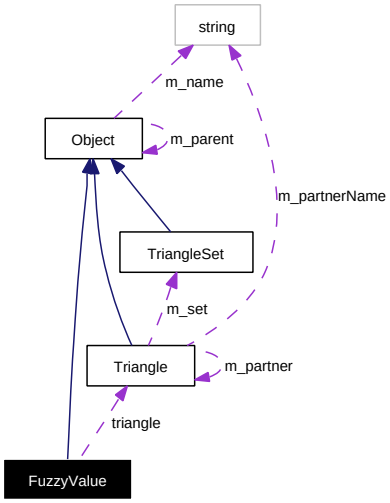
6.6 FuzzyValue Class Reference

`#include <fuzzyvalue.h>`

Inheritance diagram for FuzzyValue:



Collaboration diagram for FuzzyValue:



Public Member Functions

- **FuzzyValue** (double **degree**, const **Triangle** ∗**triangle**, std::string name)
- ∼**FuzzyValue** ()
- double **getDegree** (void) const
- const **Triangle** ∗ **getTriangle** (void) const

Protected Attributes

- double **degree**
- const **Triangle** ∗ **triangle**

6.6.1 Detailed Description

This class defines a fuzzy value which consists of a double value which is degree of membership and a triangle which this value belongs to.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 FuzzyValue::FuzzyValue (double *d*, const Triangle * *t*, std::string *n*)

This is the object that holds a fuzified value. A fuzzified value consists of two information; degree of membership, the triangle which the membership belongs.

Parameters:

d degree of membership

t triangle which the fuzzified value belongs to

n object name

6.6.2.2 FuzzyValue::~FuzzyValue ()

Destructor

6.6.3 Member Function Documentation

6.6.3.1 double FuzzyValue::getDegree (void) const [inline]

Returns:

membership degree

6.6.3.2 const Triangle* FuzzyValue::getTriangle (void) const [inline]

Returns:

triangle which this fuzzy value belongs to

6.6.4 Member Data Documentation

6.6.4.1 double FuzzyValue::degree [protected]

degree of the membership

6.6.4.2 const Triangle* FuzzyValue::triangle [protected]

triangle which this fuzzy value belongs to

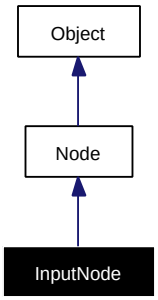
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**fuzzyvalue.h**
- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**fuzzyvalue.cpp**

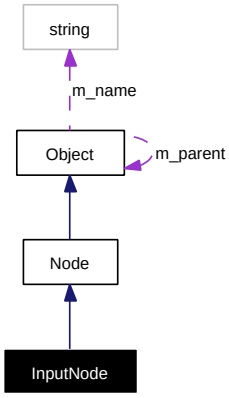
6.7 InputNode Class Reference

`#include <inputnode.h>`

Inheritance diagram for InputNode:



Collaboration diagram for InputNode:



Public Member Functions

- **InputNode** (const std::string &name="Default Input **Node**", Algorithm::Algorithm *parent=0L)
- ~**InputNode** ()
- bool **connect** (**OutputNode** *n)
- bool **disconnect** (**OutputNode** *n)
- bool **connect** (**Node** *n, bool recursive=true)
- void **emit** (double val)
- void **omit** (double val)
- double **operator=** (double v)
- double **operator+=** (double v)
- double **value** () const
- virtual std::string **toString** () const

6.7.1 Detailed Description

Input node represents a parameter of an algorithm.

Assume you have added an Algorithm::Polynomial x*(y+z) to your **Device**(p. 12). Your **Device**(p. 12) will have three **input nodes** (p. 26); x, y,z. You can connect other devices' **output nodes** (p. 40) to this nodes, so that they will update parameters on update.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 InputNode::InputNode (const std::string & *name* = "Default Input Node", Algorithm::Algorithm * *parent* = 0L)

Parameters:

name name of the parameter this node is preresenting.

parent the algorithm which this input node belongs to

Returns:

6.7.2.2 InputNode::~InputNode ()

Disconnects itself from connected nodes.

6.7.3 Member Function Documentation

6.7.3.1 bool InputNode::connect (Node * *n*, bool *recursive* = true) [virtual]

Reimplemented connect method to make sure if the parameter n really represents an **OutputNode**(p. 40), by runtime type identification with dynamic_cast.

Parameters:

n node to be connected to

recursive if true node n also connects to this node

Returns:

true on success

Reimplemented from **Node** (p. 35).

6.7.3.2 bool InputNode::connect (OutputNode * *n*)

Parameters:

n **OutputNode**(p. 40) to be connected to

Returns:

true on success

6.7.3.3 bool InputNode::disconnect (OutputNode * *n*)

Parameters:

n **OutputNode**(p. 40) to be disconnected from

Returns:

true on success

6.7.3.4 void InputNode::emit (double *val*) [virtual]

An InputNode doesn't emit signal, if it tries to, an exception will will be raised.

Parameters:
val value to emit

Reimplemented from **Node** (p. 36).

6.7.3.5 void InputNode::omit (double *val*) [virtual]

When this method is called the input node updates the parameter value of it's parent algorithm. For instance, assume this node represents parameter x of it's parent algorithm, then when val is emitted to this signal, it'll omit it, and assign val to x on it's parent algorithm.

Parameters:
val

Implements **Node** (p. 36).

6.7.3.6 double InputNode::operator+= (double *v*)

This is used to increment the input node's value by v.

Parameters:
v increment value

Returns:
returns the updated value of this node for convenience

6.7.3.7 double InputNode::operator= (double *v*)

Assign a value to this input node, it's exactly the same as **omit(double)**(p. 29).

Parameters:
v value to assign to this node

Returns:
returns the value it self for convenience

6.7.3.8 std::string InputNode::toString () const [virtual]

Returns:
debug information

Reimplemented from **Node** (p. 36).

6.7.3.9 double InputNode::value () const

Returns:
current value of the node

Reimplemented from **Node** (p. 36).

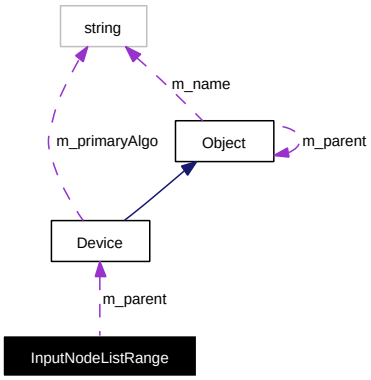
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/devices/**inputnode.h**
- /home/engin/kdev/libpense/pense/devices/**inputnode.cpp**

6.8 InputNodeListRange Class Reference

`#include <device.h>`

Collaboration diagram for InputNodeListRange:



Public Member Functions

- **InputNodeListRange** (std::pair< InputNodeList::iterator, InputNodeList::iterator > range, const **Device** *parent)
- **InputNodeListRange** & **operator**= (double v)
- **InputNodeListRange** & **operator**+= (double v)
- void **connect** (**OutputNode** *n)
- void **disconnect** (**OutputNode** *n)
- InputNodeList::iterator **begin** () const
- double **value** () const
- InputNodeList::iterator **end** () const

6.8.1 Detailed Description

This is a utulity proxy class which is not meant to be used stand-alone.

This class essentially transports all the operations from it's interface to the input nodes it contains.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 InputNodeListRange::InputNodeListRange (std::pair< InputNodeList::iterator, InputNodeList::iterator > *range*, const Device * *parent*)

Parameters:

range this parameter is supposed to be come from std::multimap::equal_range, to be more accurate from m_iNodes

parent parent of this InputNodeListRange

6.8.3 Member Function Documentation

6.8.3.1 InputNodeList::iterator InputNodeListRange::begin () const

Returns:

begin iterator

6.8.3.2 void InputNodeListRange::connect (OutputNode * *n*)

Connects all input nodes this class is representing to output node n

Parameters:

n node to be connected

6.8.3.3 void InputNodeListRange::disconnect (OutputNode * *n*)

Disconnects all input nodes from the given output node n

Parameters:

n output node to be disconnected from

6.8.3.4 InputNodeList::iterator InputNodeListRange::end () const

Returns:

end iterator

6.8.3.5 InputNodeListRange & InputNodeListRange::operator+= (double *v*)

It's almost the same as operatar==. It calls operator+= on all input noes with parameter v

Parameters:

v

Returns:

the updated input node list range

6.8.3.6 InputNodeListRange & InputNodeListRange::operator= (double *v*)

This operator assign value v to all input nodes it contains, which is defined by std::multimap::equal_range.

In essence, what this method do is to call operator== on all input nodes it contains with parameter v.

Parameters:
v the value to be assigned to all input nodes of this range

Returns:
the updated input node list range.

6.8.3.7 double InputNodeListRange::value () const

Returns:
current value of the node list

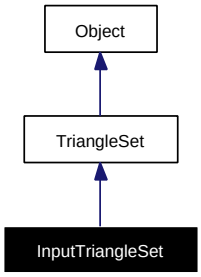
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/devices/**device.h**
- /home/engin/kdev/libpense/pense/devices/**device.cpp**

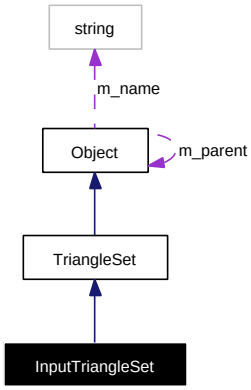
6.9 InputTriangleSet Class Reference

`#include <inputtriangleset.h>`

Inheritance diagram for InputTriangleSet:



Collaboration diagram for InputTriangleSet:



Public Member Functions

- **InputTriangleSet** (const std::string &name)
- **~InputTriangleSet** ()
- **FuzzyValueList fuzify** (double input)

6.9.1 Detailed Description

Fuzzy Input Set, which includes fuzzy triangles for inputs. This class can fuzify the input.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 InputTriangleSet::InputTriangleSet (const std::string & *name*)

Creates input triang set

Parameters:
name name of the set

Returns:

6.9.2.2 InputTriangleSet::~InputTriangleSet ()

Frees resources

6.9.3 Member Function Documentation

6.9.3.1 FuzzyValueList InputTriangleSet::fuzify (double *input*)

Fuzifies the measured input data.

Parameters:
input measured input value

Returns:
fuzzy value list

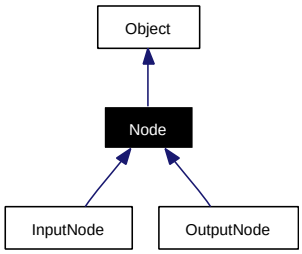
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**inputtriangleset.h**
- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**inputtriangleset.cpp**

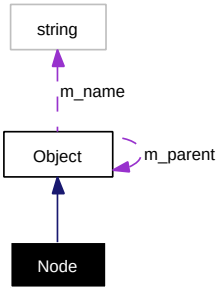
6.10 Node Class Reference

`#include <node.h>`

Inheritance diagram for Node:



Collaboration diagram for Node:



Public Member Functions

- **Node** (const std::string &name, **Object** *parent)
- ~**Node** ()
- virtual bool **connect** (**Node** *n, bool recursive=true)
- virtual bool **disconnect** (**Node** *n, bool recursive=true)
- virtual void **emit** (double v)
- virtual void **omit** (double v)=0
- double **value** () const
- **ConstNodeList nodes** () const
- virtual std::string **toString** () const

Protected Attributes

- double **m_val**

  Current value of this node, i.e. last omitted value.

6.10.1 Detailed Description

This is the abstract base class of nodes.

Every node should be derived from this class. Node objects are capable of connecting each other and keeping a list of connections.

Nodes disconnects upon destruction.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 Node::Node (const std::string & *name*, Object * *parent*)

Node c-tor

Parameters:

name name of the node

parent parent of this node. This is usually an algorithm.

6.10.2.2 Node::~Node ()

Disconnects itself from all connected nodes.

6.10.3 Member Function Documentation

6.10.3.1 bool Node::connect (Node * *n*, bool *recursive* = `true`) `[virtual]`

Connects this node to the node n provided by the user. When update() is called, this node will call update() of each node whome this node connected to.

Parameters:

n the node to be connected to

recursive if true node n also connects to this node.

Returns:

true on success, false on failure

Reimplemented in **InputNode** (p. 28), and **OutputNode** (p. 41).

6.10.3.2 bool Node::disconnect (Node * *n*, bool *recursive* = `true`) `[virtual]`

Disconnects the FIRST occurence of node n from this node.

No more updates will be received by n from this node nor no more updates will be sent.

Parameters:

n node to be disconnected

recursive should the node being disconnected also disconnet itself from this node

Returns:

true on success, false on failure

6.10.3.3 void Node::emit (double *v*) [virtual]

Emits value v to all connected nodes.

What is internally happening is all nodes **omit(double)**(p. 36) method is being called with value v.

Parameters:
v value to be emitted

Reimplemented in **InputNode** (p. 29).

6.10.3.4 ConstNodeList Node::nodes () const

Returns:
the list of connected nodes.

6.10.3.5 virtual void Node::omit (double *v*) [pure virtual]

Omit emitted signal from other nodes.

Since this is a pure virtual method you should implement this. This is where you define what your node really does.

Parameters:
v received value

Implemented in **InputNode** (p. 29), and **OutputNode** (p. 42).

6.10.3.6 std::string Node::toString () const [virtual]

Returns:
debug information

Reimplemented from **Object** (p. 40).

Reimplemented in **InputNode** (p. 29), and **OutputNode** (p. 42).

6.10.3.7 double Node::value () const

Nodes keep a copy of the value they've transmitted and with this method you can retrive the last value of this node.

Returns:
value of the node

Reimplemented in **InputNode** (p. 30).

6.10.4 Member Data Documentation

6.10.4.1 double Node::m_val [protected]

Current value of this node, i.e. last omitted value.

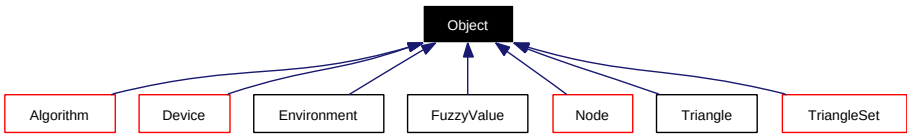
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/devices/**node.h**
- /home/engin/kdev/libpense/pense/devices/**node.cpp**

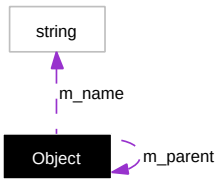
6.11 Object Class Reference

`#include <fobject.h>`

Inheritance diagram for Object:



Collaboration diagram for Object:



Public Member Functions

- **Object** (const char *name)
- **Object** (const std::string &name)
- virtual ~**Object** ()
- void **setName** (const std::string &name)
- const std::string & **name** (void) const
- **Object** (const std::string &name="Default **Object**", Object *p=NULL)
- virtual ~**Object** ()
- void **setName** (std::string name)
- std::string **name** (void) const
- void **setParent** (**Object** *p)
- **Object** * **parent** () const
- virtual bool **operator**== (const std::string &n) const
- virtual bool **operator**== (const **Object** &o) const
- virtual std::string **toString** () const

6.11.1 Detailed Description

Base class for all Fuzzy objects.

This is not PENSE::Object on purpose, this is the Fuzzy Logic framework itself, hence it is a seperate concept. It'll be used in an PENSE::Algorithm as a utulity.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Object::Object (const char * *name*)

Constructs an object with name n

Parameters:
name object name

6.11.2.2 Object::Object (const std::string & *name*)

Constructs an object with name n

Parameters:
name object name

6.11.2.3 Object::~Object () [virtual]

Destructs the object

6.11.2.4 Object::Object (const std::string & *name* = "Default Object", Object * *p* = NULL)

Constructs an object with name n

Parameters:
name object name
p parent of this object

6.11.2.5 virtual Object::~Object () [virtual]

Destructs the object

6.11.3 Member Function Documentation

6.11.3.1 std::string Object::name (void) const

Gets the object's name

Returns:
name of the object

6.11.3.2 std::string Object::name (void) const

Gets the object's name

Returns:
name of the object

6.11.3.3 bool Object::operator== (const Object & *o*) const [virtual]

Convenience method to avoid 'obj1.**name()**(p. 38) == obj2.name()'. Instead just 'obj1 == obj2' is fine.

Parameters:
o object to be compared too

Returns:
true if Object o has the same name with this object

6.11.3.4 bool Object::operator== (const std::string & *n*) const [virtual]

Convenience method to avoid 'obj.**name()**(p. 38) == std::string()'. Instead just 'obj == std::string'

Parameters:
n name to compare

Returns:
true if name n matches the name of this object

6.11.3.5 Object * Object::parent () const

Returns:
parent of this object

6.11.3.6 void Object::setName (std::string *name*)

Sets the object's name

Parameters:
name new name

6.11.3.7 void Object::setName (const std::string & *name*)

Sets the object's name

Parameters:
name new name

6.11.3.8 void Object::setParent (Object * *p*)

Sets the parent of tihs object

Parameters:
p new parent object

6.11.3.9 std::string Object::toString () const [virtual]

When any class exposed to a 'std::cout<<' this method will be called and the std::string returned by this method will be printed to stdout.

Returns:
debug information

Reimplemented in **Algorithm** (p. 10), **Polynomial** (p. 47), **Device** (p. 18), **InputNode** (p. 29), **Node** (p. 36), **OutputNode** (p. 42), and **Environment** (p. 21).

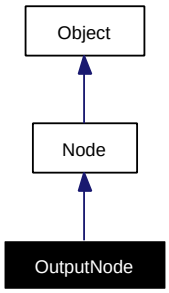
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**fobject.h**
- /home/engin/kdev/libpense/pense/**object.h**
- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**fobject.cpp**
- /home/engin/kdev/libpense/pense/**object.cpp**

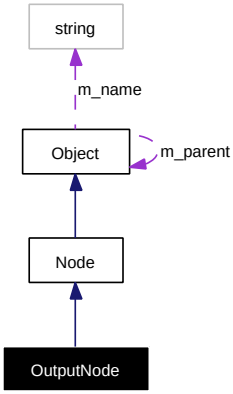
6.12 OutputNode Class Reference

`#include <outputnode.h>`

Inheritance diagram for OutputNode:



Collaboration diagram for OutputNode:



Public Member Functions

- **OutputNode** (const std::string &a="Default Output **Node**", Algorithm::Algorithm *parent=0L)
- ~**OutputNode** ()
- bool **connect** (**InputNode** *n)
- bool **connect** (**Node** *n, bool recursive)
- bool **disconnect** (**InputNode** *n)
- void **omit** (double v)
- virtual std::string **toString** () const

6.12.1 Detailed Description

OutputNode object is basically a node that can only emit (send) signals but not receive (omit) them.

In a **Device**(p. 12), OutputNode represents an algorithms output. For instance, equation $\sin(x)\cos(y)$. This equation can be represented with two **InputNode**(p. 26) objects and one OutputNode object. The result of this algorithm is emitted by this OutputNode object.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 OutputNode::OutputNode (const std::string & *a* = "Default Output Node", Algorithm::Algorithm * *parent* = 0L)

Parameters:

a algorithms name which we're representing

parent algorithm which we're representing

6.12.2.2 OutputNode::~OutputNode ()

Frees resources and disconnects itself from connected nodes.

6.12.3 Member Function Documentation

6.12.3.1 bool OutputNode::connect (Node * *n*, bool *recursive*) [virtual]

Reimplemented connect method which does runtime type identity checking with dynamic_cast.

Parameters:

n node to be connected to

recursive if true **Node**(p. 34) n also connects itself to this

Returns:

true on success

Reimplemented from **Node** (p. 35).

6.12.3.2 bool OutputNode::connect (InputNode * n)

Parameters:
n **InputNode**(p. 26) to be connected to

Returns:
true on success

6.12.3.3 bool OutputNode::disconnect (InputNode * n)

Parameters:
n **InputNode**(p. 26) to be disconnected from

Returns:
true on success

6.12.3.4 void OutputNode::omit (double v) [virtual]

An OutputNode should never omit a signal, remember that output nodes are supposed to be one way, i.e. they only emit signals not omit them.

Parameters:
v

Implements **Node** (p. 36).

6.12.3.5 std::string OutputNode::toString () const [virtual]

Returns:
debug information

Reimplemented from **Node** (p. 36).

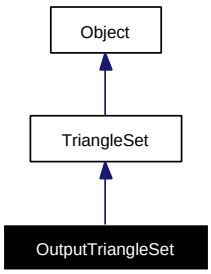
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/devices/**outputnode.h**
- /home/engin/kdev/libpense/pense/devices/**outputnode.cpp**

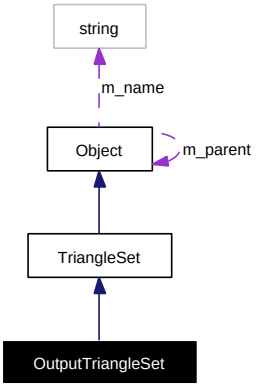
6.13 OutputTriangleSet Class Reference

`#include <outputtriangleset.h>`

Inheritance diagram for OutputTriangleSet:



Collaboration diagram for OutputTriangleSet:



Public Member Functions

- **OutputTriangleSet** (const std::string &name)
- ~**OutputTriangleSet** ()
- double **defuzify** (const **FuzzyValueList** &degrees)

6.13.1 Detailed Description

Fuzzy Output Set, which includes fuzzy triangles for output. This class defuzifies the fuzified value and gave a crisp output.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 OutputTriangleSet::OutputTriangleSet (const std::string & *name*)

Creates a output triangle set

Parameters:
name name of the set

6.13.2.2 OutputTriangleSet::~OutputTriangleSet ()

Frees resources

6.13.3 Member Function Documentation

6.13.3.1 double OutputTriangleSet::defuzify (const FuzzyValueList & *degrees*)

Gives a crisp output.

This method can raise a std::runtime_error expection if no output triangles are matched to input triangles.

Parameters:
degrees is the list of fuzified input(s)

Returns:
The crisp output value

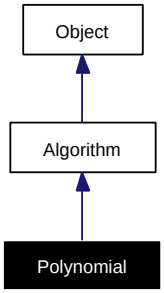
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**outputtriangleset.h**
- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**outputtriangleset.cpp**

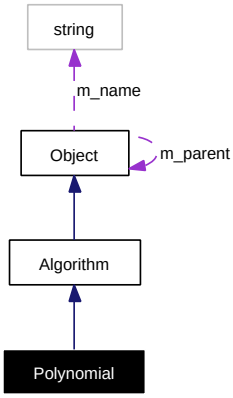
6.14 Polynomial Class Reference

`#include <polynomial.h>`

Inheritance diagram for Polynomial:



Collaboration diagram for Polynomial:



Public Member Functions

- **Polynomial** (const std::string &eq, const std::string &name="Default **Polynomial Algorithm**", Object *p=NULL)
- ~**Polynomial** ()
- double **evaluate** ()
- double **evaluate** (int count, char **names, double *values)
- double **evaluate_x** (double x)
- double **evaluate_x_y** (double x, double y)
- double **evaluate_x_y_z** (double x, double y, double z)
- std::string **equation** () const
- virtual std::string **toString** () const

Protected Member Functions

- void **parse_parameters** ()

Protected Attributes

- void * **m_e**

6.14.1 Detailed Description

This class holds an equation parsed by libmatheval.

```
#include <polynomial.h>
#include <stdexcept>
#include <iostream>

using namespace std;

int main( void )
{
	try
	{
		// Create an Polynomial object for equation "x*y*z"
		PENSE::Polynomial p( "x*y*z" );

		// Evaluate the equation for values of x = 1, y = 2, z = 3
		cout << p.evaluate_x_y_z( 1, 2, 3 ) << endl;

		// See if a variable name exists or not
		cout << "x exist: " << ( p.contains( "x" ) ? "yes" : "no" ) << endl; // this will print yes
		cout << "k exist: " << ( p.contains( "k" ) ? "yes" : "no" ) << endl; // this will print no

		// Assign variable values
		p.assign( "x", 2 );
		p.assign( "y", 3 );
		p.assign( "z", 4 );

		// Evaluate equation with the new assigned values
		cout << p.evaluate() << endl;

		// Here's the parameter list of the equation, in this case; x, y, z
		PENSE::Polynomial::ConstParameterList pl = p.parameters();

		// iterator which has both variable name and the very current value
		PENSE::Polynomial::ParameterList::const_iterator it;

		// Evaluate the equation for values of x = 1, y = 2, z = 3 again
		cout << p.evaluate_x_y_z( 1, 2, 3 ) << endl;

		// And iterate through the parameter list which we've fetched above. Note that it'll print
		// x = 1
		// y = 2
		// z = 3
		// Because evaluate_x, evaluate_x_y and evaluate_x_y_z update those values upon
		// evaluation

		for( it = pl.begin(); it != pl.end(); ++it )
		{
			cout << (*it).first << "=" << (*it).second << endl;
		}
	}
	// If we've provided an invalid equation like "x****y", this exception would be raised
	catch( std::invalid_argument const& e )
```

```
        {
                cerr << "Invalid equation:" << e.what() << endl;
        }
        return 0;
}
```

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Polynomial::Polynomial (const std::string & *eq*, const std::string & *name* = "Default Polynomial Algorithm", Object * *p* = NULL)

Creates a equation object from given equation

Parameters:

eq equation

name name of the polynomial

p parent object

6.14.2.2 Polynomial::~Polynomial ()

Destroys the matheval object and frees resources

6.14.3 Member Function Documentation

6.14.3.1 std::string Polynomial::equation () const

Returns:

the current equation in string representation

6.14.3.2 double Polynomial::evaluate (int *count*, char ** *names*, double * *values*)

Parameters:

count number of parameters you're passing

names parameter list

values value list

Returns:

evaluated result of the equation

6.14.3.3 double Polynomial::evaluate () [virtual]

Returns:

the result of the equation with the current parameter values

Implements **Algorithm** (p. 10).

6.14.3.4 double Polynomial::evaluate_x (double *x*)

This function will calculate the equation for the value of x provided by user. The value of "x" will be updated in the parameter list also. Which means;

```
myPolynomial.assign( "x", 3 );
cout << myPolynomial.evaluate() << endl;
// is same with
cout << myPolynomial.evaluate_x( 3 ) << endl;
```

Parameters:
x value of the parameter x

Returns:
evaluated result of the equation

6.14.3.5 double Polynomial::evaluate_x_y (double *x*, double *y*)

See also:
evaluate_x(p. 47)

Parameters:
x value of the parameter x
y value of the parameter y

Returns:
evaluated result of the equation

6.14.3.6 double Polynomial::evaluate_x_y_z (double *x*, double *y*, double *z*)

See also:
evaluate_x(p. 47)

Parameters:
x value of the parameter x
y value of the parameter y
z value of the parameter z

Returns:
evaluated result of the equation

6.14.3.7 void Polynomial::parse_parameters () [protected]

Parses parameters of an equation, for example x∗(y^z) has three parameters; x, y, z.

6.14.3.8 std::string Polynomial::toString () const [virtual]

Returns:
debug information

Reimplemented from **Algorithm** (p. 10).

6.14.4 Member Data Documentation

6.14.4.1 void* Polynomial::m_e [protected]

This holds the matheval object

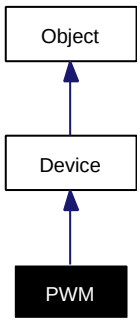
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/**polynomial.h**
- /home/engin/kdev/libpense/pense/algorithms/**polynomial.cpp**

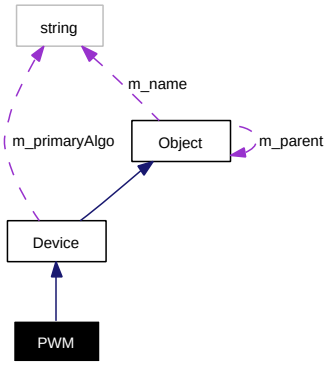
6.15 PWM Class Reference

`#include <pwm.h>`

Inheritance diagram for PWM:



Collaboration diagram for PWM:



Public Member Functions

- **PWM** (const std::string &name="Default **PWM** Controller", Environment *e=0L)
- ~**PWM** ()
- void **setCycles** (uint c)
- uint **cycles** () const
- void **setAlgorithm** (Algorithm::Algorithm *a)
- Algorithm::Algorithm * **algorithm** ()
- void **process** ()

6.15.1 Detailed Description

PWM - Pulse Width Modulation This device is supposed to provide a pulse width modulation. Which means supplying full power to the plants but only in a certain amount of time.

Say, apply full power to a motor, but only 50% of the time. With this approach torque of the motor is still enough, in contrast to voltage regulating.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 PWM::PWM (const std::string & *name* = "Default PWM Controller", Environment * *e* = 0L)

6.15.2.2 PWM::~PWM ()

Frees resources

6.15.3 Member Function Documentation

6.15.3.1 Algorithm::Algorithm * PWM::algorithm ()

Returns:
curreent acceleration algorithm

6.15.3.2 uint PWM::cycles () const

Returns:
current cycles percentage

6.15.3.3 void PWM::process () [virtual]

Processes the device for one iteration.

Can throw std::logic_error if frequency of the system is less than 100Hz

Reimplemented from **Device** (p. 16).

6.15.3.4 void PWM::setAlgorithm (Algorithm::Algorithm * *a*)

Sets acceleration algorithm.

PWM Controller will increase/decrease it's output frequency, according to this algorithm. **Algorithm**(p. 8) must use "input" parameter name as input. Also note that the name of this algorithm will be changed in this device.

6.15.3.5 void PWM::setCycles (uint *c*)

Parameters:
c cycles to be set

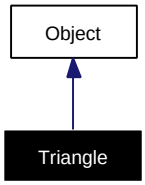
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/devices/controllers/**pwm.h**
- /home/engin/kdev/libpense/pense/devices/controllers/**pwm.cpp**

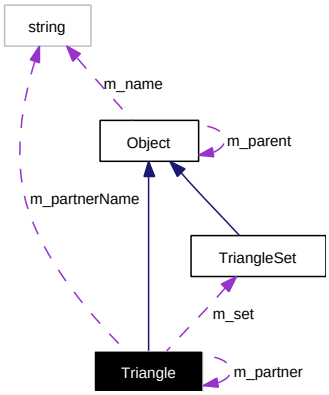
6.16 Triangle Class Reference

`#include <triangle.h>`

Inheritance diagram for Triangle:



Collaboration diagram for Triangle:



Public Member Functions

- **Triangle** (double start, double end, const std::string &name="default triangle name", const std::string &partner="default triangle partner")
- ~**Triangle** ()
- const **TriangleSet** * **of** (void)
- void **setPartner** (const std::string &name)
- const std::string & **partner** (void) const
- void **setSet** (const **TriangleSet** *set)
- double **start** (void) const
- double **end** (void) const
- double **middle** (void) const
- double **width** (void) const
- void **setStart** (double start)
- void **setEnd** (double end)
- void **setMiddle** (double middle)
- void **incMiddle** (double x)
- void **decMiddle** (double x)

- void **setWidth** (double width)
- void **expand** (double x)
- void **shrink** (double x)
- double **calcArea** (double height)
- double **area** (void) const
- double **degree** (double v) const
- bool **contains** (double v) const
- void **printInfo** () const

Protected Member Functions

- void **update** ()

6.16.1 Detailed Description

This class represents a fuzzy triangle which is defined by it's starting, ending points and partner triangle.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Triangle::Triangle (double *start*, double *end*, const std::string & *name* = "default triangle name", const std::string & *partner* = "default triangle partner")

Contructs a fuzzy triangle.

Parameters:

start start point coordinate

end end point coordinate

name name of the triangle

partner name of the partner triangle

6.16.2.2 Triangle::~Triangle ()

Destructs the object.

6.16.3 Member Function Documentation

6.16.3.1 double Triangle::area (void) const [inline]

Tells the area of the triangle.

Returns:

area of the triangle.

6.16.3.2 double Triangle::calcArea (double *height*)

Calculates the area (weight) of the triangle with provided height h.

Parameters:
height height

Returns:
the area

6.16.3.3 bool Triangle::contains (double *v*) const

Parameters:
v value to be evaluted

Returns:
true if v is between start and end points of this triangle

6.16.3.4 void Triangle::decMiddle (double *x*)

Decreases middle point by x

6.16.3.5 double Triangle::degree (double *v*) const

This method tells you the membership degree of a value for a triangle.

Parameters:
v value to be evaluated

Returns:
the membership value

6.16.3.6 double Triangle::end (void) const `[inline]`

Tells ending point coordinate of the triangle

Returns:
ending point coordinate

6.16.3.7 void Triangle::expand (double *x*)

Expands the triangle and automagivally adjusts the starting and ending points' coordinates, keeping the niddle point coordinates constant ofcourse.

Parameters:
x expand x unit

6.16.3.8 void Triangle::incMiddle (double *x*)

Increase middle point by x

6.16.3.9 double Triangle::middle (void) const [inline]

Tells middle point cordinate of the triangle which is

(starting_point + ending_point) / 2

Returns:
middle point coordinate

6.16.3.10 const TriangleSet * Triangle::of (void)

Returns:
which triangle set this triangle belongs to

6.16.3.11 const std::string & Triangle::partner (void) const

Tells the name of the partner triangle.

Returns:
name of the partner triangle

6.16.3.12 void Triangle::printInfo () const

Prints debug information, not for production release.

6.16.3.13 void Triangle::setEnd (double *end*)

Sets the ending point coordinate of the triangle.

Parameters:
end ending point coordinate

6.16.3.14 void Triangle::setMiddle (double *middle*)

Sets the middle point coordinate. This method automagically adjusts the starting and ending points' coordinate as necessary.

Parameters:
middle point

6.16.3.15 void Triangle::setPartner (const std::string & *name*)

Sets the partner triangle. Partner triangle is the triangle which corresponds to one triangle in the other set which is input or output.

Parameters:
name

6.16.3.16 void Triangle::setSet (const TriangleSet * *set*)

Sets the triangle set which this triangle belongs to

Parameters:
set triangle set

6.16.3.17 void Triangle::setStart (double *start*)

Sets the starting point coordinate of the triangle.

Parameters:
start starting point coordinate

6.16.3.18 void Triangle::setWidth (double *width*)

Sets the width. This method automagically adjusts the starting and ending points' coordinate.

Parameters:
width width

6.16.3.19 void Triangle::shrink (double *x*)

Shrinks the triangle and automagically adjusts the starting and ending points' coordinates, keeping the middle point coordinate constant ofcourse.

Parameters:
x shrink x unit

6.16.3.20 double Triangle::start (void) const [inline]

Tells starting point coordinate of the triangle

Returns:
starting point coordinate

6.16.3.21 void Triangle::update () [protected]

Updates the positions, width etc.

6.16.3.22 double Triangle::width (void) const [inline]

Tells the width size of the triangle base.

Returns:
width of the triangle

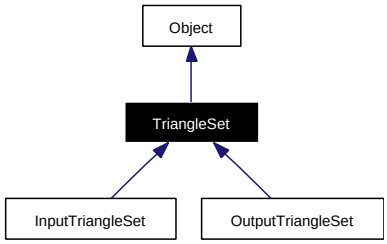
The documentation for this class was generated from the following files:

- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**triangle.h**
- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**triangle.cpp**

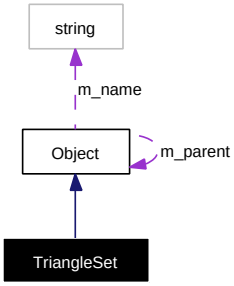
6.17 TriangleSet Class Reference

`#include <triangleset.h>`

Inheritance diagram for TriangleSet:



Collaboration diagram for TriangleSet:



Public Member Functions

- **TriangleSet** (const std::string &name)
- virtual ~**TriangleSet** ()
- virtual bool **add** (**Triangle** *triangle)
- virtual bool **remove** (const std::string &name)
- virtual **Triangle** * **get** (const std::string &name)
- bool **contains** (const std::string &name) const

Protected Attributes

- **TriangleList triangles**

6.17.1 Detailed Description

Fuzzy set, contains fuzzy triangles.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 TriangleSet::TriangleSet (const std::string & *name*)

This is the base class that will be used for object which will contain triangles.

6.17.2.2 TriangleSet::~TriangleSet () [virtual]

Destructor, deletes the list.

6.17.3 Member Function Documentation

6.17.3.1 bool TriangleSet::add (Triangle * *triangle*) [virtual]

Adds a triangle to the set.

Parameters:
triangle triangle to be added.

6.17.3.2 bool TriangleSet::contains (const std::string & *name*) const

Parameters:
name of the triangle to be searched

Returns:
true if triangle exists

6.17.3.3 Triangle * TriangleSet::get (const std::string & *name*) [virtual]

Returns the triangle named name

6.17.3.4 bool TriangleSet::remove (const std::string & *name*) [virtual]

Removes a triangle from the set.

Note: This method should tell if it was successful.

Parameters:
name of triangle to be removed.

Returns:
true on success

6.17.4 Member Data Documentation

6.17.4.1 TriangleList TriangleSet::triangles [protected]

triangle object container

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

- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**triangleset.h**
- /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/**triangleset.cpp**

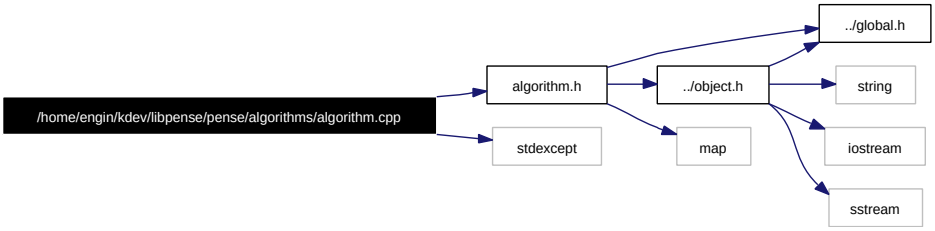
7 libpense.kdevelop File Documentation

7.1 /home/engin/kdev/libpense/pense/algorithms/algorithm.cpp File Reference

```
#include "algorithm.h"
```

```
#include <stdexcept>
```

Include dependency graph for algorithm.cpp:



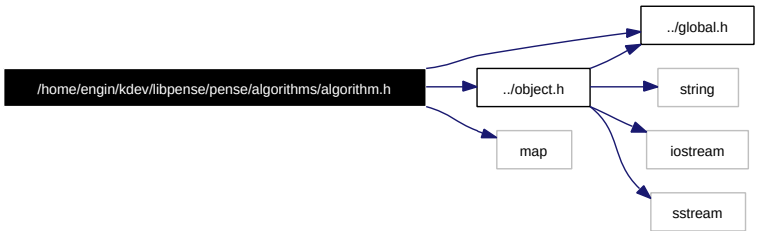
7.2 /home/engin/kdev/libpense/pense/algorithms/algorithm.h File Reference

```
#include "../global.h"
```

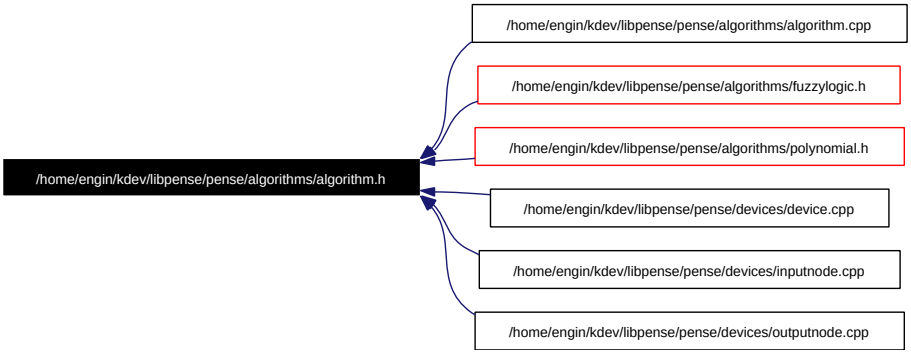
```
#include "../object.h"
```

```
#include <map>
```

Include dependency graph for algorithm.h:



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



Typedefs

- typedef std::map< std::string, double > **ParameterList**
- typedef const **ParameterList** & **ConstParameterList**

7.2.1 Typedef Documentation

7.2.1.1 typedef const ParameterList& ConstParameterList

Safe parameter list type that you can use in your programs.<

```
ConstParameterList pl = myAlgo.parameters();
```

is equal to:

```
const ParameterList &pl = myAlgo.parameters();
```

or

```
const std::map< std::string, double > &pl = myAlgo.parameters();
```

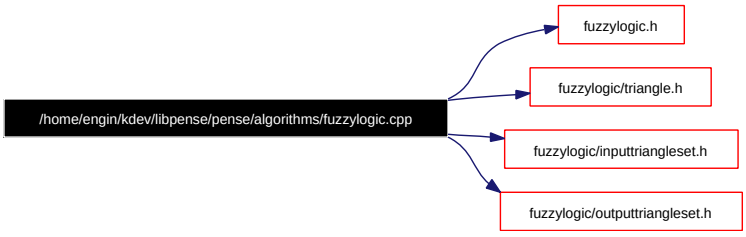
7.2.1.2 typedef std::map< std::string, double > ParameterList

Parameter list type

7.3 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic.cpp File Reference

```
#include "fuzzylogic.h"
#include "fuzzylogic/triangle.h"
#include "fuzzylogic/inputtriangleset.h"
#include "fuzzylogic/outputtriangleset.h"
```

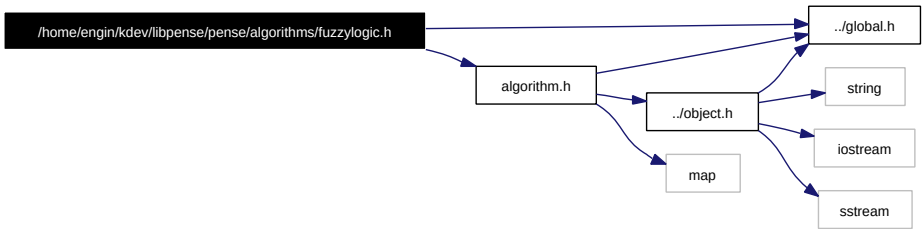
Include dependency graph for fuzzylogic.cpp:



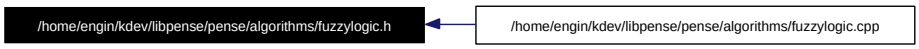
7.4 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic.h File Reference

```
#include "../global.h"
#include "algorithm.h"
```

Include dependency graph for fuzzylogic.h:



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



7.5 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/fobject.cpp File Reference

```
#include "fobject.h"
```

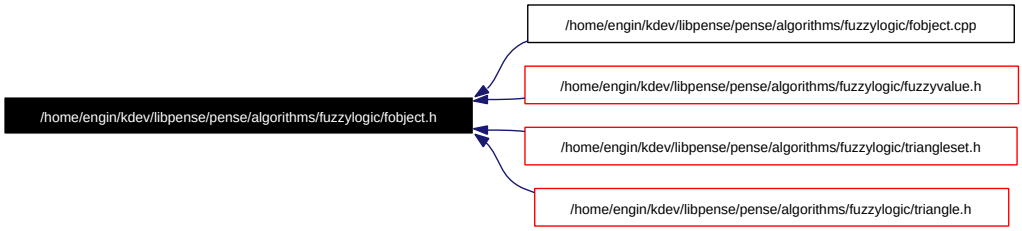
Include dependency graph for fobject.cpp:

7.6 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/fobject.h File Reference

```
#include "../../global.h"
#include <string>
```

Include dependency graph for fobject.h:

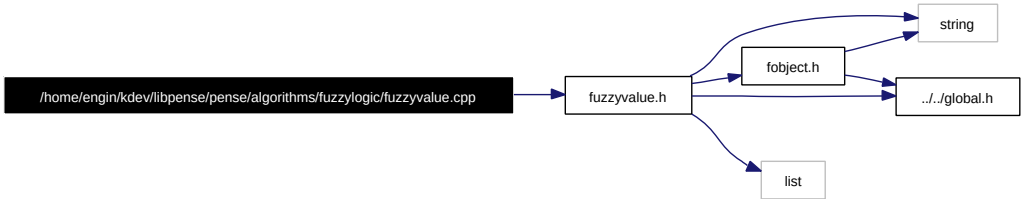
This graph shows which files directly or indirectly include this file:



7.7 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/fuzzyvalue.cpp File Reference

```
#include "fuzzyvalue.h"
```

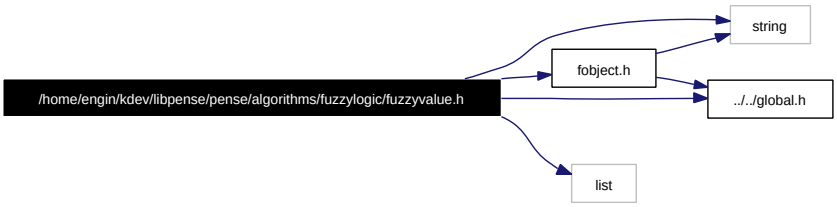
Include dependency graph for fuzzyvalue.cpp:



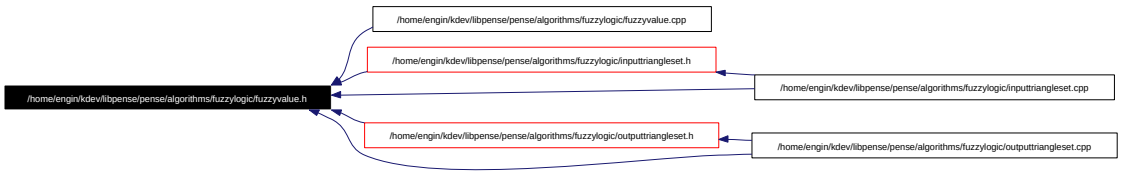
7.8 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/fuzzyvalue.h File Reference

```
#include <string>
#include "../../global.h"
#include "fobject.h"
#include <list>
```

Include dependency graph for fuzzyvalue.h:



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



Typedefs

- typedef std::list< **FuzzyValue** > **FuzzyValueList**

7.8.1 Typedef Documentation

7.8.1.1 typedef std::list<FuzzyValue> FuzzyValueList

7.9 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/inputtriangleset.cpp File Reference

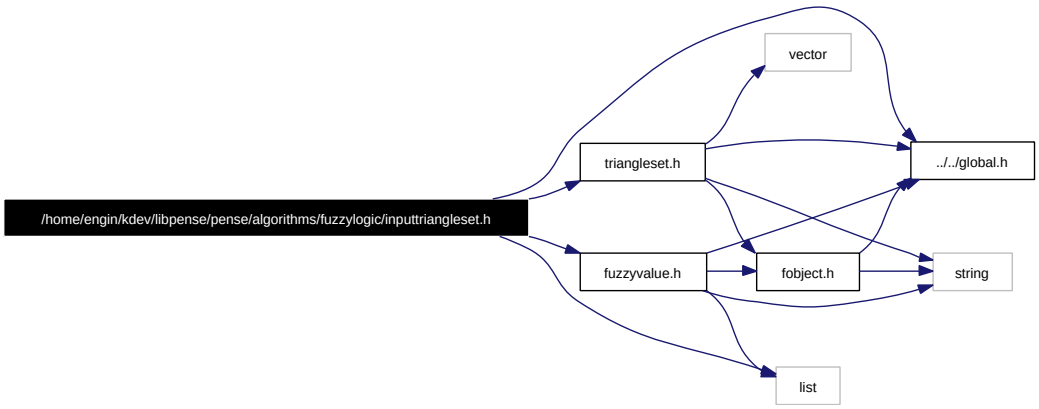
```
#include "inputtriangleset.h"
#include "triangle.h"
#include "fuzzyvalue.h"
```

Include dependency graph for inputtriangleset.cpp:

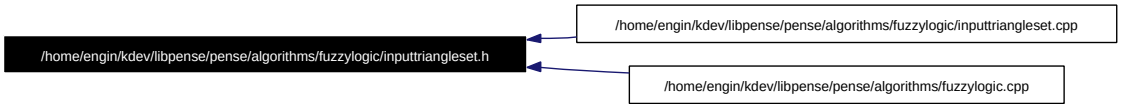
7.10 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/inputtriangleset.h File Reference

```
#include "../../global.h"
#include "triangleset.h"
#include "fuzzyvalue.h"
#include <list>
```

Include dependency graph for inputtriangleset.h:



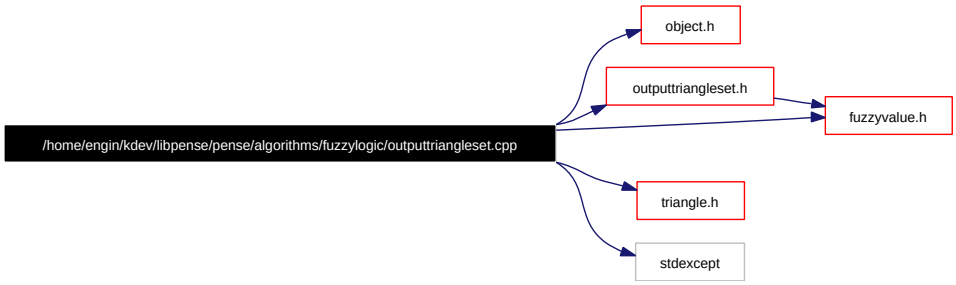
This graph shows which files directly or indirectly include this file:



7.11 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/outputtriangleset.cpp File Reference

```
#include "object.h"
#include "outputtriangleset.h"
#include "fuzzyvalue.h"
#include "triangle.h"
#include <stdexcept>
```

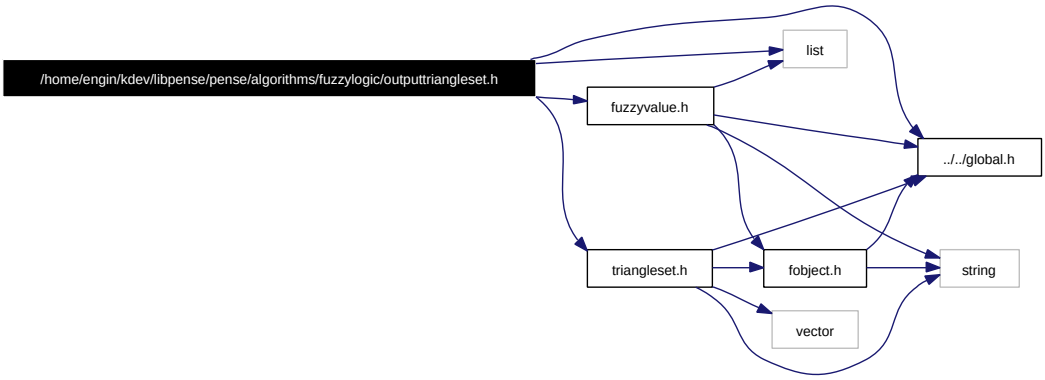
Include dependency graph for outputtriangleset.cpp:



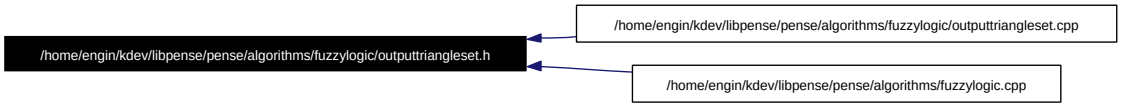
7.12 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/outputtriangleset.h File Reference

```
#include <list>
#include "../../global.h"
#include "triangleset.h"
#include "fuzzyvalue.h"
```

Include dependency graph for outputtriangleset.h:



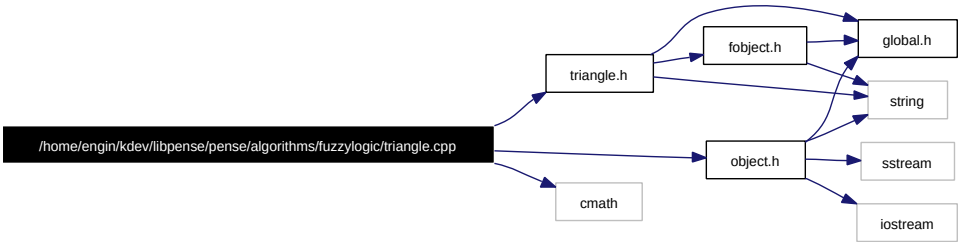
This graph shows which files directly or indirectly include this file:



7.13 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/triangle.cpp File Reference

```
#include "object.h"
#include "triangle.h"
#include <cmath>
```

Include dependency graph for triangle.cpp:

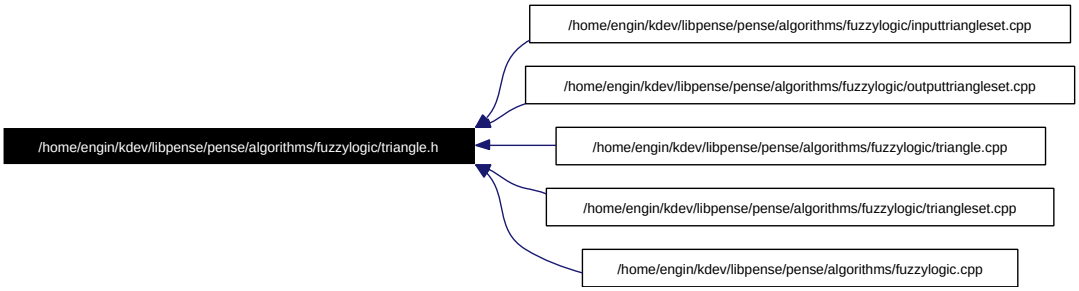


7.14 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/triangle.h File Reference

```
#include <string>
#include "../../global.h"
#include "fobject.h"
```

Include dependency graph for triangle.h:

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



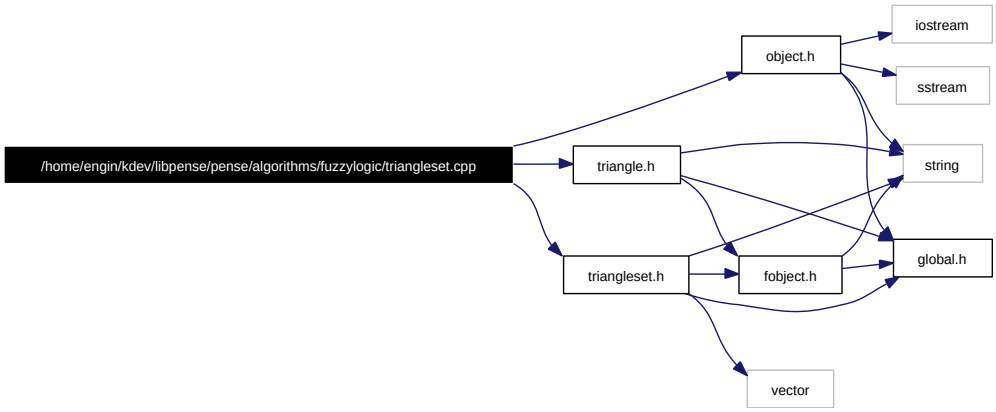
7.15 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/triangleset.cpp File Reference

```
#include "object.h"
```

```
#include "triangle.h"
```

```
#include "triangleset.h"
```

Include dependency graph for triangleset.cpp:



7.16 /home/engin/kdev/libpense/pense/algorithms/fuzzylogic/triangleset.h File Reference

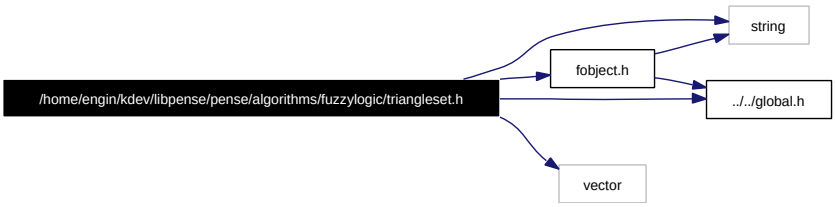
```
#include <string>
```

```
#include "../../global.h"
```

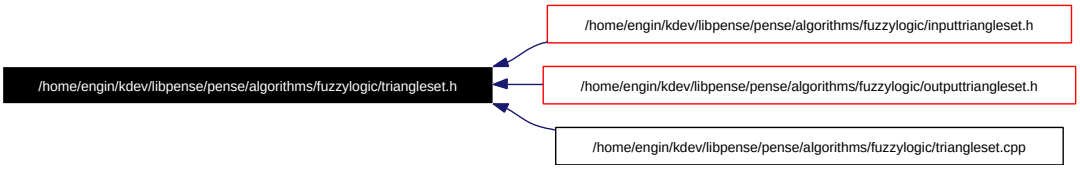
```
#include "fobject.h"
```

```
#include <vector>
```

Include dependency graph for triangleset.h:



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



Typedefs

- typedef std::vector< **Triangle** * > **TriangleList**

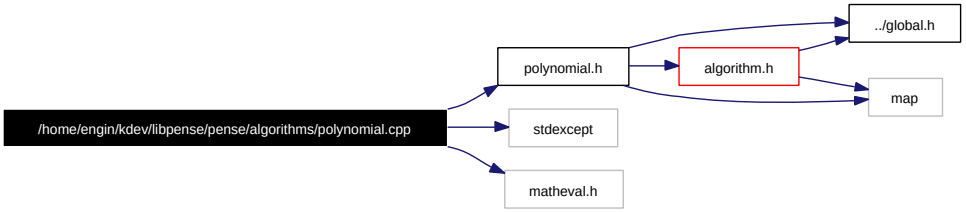
7.16.1 Typedef Documentation

7.16.1.1 typedef std::vector< Triangle* > TriangleList

7.17 /home/engin/kdev/libpense/pense/algorithms/polynomial.cpp File Reference

```
#include "polynomial.h"
#include <stdexcept>
#include <matheval.h>
```

Include dependency graph for polynomial.cpp:

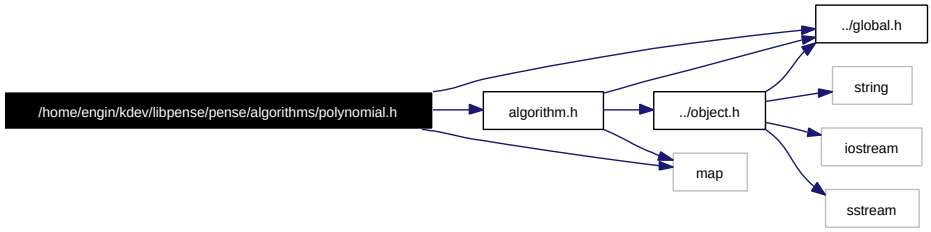


7.18 /home/engin/kdev/libpense/pense/algorithms/polynomial.h File Reference

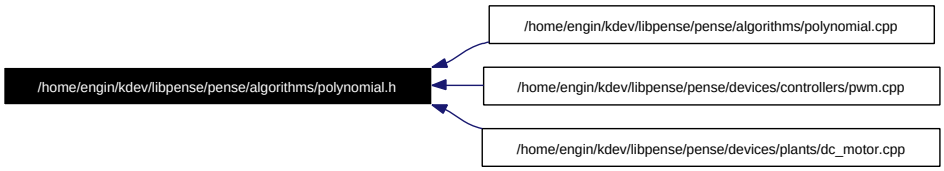
```
#include "../global.h"
#include "algorithm.h"
```

```
#include <map>
```

Include dependency graph for polynomial.h:



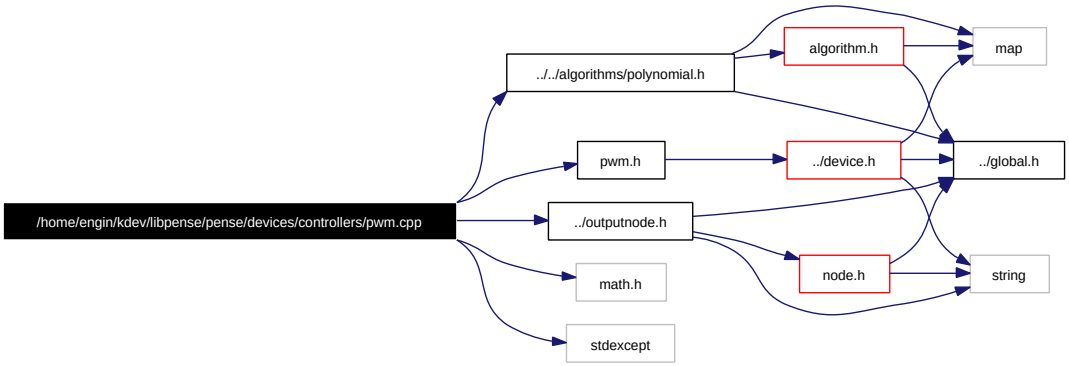
This graph shows which files directly or indirectly include this file:



7.19 /home/engin/kdev/libpense/pense/devices/controllers/pwm.cpp File Reference

```
#include "pwm.h"
#include "../../algorithms/polynomial.h"
#include "../outputnode.h"
#include <math.h>
#include <stdexcept>
```

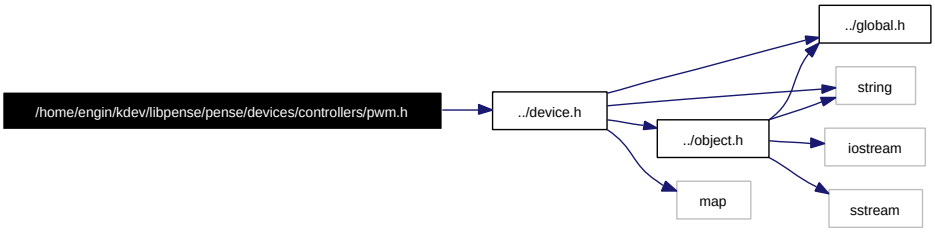
Include dependency graph for pwm.cpp:



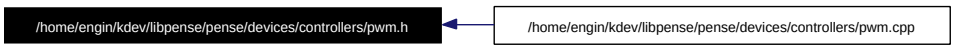
7.20 /home/engin/kdev/libpense/pense/devices/controllers/pwm.h File Reference

```
#include "../device.h"
```

Include dependency graph for pwm.h:



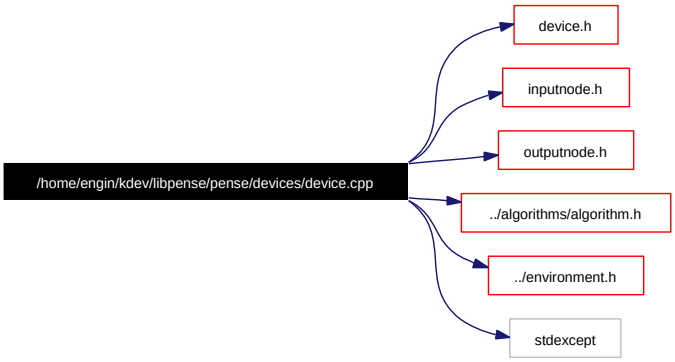
This graph shows which files directly or indirectly include this file:



7.21 /home/engin/kdev/libpense/pense/devices/device.cpp File Reference

```
#include "device.h"
#include "inputnode.h"
#include "outputnode.h"
#include "../algorithms/algorithm.h"
#include "../environment.h"
#include <stdexcept>
```

Include dependency graph for device.cpp:



Functions

- void **connect** (**Device** *src, const std::string &o, **Device** *dst, const std::string &i)
- void **disconnect** (**Device** *src, const std::string &o, **Device** *dst, const std::string &i)

7.21.1 Function Documentation

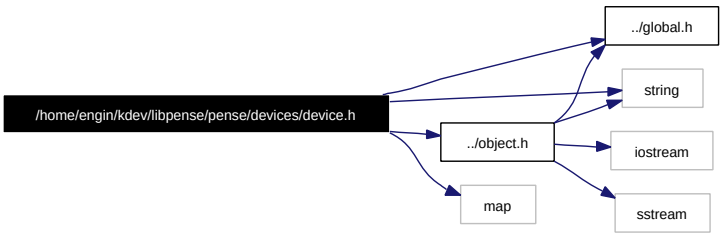
7.21.1.1 void connect (Device * *src*, const std::string & *o*, Device * *dst*, const std::string & *i*)

7.21.1.2 void disconnect (Device * *src*, const std::string & *o*, Device * *dst*, const std::string & *i*)

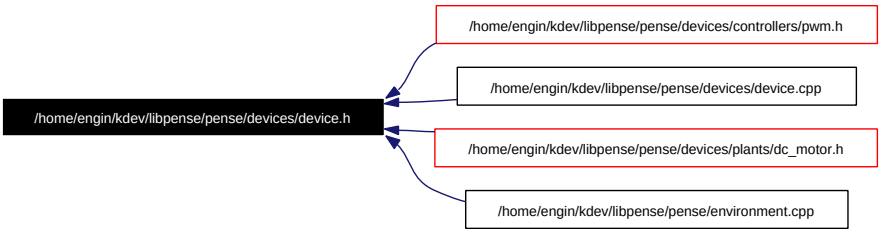
7.22 /home/engin/kdev/libpense/pense/devices/device.h File Reference

```
#include "../global.h"
#include "../object.h"
#include <string>
#include <map>
```

Include dependency graph for device.h:



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



Typedefs

- typedef std::multimap< std::string, **InputNode** * > **InputNodeList**

  InputNode(p. 26) *list type for internal usage.*

- typedef std::map< std::string, **OutputNode** * > **OutputNodeList**

  OutputNode(p. 40) *list type for internal usage.*

- typedef std::map< std::string, Algorithm::Algorithm * > **AlgorithmList**

  Algorithm(p. 8) *list type for internal usage.*

- typedef std::pair< std::string, Algorithm::Algorithm * > **AlgorithmEntry**
  Element of AlgorithmList.
- typedef std::pair< std::string, **InputNode** * > **InputNodeEntry**
  Element of InputNodeList.
- typedef std::pair< std::string, **OutputNode** * > **OutputNodeEntry**
  Element of OutputNodeList.

Functions

- void **connect** (**Device** *src, const std::string &o, **Device** *dst, const std::string &i)
- void **disconnect** (**Device** *src, const std::string &o, **Device** *dst, const std::string &i)

7.22.1 Typedef Documentation

7.22.1.1 typedef std::pair< std::string, Algorithm::Algorithm* > AlgorithmEntry

Element of AlgorithmList.

7.22.1.2 typedef std::map< std::string, Algorithm::Algorithm* > AlgorithmList

Algorithm(p. 8) list type for internal usage.

7.22.1.3 typedef std::pair< std::string, InputNode* > InputNodeEntry

Element of InputNodeList.

7.22.1.4 typedef std::multimap< std::string, InputNode* > InputNodeList

InputNode(p. 26) list type for internal usage.

7.22.1.5 typedef std::pair< std::string, OutputNode* > OutputNodeEntry

Element of OutputNodeList.

7.22.1.6 typedef std::map< std::string, OutputNode* > OutputNodeList

OutputNode(p. 40) list type for internal usage.

7.22.2 Function Documentation

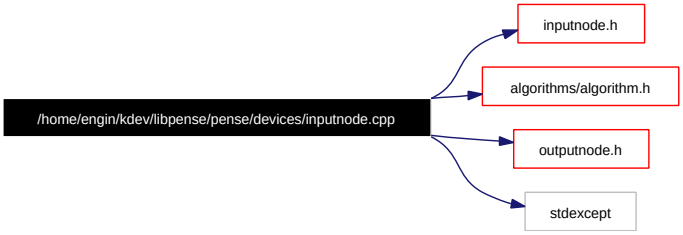
7.22.2.1 void connect (Device * *src*, const std::string & *o*, Device * *dst*, const std::string & *i*)

7.22.2.2 void disconnect (Device * *src*, const std::string & *o*, Device * *dst*, const std::string & *i*)

7.23 /home/engin/kdev/libpense/pense/devices/inputnode.cpp File Reference

```
#include "inputnode.h"
#include "algorithms/algorithm.h"
#include "outputnode.h"
#include <stdexcept>
```

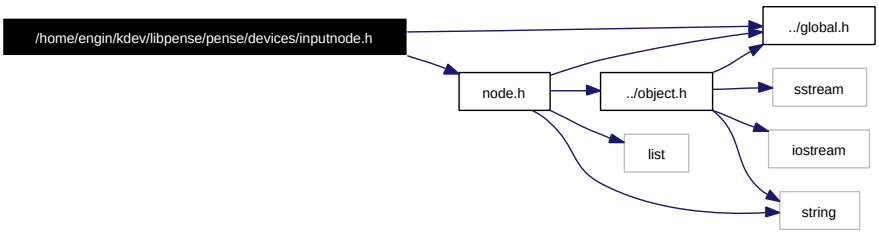
Include dependency graph for inputnode.cpp:



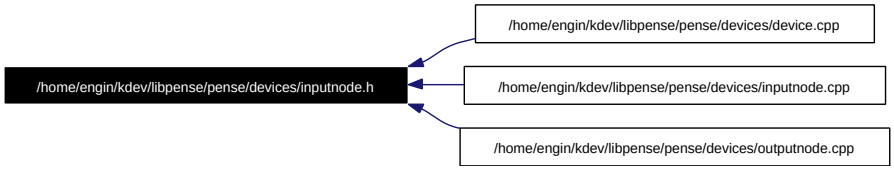
7.24 /home/engin/kdev/libpense/pense/devices/inputnode.h File Reference

```
#include "../global.h"
#include "node.h"
```

Include dependency graph for inputnode.h:



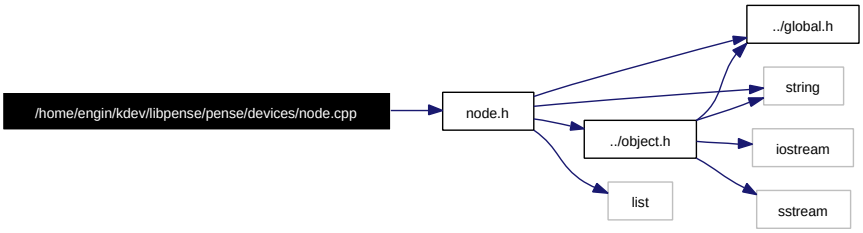
This graph shows which files directly or indirectly include this file:



7.25 /home/engin/kdev/libpense/pense/devices/node.cpp File Reference

```
#include "node.h"
```

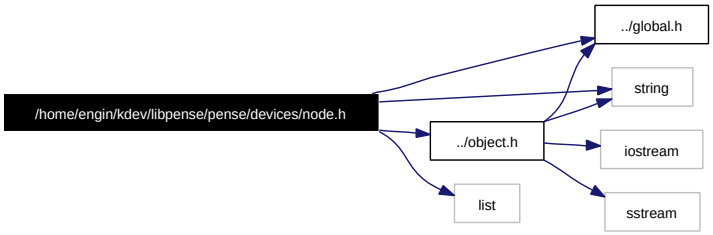
Include dependency graph for node.cpp:



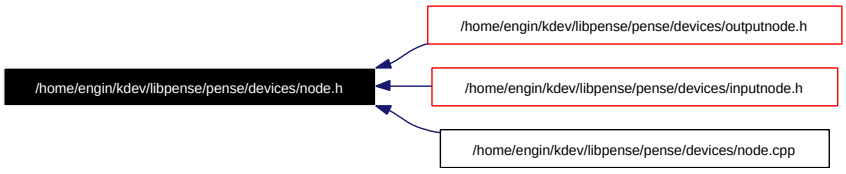
7.26 /home/engin/kdev/libpense/pense/devices/node.h File Reference

```
#include "../global.h"
#include "../object.h"
#include <string>
#include <list>
```

Include dependency graph for node.h:



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



Typedefs

- typedef std::list< **Node** * > **NodeList**
- typedef const **NodeList** & **ConstNodeList**

7.26.1 Typedef Documentation

7.26.1.1 typedef const NodeList& ConstNodeList

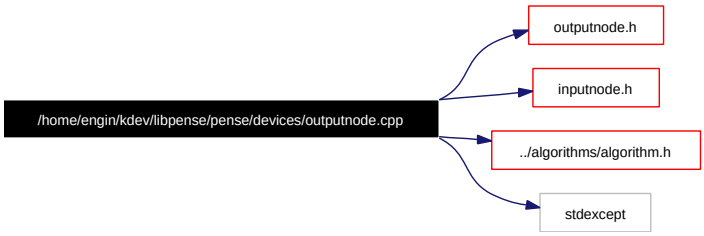
Safe node list type that you can use in your programs.

7.26.1.2 typedef std::list<Node*> NodeList

7.27 /home/engin/kdev/libpense/pense/devices/outputnode.cpp File Reference

```
#include "outputnode.h"
#include "inputnode.h"
#include "../algorithms/algorithm.h"
#include <stdexcept>
```

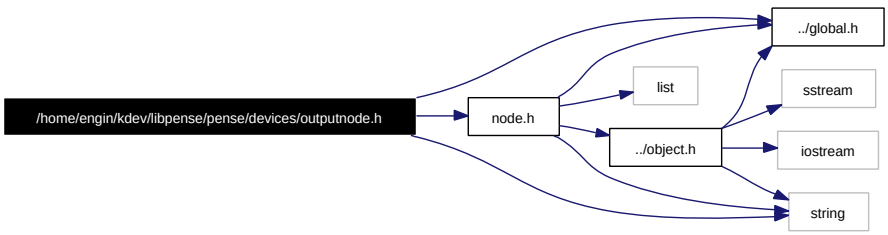
Include dependency graph for outputnode.cpp:



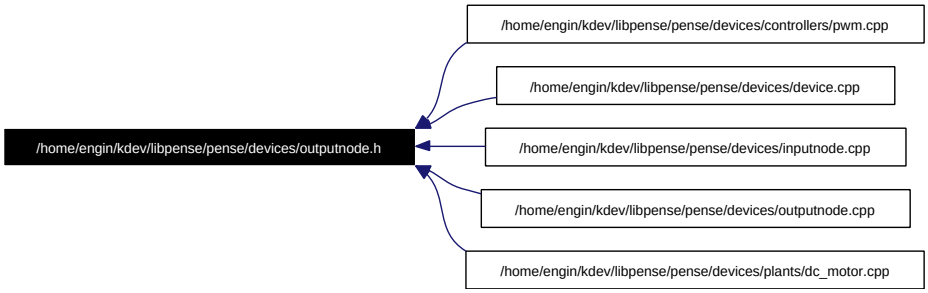
7.28 /home/engin/kdev/libpense/pense/devices/outputnode.h File Reference

```
#include "../global.h"
#include "node.h"
#include <string>
```

Include dependency graph for outputnode.h:



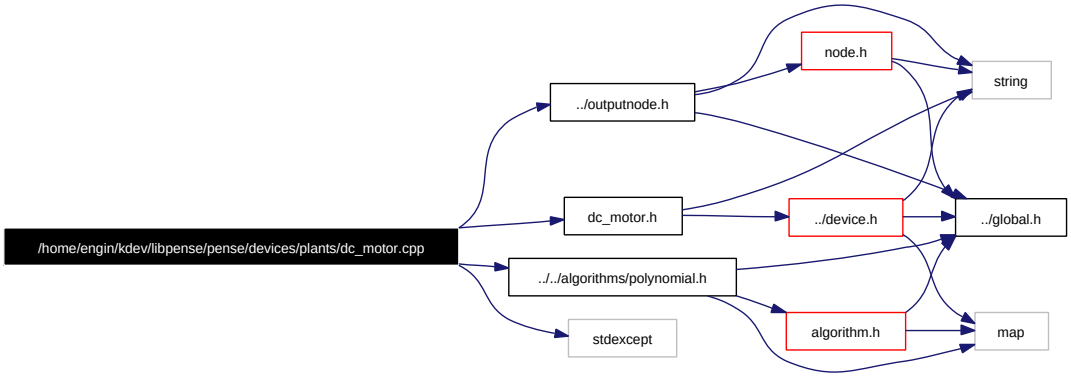
This graph shows which files directly or indirectly include this file:



7.29 /home/engin/kdev/libpense/pense/devices/plants/dc_motor.cpp File Reference

```
#include "dc_motor.h"
#include "../outputnode.h"
#include "../../algorithms/polynomial.h"
#include <stdexcept>
```

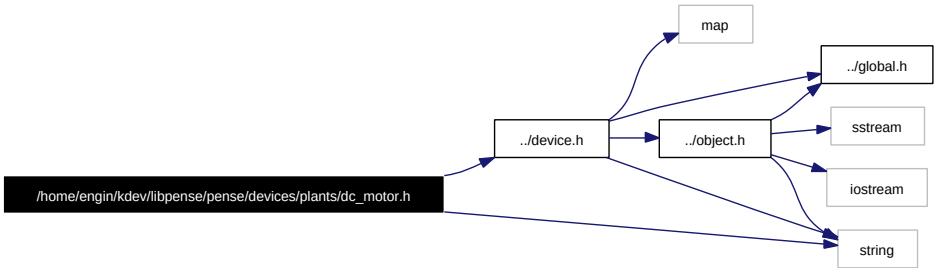
Include dependency graph for dc_motor.cpp:



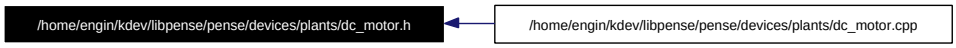
7.30 /home/engin/kdev/libpense/pense/devices/plants/dc_motor.h File Reference

```
#include "../device.h"
#include <string>
```

Include dependency graph for dc_motor.h:



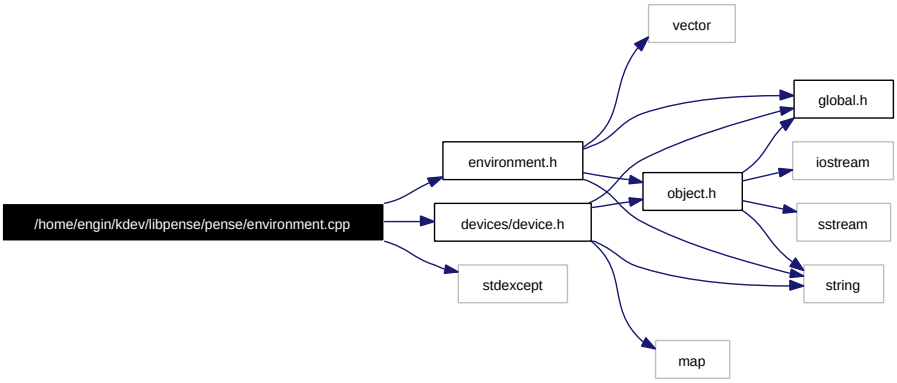
This graph shows which files directly or indirectly include this file:



7.31 /home/engin/kdev/libpense/pense/environment.cpp File Reference

```
#include "environment.h"
#include "devices/device.h"
#include <stdexcept>
```

Include dependency graph for environment.cpp:



Functions

- bool **compare** (Device::Device *d, const std::string &n)

7.31.1 Function Documentation

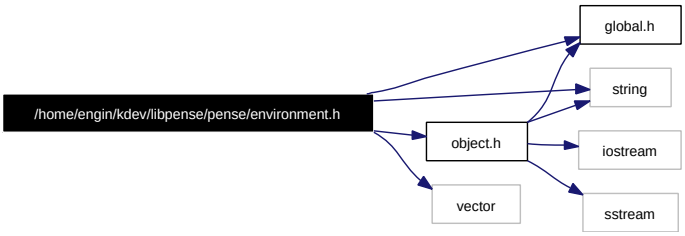
7.31.1.1 bool compare (Device::Device * *d*, const std::string & *n*)

7.32 /home/engin/kdev/libpense/pense/environment.h File Reference

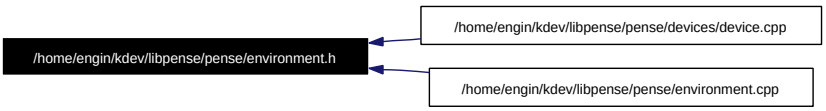
```
#include "global.h"
```

```
#include "object.h"
#include <string>
#include <vector>
```

Include dependency graph for environment.h:



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



Variables

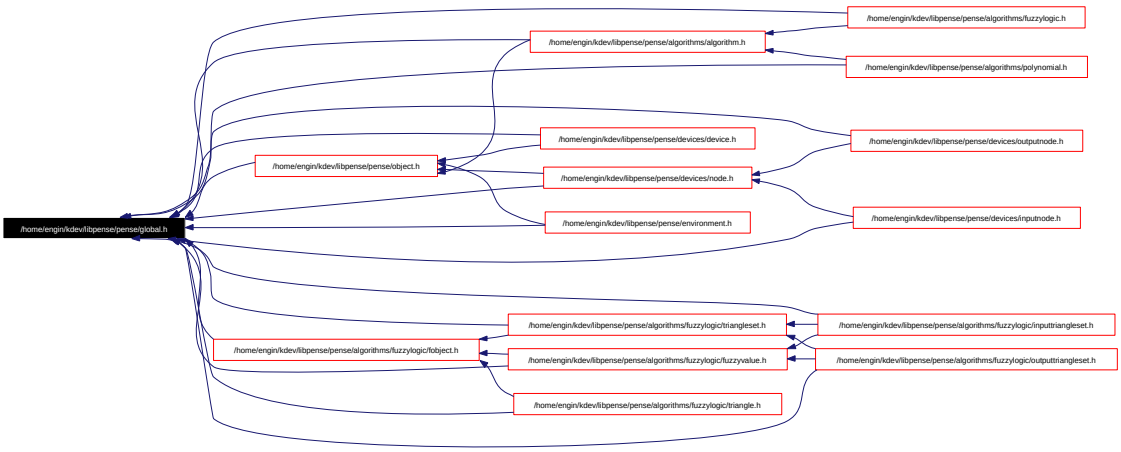
- NAMESPACE_END typedef std::vector< Device::Device * > **DeviceList**

7.32.1 Variable Documentation

7.32.1.1 NAMESPACE_END typedef std::vector< Device::Device*> DeviceList

7.33 /home/engin/kdev/libpense/pense/global.h File Reference

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



Defines

- #define **NAMESPACE_BEGIN**(x) namespace x {
- #define **NAMESPACE_END** }

7.33.1 Define Documentation

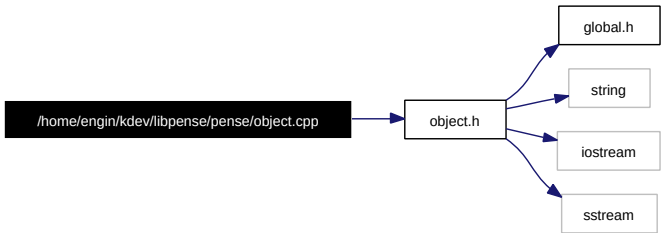
7.33.1.1 #define NAMESPACE_BEGIN(x) namespace x {

7.33.1.2 #define NAMESPACE_END }

7.34 /home/engin/kdev/libpense/pense/object.cpp File Reference

```
#include "object.h"
```

Include dependency graph for object.cpp:



Functions

- NAMESPACE_END std::ostream & **operator**<< (std::ostream &ostr, const PENSE::Object &obj)

7.34.1 Function Documentation

7.34.1.1 NAMESPACE_END std::ostream& operator<< (std::ostream & *ostr*, const PENSE::Object & *obj*)

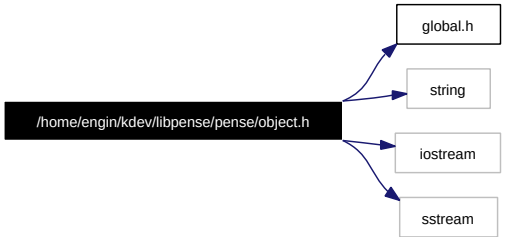
'std::cout <<' overloading. When a **Object**(p. 37) is exposed to std::cout, it's toString() method is called. For instance;

```
Object foo( "obj name", 0L );
std::cout << foo << std::endl;
// will print "[Object][obj name]" to the stdout
```

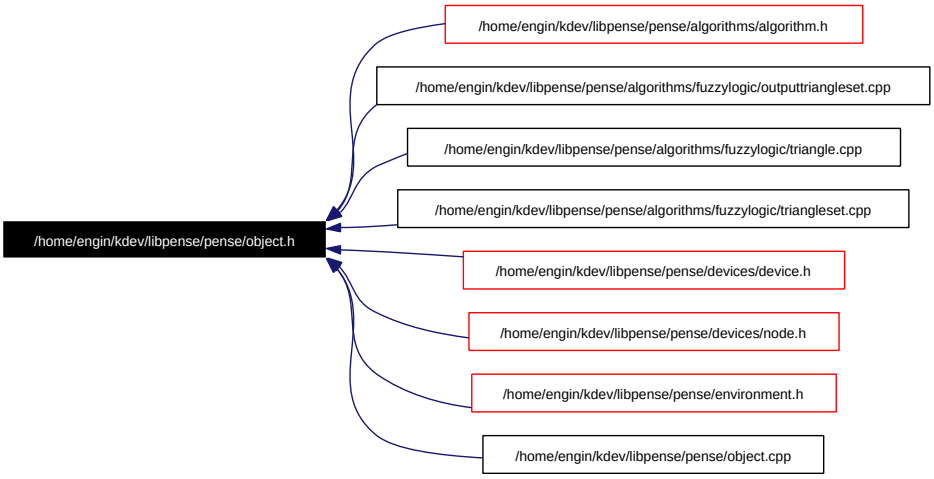
7.35 /home/engin/kdev/libpense/pense/object.h File Reference

```
#include "global.h"
#include <string>
#include <iostream>
#include <sstream>
```

Include dependency graph for object.h:



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



Classes

- class **Object**

Functions

- NAMESPACE_END std::ostream & **operator**<< (std::ostream &ostr, const PENSE::Object &obj)

7.35.1 Function Documentation

7.35.1.1 NAMESPACE_END std::ostream& operator<< (std::ostream & *ostr*, const PENSE::Object & *obj*)

'std::cout <<' overloading. When a **Object**(p. 37) is exposed to std::cout, it's toString() method is called. For instance;

```
Object foo( "obj name", 0L );
std::cout << foo << std::endl;
// will print "[Object][obj name]" to the stdout
```

Index